Preface

This volume contains the proceedings of the 28th international conference on the Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2008), organized under the auspices of the Indian Association for Research in Computing Science (IARCS).

This year's conference attracted 117 submissions. Each submission was reviewed by at least three independent referees. The final selection of the papers making up the programme was done through an electronic discussion on EasyChair, spanning two weeks, without a physical meeting of the Programme Committee (PC). All PC members participated actively in the discussion.

We have five invited speakers this year: Hubert Comon-Lundh, Uriel Feige, Erich Grädel, Simon Peyton Jones and Leslie Valiant. We thank them for having readily accepted our invitation to talk at the conference and for providing abstracts (and even full papers) for the proceedings.

We thank all the reviewers and PC members, without whose dedicated effort the conference would not be possible. We thank the Organizing Committee for making the arrangements for the conference.

This year, the conference is being held at the Indian Institute of Science, Bangalore, as part of its centenary year celebrations. It is a great honour and privilege for the conference to be recognized and associated with the institute on this occasion.

Finally, this year we have taken a decisive step in democratizing the conference by moving away from commercial publishers. Instead, we will be hosting the proceedings online, electronically, via the Dagstuhl Research Online Publication Server (DROPS). A complete copy of the proceedings will also be hosted on the FSTTCS website (www.fsttcs.org).

We do hope this direction will be sustained in the future.

December 2008

Ramesh Hariharan,
Madhavan Mukund,
V Vinay

FSTTCS 2008
IARCS Annual Conference on
Foundations of Software Technology and Theoretical Computer Science
http://drops.dagstuhl.de/opus/volltexte/2008/1771

Foundations of Software Technology and Theoretical Computer Science (Bangalore) 2008.
Editors: R. Hariharan, M. Mukund, V. Vinay; pp 1-12

Implicit Branching and Parameterized Partial Cover Problems (Extended Abstarct)

Omid Amini[1], Fedor V. Fomin[2] and Saket Saurabh[2]

[1] Max-Planck-Institut für Informatik
Omid.Amini@mpi-inf.mpg.de

[2] Department of Informatics, University of Bergen,
N-5020 Bergen, Norway.
{fedor.fomin|saket.saurabh}@ii.uib.no

ABSTRACT. Covering problems are fundamental classical problems in optimization, computer science and complexity theory. Typically an input to these problems is a family of sets over a finite universe and the goal is to cover the elements of the universe with as few sets of the family as possible. The variations of covering problems include well known problems like SET COVER, VERTEX COVER, DOMINATING SET and FACILITY LOCATION to name a few. Recently there has been a lot of study on partial covering problems, a natural generalization of covering problems. Here, the goal is not to cover all the elements but to cover the specified number of elements with the minimum number of sets.
In this paper we study partial covering problems in graphs in the realm of parameterized complexity. Classical (non-partial) version of all these problems have been intensively studied in planar graphs and in graphs excluding a fixed graph H as a minor. However, the techniques developed for parameterized version of non-partial covering problems cannot be applied directly to their partial counterparts. The approach we use, to show that various partial covering problems are fixed parameter tractable on planar graphs, graphs of bounded local treewidth and graph excluding some graph as a minor, is quite different from previously known techniques. The main idea behind our approach is the concept of *implicit branching*. We find implicit branching technique to be interesting on its own and believe that it can be used for some other problems.

1 Introduction

Covering problems are basic, fundamental and widely studied problems in algorithms and combinatorial optimizations. In general these problems ask for selecting a least sized family of sets to cover all the elements. One of the prominent covering problem is the classical SET COVER problem. SET COVER problem consists of a family $\mathscr{F}$ of sets over a universe $\mathscr{U}$ and the goal is to cover this universe $\mathscr{U}$ with the least number of sets from $\mathscr{F}$. Other classical problems in the framework of covering include well known problems like VERTEX COVER, DOMINATING SET, FACILITY LOCATION, k-MEDIAN, k-CENTER problems, on which hundreds of papers have been written.

In this paper we study the generalization of these problems to the *partial covering problems*, where the objective is not to cover all the elements but to cover the pre-specified number of elements with minimum number of objects. More precisely, in the partial covering

FSTTCS 2008
IARCS Annual Conference on
Foundations of Software Technology and Theoretical Computer Science
http://drops.dagstuhl.de/opus/volltexte/2008/1736

problem, for a given integer $t \geq 0$, we want to cover at least t elements rather than covering all the elements. For an example, in PARTIAL VERTEX COVER (PVC), the goal is to cover at least t edges with minimum number of vertices not all the edges while in PARTIAL SET COVER (PSC) the goal is to cover at least t elements of $\mathscr{U}$ with minimum number of sets from $\mathscr{F}$. Other problems are defined similarly. Partial covering problems are studied intensively not only because they generalize classical covering problems, but also because of many real life applications. They have received a lot of attention recently, see, for example [4, 5, 7, 18].

While different variations of PSC were studied intensively and many approximation algorithm and non-approximability results exist in the literature, only few things are known on their parameterized complexity. In this paper we fill this gap by initiating parameterized algorithmic study of these problems on structural graphs like planar graphs, graphs of bounded genus and graphs of bounded local treewidth. In parameterized algorithms, for decision problems with input size n, and a parameter k, the goal is to design an algorithm with runtime $\tau(k) \cdot n^{O(1)}$, where τ is a function of k alone. Problems having such an algorithm are said to be fixed parameter tractable (FPT). There is also a theory of hardness using which one can identify parameterized problems that are not amenable to such algorithms. This hardness hierarchy is represented by $W[i]$ for $i \geq 1$. For an introduction and more recent developments see the books [15, 17, 21]. In this paper, we always parameterize a problem by the size of the partial set cover, i.e. all our algorithms for finding a partial set cover of size k that cover at least t sets with input of size n are of running time $\tau(k) \cdot n^{O(1)}$.

Our Approach and Results. The main ideas behind our approach can be illustrated by planar instances of PARTIAL VERTEX COVER and PARTIAL DOMINATING SET. Let a planar graph $G = (V, E)$ on n vertices, and integers k, t, be an instance of PARTIAL VERTEX COVER. Let S be the set vertices in G of degree at least t/k. If S is sufficiently big, say, its size is at least $4k$, then (by the Four color theorem), the subgraph of G induced on S contains an independent set of size at least k. This yields that there are k vertices of S that are pairwise non-adjacent in G, and since each of these vertices covers at least t/k edges, we have that in total they cover at least t edges. If the size of S is less than $4k$, we apply *explicit branching*. The crucial observation here is that if G has a partial vertex cover of size at most k, then this cover must contain at least one vertex of S. Thus by making a guess on the vertices $x \in S$, whether x is in a partial vertex cover of size at most k, we can guarantee, that if the problem has a solution, then at least one of our guesses is correct. For each of the guesses x, we create a new subproblem for PARTIAL VERTEX COVER, where the input is the subgraph of G induced on $V \setminus \{x\}$ and we are asked to cover $t - deg(x)$ edges by $k - 1$ vertices, where $deg(x)$ is the number of edges adjacent to x. The number of subproblems we generate in this way is at most $4k$, and we call the procedure recursively on each subproblem. The depth of the recursion is at most k, and the number of recursive calls at each steps is at most $4k$, resulting in total running time $(4k)^k \cdot n^{O(1)}$. Actually, in our arguments we used planarity only to conclude that a graph has large independent set. Definitely, this approach is valid for many other graph classes with large independent sets, like bipartite graphs, degenerate graphs and graphs excluding some graph as a minor. (We provide detailed consequences of this approach in Section 5.)

The main drawback of explicit branching is that we cannot use it for many partial cov-

ering problems, in particular for PARTIAL DOMINATING SET. Even for planar graphs, the existence of a large independent set of vertices of degree at least t/k does not imply that k vertices can dominate at least t vertices. To overcome this obstacle, we do the following. We start as in the case of PARTIAL VERTEX COVER, by selecting the set S consisting of vertices of degree at least t/k. If there are more than k vertices in S which are at distance at least three from each other, we have the solution. Otherwise, we know that at least one vertex from S should be in a partial dominating set but we cannot use explicit branching by trying all vertices of S because the size of S can be too large. However, we show in this case that the graph formed by S and their neighbors is of small diameter, and thus, by well known properties of planar graphs, has small treewidth. (Very loosely small here means bounded by some function of k.) In this case we apply *implicit branching*, which means that we do not create a new subproblem for every vertex of S, but instead for every i, $1 \leq i \leq k$, we make a guess that exactly i vertices of S are in a partial dominating set. Thus we branch on k cases and try to solve the problem recursively. We formulate these ideas in details in Sections 3.1 and 3.2 and show how it is sufficient to just know the size of an intersection of an optimal partial dominating set with S rather than the actual intersection itself to solve the problem.

Again, the only property of planar graphs we mentioned here was the property that non-existence of a large set of pairwise remote vertices in a graphs yields a small treewidth. But this property can be shown not only for planar graphs, but more generally for graphs of bounded local treewidth, the class of graphs containing planar graphs, graphs of bounded genus, graphs of bounded vertex degree, and graphs excluding an apex graph as a minor. With more additional work we show that similar ideas can be used to prove that much more general problem, namely a weighted version of the PARTIAL (k, r, t)-CENTER problem, where the goal is to cover at least t elements by balls of radius r centered around at most k vertices, is FPT on graphs of bounded local treewidth. This result can be found in Section 3.2. This is mainly of theoretical interest because the running time of the algorithm is $2^{k^{O(k)}} \cdot n^{O(1)}$. Such a huge running time is due to the bounds on the treewidth of a graph, which is used in implicit branching. Due to the generality of the result for graphs with bounded local treewidth, we do not see any reasonable way of overcoming this problem. But because of numerous application, we find it is worth to search for faster practical algorithms on subclasses of graphs of bounded local treewidth, in particular on planar graphs. As a step in this direction, we obtain much better combinatorial bounds on the treewidth of planar graphs in implicit branching, which results in algorithms of running time $2^{O(k)} \cdot n^{O(1)}$ on planar graphs. The combinatorial arguments used for the exponential speedup (Section 3.3) are interesting on their own. In Section 4, we show that the PARTIAL (k, r, t)-CENTER problem is FPT on graphs excluding a fixed graph as a minor. The proof of this result is based on the decompositions theorem of Robertson and Seymour from Graph Minors [24]. The algorithm is quite involved, it uses two levels of dynamic programming and two levels of implicit branching, and can be seen as a non-trivial extension of the algorithm of Demaine et al. [10] for classical covering problems to partial covering problems.

Finally, let us remark that while DOMINATING SET is FPT on d-degenerated graphs [3], there are strong arguments that our results cannot be extended to this class of sparse graphs. This is because Golovach and Villanger [19] showed that PARTIAL DOMINATING SET is W[1]-hard on d-degenerated graphs.

2 Preliminaries

Let $G = (V, E)$ be an undirected graph where V (or $V(G)$) is the set of vertices and E (or $E(G)$) is the set of edges. We denote the number of vertices by n and number of edges by m. For a subset $V' \subseteq V$, by $G[V']$ we mean the subgraph of G induced by V'. By $N(u)$ we denote (open) neighborhood of u that is set of all vertices adjacent to u and by $N[u] = N(u) \cup \{u\}$. Similarly, for a subset $D \subseteq V$, we define $N[D] = \cup_{v \in D} N[v]$. The *distance* $d_G(u, v)$ between two vertices u and v of G is the length of the shortest path in G from u to v. The *diameter* of a graph G, denoted by $diam(G)$, is defined to be the maximum length of a shortest path between any pair of vertices of $V(G)$. By an abuse of notation, we define diameter of a graph as the maximum of the diameters of its connected components. For $r \geq 0$, the *r-neighborhood* of a vertex $v \in V$ is defined as $N^r_G[v] = \{u \mid d_G(v, u) \leq r\}$. We also let $B_r(v) = N^r_G[v]$ and call it a ball of radius r around v. Similarly $B_r(A) = \cup_{v \in A} N^r_G[v]$ for $A \subseteq V(G)$. Given a weight function $w : V \to \mathbb{R}$ and $A \subseteq V(G)$, $w(B_r(A)) = \sum_{u \in B_r(A)} w(u)$.

Given an edge $e = (u, v)$ of a graph G, the graph G/e is obtained by contracting the edge (u, v) that is we get G/e by identifying the vertices u and v and removing all the loops and duplicate edges. A *minor* of a graph G is a graph H that can be obtained from a subgraph of G by contracting edges. A graph class $\mathcal{C}$ is *minor closed* if any minor of any graph in $\mathcal{C}$ is also an element of $\mathcal{C}$. A minor closed graph class $\mathcal{C}$ is *H-minor-free* or simply *H-free* if $H \notin \mathcal{C}$.

We use the standard definitions of *treewidth* and *tree decomposition*. We use $\mathbf{tw}(G)$ to denote the treewidth of a graph G. The definition of treewidth can be generalized to take into account the local properties of G and is called *local treewidth* [16, 20]. The *local treewidth* of a graph G is the function $\mathbf{ltw}^G : \mathbb{N} \to \mathbb{N}$ that associates with every integer $r \in \mathbb{N}$ the maximum treewidth of an r-neighborhood of vertices of G, i.e., $\mathbf{ltw}^G(r) = \max_{v \in V(G)} \{\mathbf{tw}(G[N^r_G[v]])\}$. A graph class $\mathcal{G}$ has *bounded local treewidth*, if there exists a function $f : \mathbb{N} \to \mathbb{N}$ such that for each graph $G \in \mathcal{G}$, and for each integer $r \in \mathbb{N}$, we have $\mathbf{ltw}^G(r) \leq f(r)$. The class $\mathcal{G}$ has *linear local treewidth*, if in addition the function f can be chosen to be linear, that is $f(r) = cr$ where $c \in \mathbb{R}$ is a constant. For a given function $f : \mathbb{N} \to \mathbb{N}$, $\mathcal{G}_f$ is the class of all graphs G of local tree-width at most f, that is, $\mathbf{ltw}^G(r) \leq f(r)$ for every $r \in \mathbb{N}$. A well known graph classes which are known to have bounded local treewidth are planar graphs, graphs of bounded genus, and graphs of bounded maximum degree. By a result of Robertson and Seymour [22], $f(r)$ can be chosen as $3r$ for planar graphs. Similarly Eppstein [16] showed that $f(r)$ can be chosen as $c_g g(\Sigma) r$ for graphs embeddable in a surface Σ, where $g(\Sigma)$ is the genus of the surface Σ and c_g is a constant depending only on the genus of the surface. Demaine and Hajiaghayi [11] extended this result and showed that the concept of bounded local treewidth and linear local treewidth are the same for minor closed families of graphs.

3 FPT Algorithms for Weighted Partial-(k, r, t)-Center Problem

3.1 Developing a Step by Step Procedure

In this section we give a template of a generic algorithm for partial covering problems arising on graphs. We use this later to show that partial covering problems arising on graphs are fixed parameter tractable in graphs of bounded local treewidth. We formulate the template

through the following problem.

> WEIGHTED PARTIAL-(k,r,t)-CENTER (WP-(k,r,t)-C): Given an undirected graph $G = (V,E)$, with weight function $w : V \to \{0,1\}$ and integers k, r and t. The problems asks whether there exists a $C \subseteq V$ of size at most k (k centers), such that $w(B_r(C)) \geq t$. Here k and r are the parameters.

When all the vertices have weight 1 this is a PARTIAL-(k,r,t)-CENTER (P-(k,r,t)-C) problem, and for $r = 1$ and $w(v) = 1$ for all $v \in V$ this is PARTIAL DOMINATING SET problem. To formulate PSC problem as WP-(k,r,t)-C problem, we consider the incidence bipartite graph associated with the instance of PSC problem and give weights 1 to the vertices associated with elements and 0 to the vertices associated with sets. Since PVC can be transformed to PSC problem, WP-(k,r,t)-C also generalizes PVC. One defines PARTIAL HITTING SET similarly.

Unlike the non-partial and non-weighted version of WP-(k,r,t)-C problem, the first major challenge in partial covering problems is: which t elements we choose to cover? To find an answer to this we define the following set S and the corresponding graph $\mathcal{G}$, which forms the first step of the algorithm:

(T1) Define $S = \{v \mid v \in V,\ w(B_r(v)) \geq t/k\}$ and $\mathcal{G} = \bigcup_{v \in S} G[B_r(v)]$.

The basic observation is that if there exists a subset $C \subseteq V$ of size at most k such that $w(B_r(C,r)) \geq t$ then $C \cap S \neq \emptyset$. Given the graph $\mathcal{G}$ our second idea is to:

(T2) Check the diameter of $\mathcal{G}$, and if $diam(\mathcal{G})$ is large then we argue that this is a YES instance by providing a subset C of size at most k and $w(B_r(C)) \geq t$.

Now when the $diam(\mathcal{G})$ is small, the treewidth of the graph $\mathcal{G}$ is bounded and hence dynamic programming over graphs with bounded treewidth can be used. But we still do not know whether we can find the desired C among the vertices of $\mathcal{G}$. Hence even if we find out that there is no $X \subseteq S$ such that $|X| \leq k$ and $w(B_r(X)) \geq t$, we can not guarantee that this is a NO instance of the problem. So to overcome this difficulty we resort to an implicit branching by using the earlier observation that there is no desired C whose intersection with S is empty. Before we go further, given a vertex set S and $\mathcal{G}$ (as defined above), we define $\mu(S,i) = \max_{A \subseteq S, |A|=i}\{w(B_r(A))\}$.

(T3) Using dynamic programming over graphs with bounded treewidth, compute $\mu(S,i)$ for $\mathcal{G}$ for $1 \leq i \leq k$ as well as a subset $A_i \subseteq S$ such that $w(B_r(A_i)) = \mu(S,i)$.

(T4) Now we make k recursive calls to reduce the size of k on the fact that if there exists a C then its intersection with S is between $1 \leq i \leq k$. Now we reduce the parameters t to $t - \mu(S,i)$ and k to $k - i$ and try to solve the problem recursively.

In the recursive steps, we follow the above steps and either we move forward to a larger $\mathcal{G}$ or we get a desired solution for the problem. More precisely, suppose we are at the i^{th} step of recursion then we do as follows:

(T5) Enlarge $\mathcal{G}$ by adding some new vertices to S. Let S_i be the set of new vertices added to S that is those set of vertices which are not in S and $w(B_r(v)) \geq t/k$ where t and k are the current parameters obtained after reductions done in previous recursive calls.

(T6) Either we bound the diameter and hence the treewidth of $\mathcal{G}$. Else, we select a set C of at most k vertices such that $w(B_r(C)) \geq t$ and C respects the guesses made on the number of vertices we need to select from S_j, $1 \leq j \leq i-1$. That is, the possible number of vertices in $C \cap S_j$.

This completes the framework in which we will be working. In the next Section we prove that WP-(k,r,t)-C Problem is FPT in graphs with bounded local treewidth by proving the necessary technical lemmas needed for this generic algorithm to work.

3.2 An Algorithm for WP-(k,r,t)-C in Graphs of Bounded Local Treewidth

We first give an upper bound on the treewidth of $\mathcal{G}$, the graphs we obtained in the recursive calls which is crucial for analysis of the algorithm.

LEMMA 1. *Let G be a graph on n vertices and m edges and H be an induced subgraph of G such that the diameter of each of the connected components of H is at most ℓ. Let C be a subset of $V(H)$ of size at most k and A be a subset of $V(G)$. Then there exists a function $g(k,r,\ell)$ such that if $diam(G[B_r(A) \cup H]) > g(k,r,\ell)$, then there is a subset $T \subseteq A$ such that* **(a)** *$|T| \geq k$;* **(b)** *for all $u,v \in T, d_G(u,v) \geq 2r+1$; and* **(c)** *for all $u \in T$ and for all $v \in C$, $d_G(u,v) \geq 2r+1$. In particular, one can take $g(k,r,\ell) = (6r+2)2k\ell$ and find the desired set T in $O(m+n)$ time.*

PROOF. Since C is a subset of size at most k, we have that it intersects at most k connected components of H. Let these connected components be $H_1,\ldots,H_r$, where $r \leq k$. We contract each of these connected components to a vertex and obtain a new graph G'. Let the contractions of $H_1,\ldots,H_r$ correspond to vertices $v_{H_1},\cdots,v_{H_r}$ in our new graph G' and this set of vertices be called X. For a vertex $v \in V(G)$, we define its image, $im(v)$, in G' as v_{H_i} if it is in H_i for $1 \leq i \leq r$ and v otherwise. For a subset $W \subseteq V$, its image $im(W)$ in $V(G')$, is defined as the set $\{im(v) \mid v \in W\}$.

For any subset $W \subseteq V(G)$, we claim that $diam(G'[im(W) \cup X]) \geq diam(G[W \cup H])/\ell$ (let us remind that we define the diameter of the graph as the maximum diameter of its connected components).

To prove the claim we observe that a path P' in $G'[im(W) \cup X]$ can be lifted to a path P in $G[W \cup H]$ by replacing every vertex in X on path P' by local paths in each connected component H_j of H. As the diameter of each H_j is bounded by ℓ, in this way, the length of a path can only be increased by at most a constant multiplicative factor ℓ. This gives us $diam(G[W \cup H]) \leq \ell \cdot\ diam(G'[im(W) \cup X])$, which completes the proof of the claim.

To finish the proof of the lemma we proceed as follows: We apply the above claim to the subset $W = B_r(A)$. Since $diam(G[B_r(A) \cup H]) > g(k,r,\ell) = (6r+2)2k\ell$, we have that

$$diam(G'[im(B_r(A)) \cup X]) \geq \frac{diam(G[B_r(A) \cup H])}{\ell} > \frac{g(k,r,\ell)}{\ell} = 2(6r+2)k.$$

Thus there is a connected component $\mathfrak{C}$ of $G'[im(B_r(A)) \cup X]$ of diameter more than $2(6r+2)k$. Let $im(v_1),\ldots,im(v_\kappa)$, $\kappa \leq k$, be the image of vertices of C in this component. Observe that $im(A) \cup \{im(v_1),\ldots,im(v_\kappa)\}$ form an r-center in $\mathfrak{C}$. Since the diameter of this component is at least $2(6r+2)k$, we can find a subset $Y \subseteq im(A) \cup \{im(v_1),\ldots,im(v_\kappa)\}$ of size at least $2k$ such that for any two vertices $u,v \in Y$, $d_{G'}(u,v) \geq 4r+1$. To see this, let us assume that $P = u_0u_1u_2\cdots u_q$, $q \geq 2(6r+2)k$, is a path which realizes this diameter. Let $V_i \subseteq V(\mathfrak{C})$ be the subset of vertices of distance exactly i from u_0. Since $im(A) \cup \{im(v_1),\ldots,im(v_\kappa)\}$ forms an r-center, its intersection with $\bigcup_{j=i}^{i+2r} V_i$, $1 \leq i \leq q-2r$, is non-empty. Now one

can form Y by selecting a vertex of $im(A) \cup \{im(v_1), \ldots, im(v_\kappa)\}$ from $\cup_{i=0}^{2r} V_i$ and then alternately not selecting any vertex from next $4r+1$ V_i's and then selecting a vertex of $im(A) \cup \{im(v_1), \ldots, im(v_\kappa)\}$ from one of the next $2r+1$ blocks of V_i's, and so on.

We put $Z = Y \cap \{im(v_1), \ldots, im(v_\kappa)\}$. Let us remark that, for each vertex v in $\{im(v_1), \ldots, im(v_\kappa)\} \setminus Z$ there is at most one vertex v in $Y \setminus Z$ such that $d_{G'}(u,v) \leq 2r$. Otherwise it will violate the condition that the distance between any two vertices from Y is at least $4r+1$ in G'. We construct the set T' by removing all vertices from $Y \setminus Z$ which are at distance at most $2r$ from $\{im(v_1), \ldots, im(v_\kappa)\} \setminus Z$. The subset $T' \subseteq im(A)$ satisfies the following conditions: **(a)** $|T'| \geq k$; **(b)** for all $u, v \in T'$, $d_{G'}(u,v) \geq 2r+1$; and **(c)** for all $u \in T'$ and for all $im(v_j)$, $1 \leq j \leq k$, $d_{G'}(u, im(v_j)) \geq 2r+1$.

Lifting the subset T' to G one gets a T (by taking inverse image of vertices in T') of the desired kind. ∎

Another essential part of our algorithm is dynamic programming on graphs with bounded treewidth which will be used in **(T6)**. To do so we use a variation of the Theorem 4.1 of [9].

THEOREM 2. $[\star]$* *Let G be a graph on n vertices, given with (a) a weight function $w : V \to \{0,1\}$, (b) a tree decomposition of width $\leq$ **b**, and (c) positive integers k, r and t. Furthermore let $S_1, \cdots, S_p$ be disjoint subsets of $V(G)$ with an associated positive integer a_i for $1 \leq i \leq p$ and $\sum_{i=1}^{p} a_i = a$. Then we can check the existence of a weighted partial-(k,r,t)-center such that it contains a_i elements from S_i, $1 \leq i \leq p$, in $O((2r+1)^{\frac{3\mathbf{b}}{2}} 2^{\frac{a}{2}} \cdot nt)$ time and, in case of a positive answer, construct a weighted partial-(k,r,t)-center of G in the same time.*

The rest of the section is devoted to the proof of the following theorem.

THEOREM 3. *Let $f : \mathbb{N} \to \mathbb{N}$ be a given function. Then WP-(k,r,t)-C problem can be solved in time $O(\tau(k,r) \cdot t \cdot (m+n))$ for graphs in $\mathcal{G}_f$, where τ is a function of k and r. In particular, WP-(k,r,t)-C problem is FPT for planar graphs, graphs of bounded genus and graphs of bounded maximum degree.*

Let us remark that for fixed k, r and t, our algorithm runs in linear time.

PROOF. The proof of the theorem is divided into three parts: Algorithm, correctness and the time complexity. We first describe the algorithm.

Algorithm: First we set up notations used in the algorithm. By $\mathcal{S}$ we mean a family of pairs (X,i) where X is a subset of $V(G)$, i is a positive integer, and for any two elements $(X_1,i_1), (X_2,i_2) \in \mathcal{S}$, $X_1 \cap X_2 = \emptyset$. Given a family $\mathcal{S}$, we define $\rho(\mathcal{S}) = \sum_{(X,i)\in\mathcal{S}} i$ and

$$\mu(w,\mathcal{S}) = \max\left\{ w(B_r(D)) \;\middle|\; D \subseteq V(G), |D| = \rho(\mathcal{S}), \forall (X,i) \in \mathcal{S}\ |D \cap X| = i \right\},$$

that is a subset $D \subseteq \bigcup_{(X,i)\in\mathcal{S}} X$ of size $\rho(\mathcal{S})$, under the additional constraint that for each element (X,i) of $\mathcal{S}$ we pick exactly i elements in X. A subset D realizing $\mu(w,\mathcal{S})$ will be called an *$\mathcal{S}$-center*. Our detailed algorithm is given in Figure 1.

Correctness: The correctness of the algorithm follows (almost directly) from its detailed descriptions in the earlier sections and hence we remark on the necessary points of the proof. Whenever we answer YES, we output a set C which has weight at least t that is $w(B_r(C)) \geq t$ and C is of size at most k and hence these steps do not require any justification.

*Results marked with $[\star]$ will appear in the long version of the paper.

Algorithm PCentre(G**,** r**,** k**,** t**,** w**,** $\mathcal{S}$**,** C**,** S**,** $\mu(w,\mathcal{S})$**)**
(The algorithms takes as an input (*a*) a graph $G = (V, E) \in \mathscr{G}_f$, (*b*) positive integers k, r and t, (*c*) a weight function $w : V \to \{0, 1\}$, (*d*) a family $\mathcal{S}$ of pairs (X, i), (*e*) an $\mathcal{S}$-center C, (*f*) a set S which is equal to $\cup_{(X,i)\in\mathcal{S}} X$ and (*g*) the value of $\mu(w,\mathcal{S})$. It returns either a set C such that $w(B_r(C)) \geq t$ or returns NO, if no such set exists. The algorithm is initialized with **PCentre**$(G, r, k, t, w, \emptyset, \emptyset, \emptyset, 0)$).
Step 0 : If $\mu(w,\mathcal{S}) \geq t$, then answer YES and return C.
Step 1: If $k = 0$ and $\mu(w,\mathcal{S}) < t$, then **return** NO and EXIT.
Step 2: First define A as follows: $A = \{v \mid v \in V\ , v \notin S, w(B_r(v)) \geq t/k\}$. If A is empty **return** NO and EXIT. Else let $S = S \cup A$ and define $\mathcal{G} = \bigcup_{v\in S} G[B_r(v)]$.
Step 3: Compute the diameter, $diam$, of $\mathcal{G}$.
Step 4: If $diam > ((12r + 4)(k + \rho(\mathcal{S})))^{|\mathcal{S}|+1}$ then apply Lemma 1 to find the subset $T \subseteq A$ of size k such that: **(a)** for all $u, v \in T$, $d_G(u, v) \geq 2r + 1$; and **(b)** for all $u \in T$ and for all $v \in C$, $d_G(u, v) \geq 2r + 1$ and **return** $C = C \cup T$ and EXIT.
Step 5: Else, the graph $\mathcal{G}$ has bounded local treewidth, compute a tree decomposition of width $f(diam)$ of $\mathcal{G}$.
Step 6: For every $1 \leq p \leq k$, using the dynamic programming of Theorem 2, compute a $\mathcal{S} \cup \{(A, p)\}$-center D_p of weight $\mu(w,\mathcal{S} \cup \{(A, p)\})$. If for some ***recursive calls***, $1 \leq p \leq k$, **PCentre**$(G, r, k - p, t - \mu(w,\mathcal{S} \cup \{(A, p)\}), w, \mathcal{S} \cup \{(A, p)\}, D_p, S, \mu(w,\mathcal{S} \cup \{(A, p)\}))$ returns a set C then answer YES and **return** C else answer NO and EXIT.

Figure 1: Algorithm for Weighted Partial Center Problem

Our observation is that if there exists a subset C such that $w(B_r(C)) \geq t$ and $|C| \leq k$, then C and $A = \{v \mid v \in V,\ w(B_r(v)) \geq t/k\}$ have non empty intersection. Hence we recursively solve the problem with an assumption that $|C \cap A| = p$, $p \in \{1, 2, \cdots, k\}$. In recursive steps we have a family $\mathcal{S}$ of pairs (X, i) such that we want to compute C with additional constraints that for all $(X, i) \in \mathcal{S}$, $|C \cap X| = i$. At this stage the only way we can have solution is when there exists a non-empty set A such that

$$C \cap A \neq \emptyset \text{ where } A = \left\{ v \;\middle|\; v \in V\ , v \notin (\cup_{(X,i)\in\mathcal{S}} X), w(B_r(v)) \geq \tfrac{t-\mu(w,\mathcal{S})}{k-\rho(\mathcal{S})} \right\} \neq \emptyset.$$

Now based on the diameter of the graph $\mathcal{G} = \bigcup_{v\in S} G[B_r(v)]$, where $S = A \cup_{(X,i)\in\mathcal{S}} X$, we either apply Lemma 1 or make further recursive calls.
(1.) When we apply Lemma 1, the diameter of the graph is more than $((12r + 4)k)^{|\mathcal{S}|+1}$, and hence we obtain a set $T \subseteq A$ such that T is of cardinality $k - \rho(\mathcal{S})$ and the distance between any two vertices in T and distance between vertices of T and C, C a $\mathcal{S}$-center, is at least $2r + 1$. In $|C \cup T| = |C| + |T| \leq \rho(\mathcal{S}) + k - \rho(\mathcal{S}) \leq k$, and

$$w(B_r(C \cup T)) = w(B_r(C)) + w(B_r(T)) \geq \mu(w,\mathcal{S}) + (k - \rho(\mathcal{S})) \times \tfrac{t-\mu(w,\mathcal{S})}{k-\rho(\mathcal{S})} \quad \geq t.$$

(2.) Else the diameter and hence the treewidth of the graph $\mathcal{G}$ is at most $f(((12r + 4)k)^{|\mathcal{S}|+1})$. Hence in this case there is a solution to the problem precisely when there exists p, $1 \leq p \leq k - \rho(\mathcal{S})$, for which recursive call to **PCentre** returns a solution in Step 6 of the algorithm. This completes the correctness of the algorithm.

Time Complexity: The running time depends on the number of recursive calls we make and the upper bound on the treewidth of the graphs $\mathcal{G}$ which we obtain during the execution of the algorithm. First we bound the number of recursive calls. An easy bound is k^k since the number of recursive calls made at any step is at most k and the depth of the recursion tree is also at most k. This bound can be improved as follows. Let $N(k)$ be the number of recursive

calls. Then $N(k)$ satisfies the recurrence $N(k) \leq \sum_{i=1}^{k} N(k-i)$, which solves to 2^k.

At every recursive call we perform a dynamic programming algorithm and since the size of the family $\mathcal{S}$ is at most $k-1$, the diameter of the graph does not exceed $((12r+4)k)^k$ at any step of the algorithm. Let $h(r,k) = 3 \cdot f(((12r+4)k)^k)/2$. Then the dynamic programming algorithm can be performed in $O((2r+1)^{h(r,k)}2^{\frac{k}{2}} \cdot (n+m)t)$ time in any recursive step of the algorithm. Hence the total time complexity of the algorithm is upper bounded by $O((2r+1)^{h(r,k)}2^{\frac{3k}{2}} \cdot (n+m)t)$. This completes the proof. ∎

3.3 Improved Algorithm for Planar Graphs

In the last section we gave an algorithm for WP- (k,r)-C problem in graphs of bounded local treewidth. The time complexity of the algorithm was dominated by the upper bound on the treewidth of the graph $\mathcal{G}$, which were considered in the recursive steps of the algorithm. If the input to the algorithm **Algorithm PCentre** is planar, then a direct application of Lemma 1 gives us that the treewidth of the graph $\mathcal{G}$, obtained in the recursive steps of the algorithm, is bounded by $O((rk)^{O(rk)})$. In this section we reduce this upper bound to $O(rk)$ using grid arguments. We also need to slightly modify **Algorithm PCenter** by replacing the diameter arguments with treewidth based arguments. We give the modified steps here:

Modified Step 3: Compute the treewidth of $\mathcal{G}$.

Modified Step 4: If $\mathbf{tw}(\mathcal{G}) > g(r,k)$ (to be specified later) find a subset $T \subseteq A$ of size k such that: **(a)** for all $u,v \in T$, $d_G(u,v) \geq 2r+1$; and **(b)** for all $u \in T$ and for all $v \in C$, $d_G(u,v) \geq 2r+1$ and **return** $C = C \cup T$ and EXIT.

Modified Step 5: Else, the graph $\mathcal{G}$ has bounded treewidth, compute a tree decomposition of width at most $g(r,k)$ of $\mathcal{G}$.

To give the combinatorial bound on the treewidth of the graph $\mathcal{G}$, we need the following relation between the size of grids and the treewidth of the planar graph.

LEMMA 4.[23] *Let $s \geq 1$ be an integer. The treewidth of every planar graph G with no $(s \times s)$-grid as a minor is upper bounded by $6s-4$.*

The notations used in the next lemma is the same as in **Algorithm PCentre**.

LEMMA 5. $[\star]$ *Let $G=(V,E)$ be a planar graph on n vertices and m edges. Let k,r and t be positive integers, and w be a weight function $w: V \to \{0,1\}$. Suppose that at some step in* **Algorithm PCentre** *we are given a family $\mathcal{S}$ of pairs (X,i), an $\mathcal{S}$-center C, a set $S = \cup_{(X,i)\in\mathcal{S}} X$ and the value of $\mu(\omega,\mathcal{S})$. Furthermore let $A = \{v \mid v \in V, v \notin S, w(B_r(v)) \geq t/k'\} \neq \emptyset$, $S^* = S \cup A$, where $k' = k - \sum_{(X,i)\in\mathcal{S}} i$. Finally, let $\mathcal{G} = \bigcup_{v\in S^*} G[B_r(v)]$. Then either there is a subset $T \subseteq A$ of size k' such that* **(a)** *for all $u,v \in T$, $d_G(u,v) \geq 2r+1$; and* **(b)** *for all $u \in T$ and for all $v \in C$, $d_G(u,v) \geq 2r+1$ or $\mathbf{tw}(\mathcal{G}) \leq O(rk)$.*

Let us set $g(r,k) = 6h(r,k)$. We can compute in $O(|\mathcal{G}|^4)$ time a tree decomposition of width ω of $\mathcal{G}$ such that $\mathbf{tw}(\mathcal{G}) \leq \omega \leq 1.5\mathbf{tw}(\mathcal{G})$ [25]. Moreover, given a graph $\mathcal{G}$, one can also construct a grid minor of size $(b/4) \times (b/4)$ where the largest grid minor possible in $\mathcal{G}$ is of order $b \times b$, in time $O(|\mathcal{G}|^2 \log |\mathcal{G}|)$ [6]. Hence if $\omega > g(r,k)$ then the $\mathbf{tw}(G) > 4h(r,k)$ and then by applying the polynomial time algorithm to compute grid minor, we can obtain a grid of size $\frac{4}{24}h(r,k)$. Let us finally observe that the proof of Lemma 5 is constructive, in a

sense that given the grid $\mathcal{H}$, we can construct the desired set T in polynomial time. Hence by setting $h(r,k) = O(rk)$ in the time complexity analysis of Theorem 3 , we obtain the following theorem.

THEOREM 6. *WP-(k,r,t)-C problem can be solved in time $O(2^{O(kr)} \cdot n^{O(1)})$ on planar graphs.*

4 H-minor free graphs

The arguments of the previous sections were based on a specific graph class property, namely, that a graph with small diameter has bounded treewidth. Thus the natural limit of our framework is the class of graphs of bounded local treewidth. We overcome this limit and extend the framework on the class of graphs excluding a fixed graph H as minor. To do so we need to use the structural theorem of Robertson and Seymour [24] and an algorithmic version of this theorem by Demaine et al. [13]. The algorithm is quite involved, it uses two levels of dynamic programming and two levels of implicit branching, and can be seen as a non-trivial extension of the algorithm of Demaine et al. [10] for classical covering problems to partial covering problems.

THEOREM 7. $[\star]$ *PDS is fixed parameter tractable for the class of H-minor free graphs and the algorithm takes time $O(\tau(k) \cdot t \cdot n^{C_H})$, where τ is the function of k only and C_H is the constant depending only on the size of H.*

5 Partial Vertex Cover

While the results of the previous section can be used to prove that PVC is FPT on H-minor free graphs, we do not need that heavy machinery for this specific problem. In this section we show how implicit branching itself does the job, even for more general classes of graphs. We present a simple modification to our framework developed in the Section 3.1 and use it to show that PVC problem is FPT in triangle free graphs. Given a graph $G = (V,E)$ and a subset $S \subseteq V$, by $\partial S \subseteq E$ we denote the set of all edges having at least one end-point in S. Our modification in the generic algorithm is in step **(T2)**.

(T2′) Bound the size of S as a function of the parameter in every recursive step.

We call a graph class $\mathscr{G}$ *hereditary* if for any $G \in \mathscr{G}$, all the induced subgraphs of G also belong to $\mathscr{G}$. Let $\xi : \mathbb{N} \to \mathbb{N}$ be an increasing function. We say that a hereditary graph class $\mathscr{G}$ has the *ξ-bounded independent set property*, or simply the property IS_ξ, if for any $G \in \mathscr{G}$ there exists an independent set $X \subseteq V(G)$ such that $|V(G)| \leq \xi(|X|)$ and X can be found in time polynomial in the input size. There are various graph classes which have the property of IS_ξ. Every bipartite graph has an independent set of size at least $n/2$ and hence we can choose $\xi_b : \mathbb{N} \to \mathbb{N}$ as $\xi_b(k) = 2k$. A triangle free graph has an independent set of size at least $\max\{\Delta, n/(\Delta+1)\}$ where Δ is the maximum degree of the graph which implies that a triangle free graphs has an independent set of size at least $\sqrt{n}/2$. In this case we can choose the function $\xi_t : \mathbb{N} \to \mathbb{N}$ by $\xi_t(k) = 4k^2$. Every H-minor free graphs, and in particular for planar graphs and graphs of bounded genus have chromatic number at most $g(H)$ for some function depending on H alone. In this case G has an independent set of size at least $n/g(|H|)$ and we can take $\xi_H(n) = g(H)n$. For planar graphs $g(H)$ is 4.

We can show that if a graph class $\mathcal{G}$ has the property IS_{ξ}, then in the case of PVC for every $G \in \mathcal{G}$ either we can upper bound the size of S used in the implicit branching step by $\xi(k)$ or we can find a subset V' of size at most k such that $|\partial V'| \geq t$. The main theorem of this section is as follows.

THEOREM 8. $[\star]$ *Let $\mathcal{G}$ be a hereditary graph class with the property of* IS_{ξ} *for some integer function ξ. Then* PVC *can be solved in $O(\tau(k) \cdot n^{O(1)})$ time in $\mathcal{G}$ where $\tau(k) = \xi(k)^k$.*

6 Conclusion

In this paper we obtained a framework to give FPT algorithms for various partial covering problems in graphs with locally bounded treewidth and graphs excluding a fixed graph H as a minor. The main idea behind our approach was the concept of implicit branching which is of independent interest. We believe that it will be useful for other problems as well. We conclude with some open questions. For planar graphs (and even more generally, for H-minor free graphs), many non-partial versions of parameterized problems can be solved in subexponential time [12, 14]. We show that for planar graphs PARTIAL DOMINATING SET can be solved in time $2^{O(k)} \cdot n^{O(1)}$. Is this result tight, in a sense that up to some assumption in the complexity theory, there is no time $2^{o(k)} \cdot n^{O(1)}$ algorithm solving this problem on planar graphs?

Many non-partial parameterized problems on planar graphs can be solved by reducing to a kernel of linear size [2]. This does not seem to be the case for their partial counterparts and an interesting question here is, whether PARTIAL DOMINATING SET or PARTIAL VERTEX COVER can be reduced to polynomial sized kernels on planar graphs.

References

[1] J. Alber, H. Fan, M. R. Fellows, H. Fernau, R. Niedermeier, F. A. Rosamond, and U. Stege. *A refined search tree technique for Dominating Set on planar graphs.* Journal of Computer and System Sciences, 71(4), 385-405 (2005).

[2] J. Alber, M. R. Fellows and R. Niedermeier. *Polynomial-time Data Reduction for Dominating Set*. Journal of ACM 51(3): 363-384 (2004).

[3] N. Alon and S. Gutner. *Linear Time Algorithms for Finding a Dominating Set of Fixed Size in Degenerated Graphs.* In the Proc. of COCOON, LNCS 4598,394-405 (2007).

[4] S. Arora and G. Karakostas. *A $2 + \epsilon$ Approximation Algorithm for the k-MST Problem*. In Proc. of Symposium on Discrete Algorithms (SODA), 754-759 (2000).

[5] Reuven Bar-Yehuda. *Using Homogenous Weights for Approximating the Partial Cover Problem*. In Proc. of Symposium on Discrete Algorithms (SODA), 71-75 (1999).

[6] H. L. Bodlaender, A. Grigoriev, A. M. C. A. Koster. *Treewidth Lower Bounds with Brambles*. In Proc. of European Symposium on Algorithms (ESA), LNCS 3669, 391-402, (2005).

[7] M. Charikar, S. Khuller, D. Mount and G. Narasimhan. *Algorithms for Facility Location Problems with Outliers*. In Proc. of Symposium on Discrete Algorithms (SODA), 642-651 (2001).

[8] J. Chen, I. A. Kanj and G. Xia. *Improved Parameterized Upper Bounds for Vertex Cover*. In Proc. of Mathematical Foundations of Computer Science (MFCS), LNCS 4162, 238-249 (2006).

[9] E. D. Demaine, F. V. Fomin, M.T. Hajiaghayi and D. M. Thilikos. *Fixed Parameter Algorithms for (k, r)-Center in Planar Graphs and Map Graphs*. ACM transactions on Algorithms, 1(1): 33-47 (2005).

[10] E. D. Demaine, F. V. Fomin, M.T. Hajiaghayi and D. M. Thilikos. *Subexponential Parameterized Algorithms on Bounded-genus Graphs and H-minor-free Graphs*. Journal of ACM 52(6): 866-893 (2005).

[11] E. D. Demaine and M.T. Hajiaghayi. *Equivalence of Local Treewidth and Linear Local Treewidth and its Algorithmic Applications*. In Proc. of Symposium on Discrete Algorithms (SODA), 840-849 (2004).

[12] E. DEMAINE AND M. HAJIAGHAYI, *The bidimensionality theory and its algorithmic applications*, The Computer Journal, to appear.

[13] E. D. Demaine, M. T. Hajiaghayi and K. C. Kawarabayashi. *Algorithmic Graph Minor Theory: Decomposition, Approximation and Coloring*. In Proc. of Foundations of Computer Science (FOCS), 637-646 (2005).

[14] F. Dorn, F. V. Fomin, and D. M. Thilikos. *Subexponential parameterized algorithms*, Computer Science Reviews, to appear.

[15] R.G. Downey and M.R. Fellows. *Parameterized Complexity*, Springer, (1999).

[16] D. Eppstein. *Diameter and Treewidth in Minor Closed Graph Families*, Algorithmica, 27 (3-4): 275-291 (2000).

[17] J. Flum and M. Grohe. *Parameterized Complexity Theory*, Springer, (2006).

[18] R. Gandhi, S. Khuller and A. Srinivasan. *Approximation Algorithms for Partial Covering Problems*. Journal of Algorithms 53(1): 55-84 (2004).

[19] P. Golovach and Y. Villanger. *Parameterized Complexity for Domination Problems on Degenerate Graphs*. To appear in the proceedings of WG, LNCS, (2008).

[20] M. Grohe. *Local Treewidth, Excluded Minors and Approximation Algorithms*. Combinatorica, 23(4): 613-632 (2003).

[21] R. Niedermeier. *Invitation to Fixed-Parameter Algorithms*, Oxford University Press, (2006).

[22] N. Robertson and P. Seymour. *Graph minors V, Excluding a Planar Graph*. Journal of Combinatorial Theory, Series B, 41(2): 92-114, (1986).

[23] N. Robertson, P. Seymour and R. Thomas. *Quickly Excluding a Planar Graph*. Journal of Combinatorial Theory, Series B, 62(2): 323-348 (1994)

[24] N. Robertson and P. Seymour. *Graph minors XVI, Excluding a Non-planar Graph*. Journal of Combinatorial Theory, Series B, 81(1): 43-76 (2003).

[25] P. D. Seymour and R. Thomas. *Call routing and the ratcatcher*. Combinatorica, 14(2):217241, (1994).

Foundations of Software Technology and Theoretical Computer Science (Bangalore) 2008.
Editors: R. Hariharan, M. Mukund, V. Vinay; pp 13-24

Sound Lemma Generation for Proving Inductive Validity of Equations

Takahito Aoto
RIEC, Tohoku University
aoto@nue.riec.tohoku.ac.jp

ABSTRACT. In many automated methods for proving inductive theorems, finding a suitable generalization of a conjecture is a key for the success of proof attempts. On the other hand, an obtained generalized conjecture may not be a theorem, and in this case hopeless proof attempts for the incorrect conjecture are made, which is against the success and efficiency of theorem proving. Urso and Kounalis (2004) proposed a generalization method for proving inductive validity of equations, called sound generalization, that avoids such an over-generalization. Their method guarantees that if the original conjecture is an inductive theorem then so is the obtained generalization. In this paper, we revise and extend their method. We restore a condition on one of the characteristic argument positions imposed in their previous paper and show that otherwise there exists a counterexample to their main theorem. We also relax a condition imposed in their framework and add some flexibilities to some of other characteristic argument positions so as to enlarge the scope of the technique.

1 Introduction

Reasoning on data structures or recursively defined domains is very common in formal treatments of programs such as program verification and program transformation. Such a reasoning often needs highly use of induction, that is, the properties of interest are not only (general) theorems which hold in all models of the theory but *inductive theorems* which hold only in a particular model, the initial model of the theory.

Although automated reasoning of inductive theorems has been investigated in many years, comparing to the high degree of automation on automated proving of (general) theorems, automated proving of inductive theorems is still considered as a very challenging problem [8]. Many approaches to automated proving of inductive theorems are known: explicit induction with sophisticated heuristics and/or decision procedures [4, 5, 6, 11, 13, 17], *implicit* induction methods such as inductionless induction/coverset induction/rewriting induction [3, 7, 9, 14, 16, 19].

In all these approaches, it is commonly understood that an introduction of suitable lemmas is an important key for the success of proof attempts. Thus techniques for finding suitable lemmas in the course of proof attempts have been investigated [12, 15, 18, 21, 22]. Among them, one of the most basic methods is *generalization*—replacing some of equivalent subterms of the conjecture by a fresh variable. Proving generalized conjecture is often easier than the original conjecture because generalization often suppress the complexity at the induction step and sometimes makes another induction scheme possible. On the other hand, the generalized conjecture may not be a theorem any more—this phenomenon is often refereed to as *over-generalization*. Because hopeless proof attempts for the incorrect conjecture is against the success and efficiency of theorem proving, any over-generalization is always better to be avoided.

FSTTCS 2008
IARCS Annual Conference on
Foundations of Software Technology and Theoretical Computer Science
http://drops.dagstuhl.de/opus/volltexte/2008/1737

Urso and Kounalis [21] proposed a generalization method called *sound generalization*, which avoid such an over-generalization in automated inductive theorem proving of equations. Their method is sound in the sense it guarantees that if the original conjecture is an inductive theorem then so is the obtained generalization. Thus the original conjecture can be safely replaced by the obtained generalization if the criteria is satisfied. However, the paper [21] contains an incorrect proof and, in fact, there exists a counterexample to their main theorem.

Example 1 (counterexample) *Let* $\mathcal{S} = \{\mathrm{Nat}\}$, $\mathcal{F} = \{\ \mathsf{plus}^{\mathrm{Nat}\times\mathrm{Nat}\to\mathrm{Nat}},\ \mathsf{f}^{\mathrm{Nat}\to\mathrm{Nat}},\ \mathsf{s}^{\mathrm{Nat}\to\mathrm{Nat}},\ 0^{\mathrm{Nat}}\ \}$ *and*

$$\mathcal{R} = \left\{ \begin{array}{lcl} \mathsf{plus}(0,y) & \to & y \\ \mathsf{plus}(\mathsf{s}(x),y) & \to & \mathsf{s}(\mathsf{plus}(x,y)) \\ \mathsf{f}(0) & \to & \mathsf{s}(\mathsf{s}(0)) \\ \mathsf{f}(\mathsf{s}(x)) & \to & \mathsf{s}(\mathsf{s}(x)) \end{array} \right\}$$

Then $\mathcal{R}$ *is a monomorphic TRS and the argument* 1 *is a downward position of* f *[21]. In a mathematical notation,* f *is a function* f *like this:*

$$f(x) = \left\{ \begin{array}{ll} 2 & \text{if } x = 0 \\ x+1 & \text{if } x > 0 \end{array} \right.$$

Let $s \equiv \mathsf{s}(\mathsf{f}(\mathsf{plus}(x,\mathsf{s}(0))))$, $t \equiv \mathsf{f}(\mathsf{s}(\mathsf{plus}(x,\mathsf{s}(0))))$ *and consider a conjecture* $s \doteq t$*, i.e.*

$$\mathsf{s}(\mathsf{f}(\mathsf{plus}(x,\mathsf{s}(0)))) \doteq \mathsf{f}(\mathsf{s}(\mathsf{plus}(x,\mathsf{s}(0))))$$

Then clearly this is an inductive theorem (on natural numbers), since we have $f(x+1)+1 = (x+2)+1 = f(x+2)$*. Now let us try a generalization of this conjecture based on the original sound generalization [21]. We have* $1.1.1 = \mathrm{BP}(s)$ *and* $1.1.1 = \mathrm{BP}(t)$ *and thus* 1.1 *is a bottom path of* s *and* t*. Since* $\mathsf{bot}(s,1.1) \equiv \mathsf{plus}(x,\mathsf{s}(0)) \equiv \mathsf{bot}(t,1.1)$ *and* $s/1.1. \equiv \mathsf{plus}(x,\mathsf{s}(0)) \equiv t/1.1$*, the generalization at* 1.1 *in* s *and at* 1.1 *in* t *is possible. Hence we obtain a generalized conjecture*

$$\mathsf{s}(\mathsf{f}(y)) \doteq \mathsf{f}(\mathsf{s}(y))$$

However, this is not an inductive theorem since $\mathsf{s}(\mathsf{f}(0)) \to_{\mathcal{R}} \mathsf{s}(\mathsf{s}(\mathsf{s}(0)))$ *and* $\mathsf{f}(\mathsf{s}(0)) \to_{\mathcal{R}} \mathsf{s}(\mathsf{s}(0))$*. Therefore, this generalization is not sound contrary to the* ***Theorem 37*** *of [21].*

The purpose of this paper is to correct and extend the sound generalization proposed in [21]. In the sound generalization, generalizable subterms are computed based on five types of argument positions of functions—namely, reflective argument, downward position, upward position, down-contextual position, and up-contextual position when the term rewriting system is monomorphic. We clarify that the notion of downward position should be weakened as in their previous paper [20] that proposes induction on term partition, otherwise there is a counterexample (as presented above) and that the notion of down-contextual and up-contextual position can be enlarged so that more flexible rewrite rules are allowed for functions to have such positions. We relax the definition of monomorphic signature and localize the monomorphic and left-linearity conditions so as to enlarge the scope of the sound generalization.

The rest of the paper is organized as follows. After fixing basic notation (Section 2), we introduce a relaxed definition of monomorphic signature and revised definitions of argument positions and prove the characterization lemmas for these argument positions (Section 3). The term partition and sound generalization techniques are presented in Section 4. Section 5 concludes.

2 Preliminaries

We assume familiarity with basic notations on (many-sorted) term rewriting ([2, 10]).

Let $\mathcal{S}$ be a set of sorts and $\mathcal{F}$ be a set of $\mathcal{S}$-sorted function symbols. We assume there is a function from $\mathcal{F}$ to $\mathcal{S}^* \times \mathcal{S}$, denoted by sort. For $f \in \mathcal{F}$, let $\mathrm{sort}(f) = \langle \tau_1 \cdots \tau_n, \tau_0 \rangle$. Then $\langle \tau_1 \cdots \tau_n, \tau_0 \rangle$ is called the sort of f and denoted by $\tau_1 \times \cdots \times \tau_n \to \tau_0$. If $n = 0$, we write $\mathrm{sort}(f) = \tau$ and f is called a *constant* of sort τ.

Let V^τ be the set of variables of sort $\tau \in \mathcal{S}$. We assume there is a countably infinite set V^τ of variables for each $\tau \in \mathcal{S}$. We denote by V the set $\bigcup_{\tau \in \mathcal{S}} V^\tau$. The set $\mathrm{T}(\mathcal{F}, V)^\tau$ of terms of sort $\tau \in \mathcal{S}$ over $\mathcal{F}, V$ is defined inductively as: (1) $V^\tau \subseteq \mathrm{T}(\mathcal{F}, V)^\tau$; (2) if $f \in \mathcal{F}$, $\mathrm{sort}(f) = \tau_1 \times \cdots \times \tau_n \to \tau_0$ $(n \geq 0)$, $t_i \in \mathrm{T}(\mathcal{F}, V)^{\tau_i}$ for $1 \leq i \leq n$, then $f(t_1, \ldots, t_n) \in \mathrm{T}(\mathcal{F}, V)^{\tau_0}$. We denote by $\mathrm{T}(\mathcal{F}, V)$ the set $\bigcup_{\tau \in \mathcal{S}} \mathrm{T}(\mathcal{F}, V)^\tau$. We write t^τ if $t \in \mathrm{T}(\mathcal{F}, V)^\tau$. The set of variables contained in a term t is denoted by $\mathrm{V}(t)$. We use $\equiv$ to denote the syntactical equality.

A *position* is a (possibly empty) sequence of positive integers. The empty sequence is denoted by ϵ and the concatenation of positions p and q is by $p.q$. The set $\mathrm{Pos}(t)$ of positions (or *occurrence*) in a term t and the *subterm* t/p of t at the position p are recursively defined as follows: for $t \in V$, $\mathrm{Pos}(t) = \{\epsilon\}$ and $t/\epsilon = t$; for $t \equiv f(t_1, \ldots, t_n)$, $\mathrm{Pos}(t) = \{\epsilon\} \cup \bigcup_{1 \leq i \leq n} \{i.p \mid p \in \mathrm{Pos}(t_i)\}$, $t/\epsilon = t$, and $t/i.p = t_i/p$. If $p \in \mathrm{Pos}(t)$ and $\mathrm{sort}(t/p) = \mathrm{sort}(s)$, we write $t[s]_p$ the term obtained from t by replacing the subterm with s at the position p. A variable $x \in \mathrm{V}(t)$ is said to have a *linear variable occurrence* in t if there exists a unique $p \in \mathrm{Pos}(t)$ such that $x \equiv t/p$. The prefix ordering $\leq$ on positions are defined as $p \leq q$ iff $q = p.r$ for some position r. We write $p \mid q$ if neither $p \leq q$ nor $q \leq p$ hold. A set of position P is said to be *prefix-closed* if $p \in P$ and $q \leq p$ imply $q \in P$. The function symbol that occurs in t at a position $p \in \mathrm{Pos}(t)$ is denoted by $t(p)$. In particular, the *root symbol* of a term t is $t(\epsilon)$.

Suppose $\Box^\tau$ is a constant of sort τ and $\{\Box^\tau \mid \tau \in \mathcal{S}\} \cap \mathcal{F} = \emptyset$. A *context* is an element in $\mathrm{T}(\mathcal{F} \cup \{\Box^\tau \mid \tau \in \mathcal{S}\}, V)$. The special constants $\Box^\tau$ are called *holes*. If the holes occurring in a context C are $\Box^{\tau_1}, \ldots, \Box^{\tau_n}$ from left to right and $t_1, \ldots, t_n$ are terms of sorts $\tau_1, \ldots, \tau_n$, respectively, then we denote by $C[t_1, \ldots, t_n]$ the term obtained by replacing the holes $\Box^{\tau_1}, \ldots, \Box^{\tau_n}$ with the terms $t_1, \ldots, t_n$. The superscript of holes is often omitted if no confusion arises. For a position p, we write $C[u]_p$ if $C/p \equiv \Box$.

A map σ from V to $\mathrm{T}(\mathcal{F}, V)$ is called a *substitution* if (1) σ preserves sort, i.e. $\mathrm{sort}(x) = \mathrm{sort}(\sigma(x))$ and (2) the domain of σ is finite, where the domain of σ is given by $\mathrm{dom}(\sigma) = \{x \in V \mid \sigma(x) \not\equiv x\}$. A substitution σ such that $\mathrm{dom}(\sigma) = \{x_1, \ldots, x_n\}$ and $\sigma(x_i) \equiv t_i$ $(1 \leq i \leq n)$ is also written as $\{x_1 := t_1, \ldots, x_n := t_n\}$. We identify the substitution σ and its homomorphic extension. A term $\sigma(t)$ is called an *instance* of the term t; $\sigma(t)$ is also written as $t\sigma$.

A pair $\langle l, r \rangle$ of terms l, r satisfying conditions (1) $l(\epsilon) \in \mathcal{F}$ and (2) $\mathrm{V}(r) \subseteq \mathrm{V}(l)$ (3) $\mathrm{sort}(l) =$

$\mathrm{sort}(r)$ is said to be a *rewrite rule*. A rewrite rule $\langle l, r\rangle$ is denoted by $l \to r$. A tuple $\langle \mathcal{S}, \mathcal{F}, \mathcal{R}\rangle$ is a *term rewriting system (TRS)*. If no confusion arises, $\langle \mathcal{S}, \mathcal{F}, \mathcal{R}\rangle$ is abbreviated as $\mathcal{R}$. If there exist a position p, a substitution σ, and a rewrite rule $l \to r \in \mathcal{R}$ such that $s/p \equiv l\sigma$ and $t \equiv s[r\sigma]_p$, we write $s \to_{\mathcal{R}} t$. We call $s \to_{\mathcal{R}} t$ a *rewrite step*, p a *redex occurrence*, and $\to_{\mathcal{R}}$ the rewrite relation of the TRS $\mathcal{R}$. The reflexive transitive closure and equivalence closure of $\to_{\mathcal{R}}$ are denoted by $\overset{*}{\to}_{\mathcal{R}}$ and $\overset{*}{\leftrightarrow}_{\mathcal{R}}$, respectively. A TRS $\mathcal{R}$ is *terminating* if $\to_{\mathcal{R}}$ is noetherian i.e. there is no infinite sequence $t_0 \to_{\mathcal{R}} t_1 \to_{\mathcal{R}} \cdots$; is *confluent* if $\overset{*}{\leftarrow}_{\mathcal{R}} \circ \overset{*}{\to}_{\mathcal{R}} \subseteq \overset{*}{\to}_{\mathcal{R}} \circ \overset{*}{\leftarrow}_{\mathcal{R}}$. A term is said to be *normal* if there exists no s such that $t \to_{\mathcal{R}} s$. Any normal term s such that $t \overset{*}{\to}_{\mathcal{R}} s$ is called a *normal form* of t. One can easily show that if a TRS $\mathcal{R}$ is terminating and confluent, any term s has a unique normal form; the normal form of s is denoted by $s{\downarrow_{\mathcal{R}}}$, or simply by $s{\downarrow}$ if no confusion arises.

The set of *defined function symbols* is given by $\mathcal{D}_{\mathcal{R}} = \{l(\epsilon) \mid l \to r \in \mathcal{R}\}$ and the set of *constructor symbols* by $\mathcal{C}_{\mathcal{R}} = \mathcal{F} \setminus \mathcal{D}_{\mathcal{R}}$. The set of defined symbols appearing in a term t is denoted by $\mathcal{D}_{\mathcal{R}}(t)$. If $\mathcal{R}$ is obvious from its context, we omit the subscript $\mathcal{R}$ from $\mathcal{D}_{\mathcal{R}}, \mathcal{C}_{\mathcal{R}}$. Terms in $\mathrm{T}(\mathcal{C}, V)$ are said to be *constructor terms*.

An *equation* $l \doteq r$ is a pair $\langle l, r\rangle$ of terms of the same sort. When we write $l \doteq r$, however, we do not distinguish $\langle l, r\rangle$ and $\langle r, l\rangle$. A term t is said to be *ground* if $\mathrm{V}(t) = \emptyset$. The set of ground terms is denoted by $\mathrm{T}(\mathcal{F})$. If $t\sigma \in \mathrm{T}(\mathcal{F})$, $t\sigma$ is called a *ground instance* of t. The ground instance of an equation is defined analogously. A *ground substitution* is a substitution σ_g such that $\sigma_g(x) \in \mathrm{T}(\mathcal{F})$ for any $x \in \mathrm{dom}(\sigma_g)$. Without loss of generality, we assume that $t\sigma_g$ is ground (i.e. $\mathrm{V}(t) \subseteq \mathrm{dom}(\sigma_g)$) when we speak of an instance $t\sigma_g$ of t by a ground substitution σ_g; and so for ground instances of equations. An *inductive theorem* of a TRS $\mathcal{R}$ is an equation that is valid on $\mathrm{T}(\mathcal{F})$, i.e. $s \doteq t$ is an inductive theorem if $s\sigma_g \overset{*}{\leftrightarrow}_{\mathcal{R}} t\sigma_g$ holds for any ground instance $s\sigma_g \doteq t\sigma_g$. We write $\mathcal{R} \vdash_{ind} s \doteq t$ if $s \doteq t$ is an inductive theorem. A TRS $\mathcal{R}$ is said to be *sufficiently complete* if for any ground term $t_g \in \mathrm{T}(\mathcal{F})$, there exists a constructor ground term $s_g \in \mathrm{T}(\mathcal{C})$ such that $t_g \overset{*}{\leftrightarrow}_{\mathcal{R}} s_g$. One can easily show that if the TRS is sufficiently complete, terminating, and confluent then the normal form of any ground term is a constructor term.

Throughout this paper, we only deal with the TRSs that are sufficiently complete, terminating, and confluent.

3 Characterization of Monomorphic Equations

In this section, we introduce a relaxed definition of monomorphic signature and revised definitions of argument positions—reflective argument position, downward and upward argument positions, and contextual positions—and present lemmas that characterize these positions.

The notion of monomorphic signature is introduced by Urso and Kounalis [20, 21]. We here generalize the notion to monomorphic sorts, terms, etc.

DEFINITION 1.*[monomorphic sort]*

1. *A sort τ is said to be monomorphic if (i) there is only one constructor constant of the sort τ (denoted by $\perp^{\tau}$), (ii) for each non-constant constructor $g \in \mathcal{C}$ of sort $\tau_1 \times \cdots \times \tau_n \to \tau$, there exists a unique $1 \leq i \leq n$ such that $\tau_i = \tau$; such i is called the reflective*

argument position of g and denoted by $RA(g)$.

2. *A variable, term, equation, and rule are said to be monomorphic if its sort is monomorphic.*

Intuitively, a sort is monomorphic if each normal term of that sort has a list structure. For example, NatList (with $\mathsf{nil} : \mathrm{NatList}$ and $\mathsf{cons} : \mathrm{Nat} \times \mathrm{NatList} \to \mathrm{NatList}$), Nat (with $0 : \mathrm{Nat}$ and $\mathsf{s} : \mathrm{Nat} \to \mathrm{Nat}$) are monomorphic while Tree, Bool are not.

We here removed one of the conditions contained in the original definition of monomorphicness. Let $\succ_{\mathcal{S}}$ be a relation on $\mathcal{S}$ given by $\tau \succ_{\mathcal{S}} \rho$ iff there exists a ground constructor term $s_g[u_g^{\rho}]^{\tau}$ with $\tau \neq \rho$. In the original definition, the monomorphic signature is the one with only monomorphic sorts such that there are no ρ, δ such that $\rho \succ_{\mathcal{S}} \delta \succ_{\mathcal{S}} \rho$. The acyclicity of $\succ_{\mathcal{S}}$, however, turns out to be unnecessary in the subsequent development for sound generalization. Moreover, the monomorphic condition can be localized so that the signature may contain non-monomorphic sorts as well. This relaxation is useful, for example, to deal with BoolList.

We introduce a notion of reflective positions in a monomorphic term as a successive sequence of reflective argument positions from its root. Then, based on this, we define a join operator. This is in contrast to the original definition in [20, 21] where the join operator is defined as the replacement with $\bot$. The elimination of the extra restriction of monomorphic signature is achieved due to our new definition.

DEFINITION 2.*[reflective position] The set $\mathrm{RPos}(t)$ of reflective positions in t is defined as follows: (i) $\epsilon \in \mathrm{RPos}(t)$ (ii) if $t \equiv g(t_1, \ldots, t_n)$ with $g \in \mathcal{C}$, $i = RA(g)$, and $p \in \mathrm{RPos}(t_i)$ then $i.p \in \mathrm{RPos}(t_i)$.*

For example, we have $\mathrm{RPos}(\mathsf{s}(\mathsf{s}(0))) = \{\epsilon, 1, 1.1\}$. Since $\mathrm{RPos}(t)$ is total w.r.t. the prefix ordering $\leq$, there exists a position p that is greatest (w.r.t. $\leq$) in $\mathrm{RPos}(t)$.

DEFINITION 3.*[greatest reflective position] Let t be a monomorphic term. The greatest element w.r.t. the prefix ordering in $\mathrm{RPos}(t)$ is called the greatest reflective position (grp) of t.*

DEFINITION 4.*[join operator] For each monomorphic sort τ, a join operator $\otimes^{\tau}$ on the set $\mathrm{T}(\mathcal{C})$ is defined as follows: for ground constructor terms s_g and t_g of sort τ, $s_g \otimes^{\tau} t_g = s_g[t_g]_p$ where p is the grp of s_g. We omit the superscript τ if no confusion arises.*

The following properties of join operator is easily verified.

LEMMA 5.*[properties of join operator] Let $s_g, t_g, u_g \in \mathrm{T}(\mathcal{C})$ be monomorphic terms.*

1. *If $s_g \otimes t_g \equiv s_g \otimes u_g$ then $t_g \equiv u_g$. If $s_g \otimes t_g \equiv u_g \otimes t_g$ then $s_g \equiv u_g$.*
2. *$(s_g \otimes t_g) \otimes u_g \equiv s_g \otimes (t_g \otimes u_g)$.*
3. *$p \in \mathrm{RPos}(u_g)$ implies $u_g[s_g \otimes t_g]_p \equiv u_g[s_g]_p \otimes t_g$.*
4. *$\bot \otimes t_g \equiv t_g$ and $s_g \otimes \bot \equiv s_g$.*

LEMMA 6.*[decomposition at a reflective position] Suppose $t_g, u_g \in \mathrm{T}(\mathcal{F})$ are monomorphic and $p \in \mathrm{RPos}(t_g)$. Then $t_g[u_g]_p{\downarrow} \equiv t_g[\bot]_p{\downarrow} \otimes u_g{\downarrow}$.*

PROOF. By induction on p. (B.S.) Suppose $p = \epsilon$. Then $t_g[u_g]_p{\downarrow} \equiv u_g{\downarrow} \equiv \bot \otimes u_g{\downarrow} \equiv \bot{\downarrow} \otimes u_g{\downarrow} \equiv t_g[\bot]_p{\downarrow} \otimes u_g{\downarrow}$. (I.S.) Let $p = i.q$ with $t_g \equiv g(t_1, \ldots, t_n)$, $g \in \mathcal{C}$, $i = RA(g)$, and

$q \in \mathrm{RPos}(t_i)$. Then

$$\begin{array}{rcll} t_g[u_g]_p\downarrow & \equiv & g(t_1,\ldots,t_i[u_g]_q,\ldots,t_n)\downarrow & \text{by definition} \\ & \equiv & g(t_1\downarrow,\ldots,t_i[u_g]_q\downarrow,\ldots,t_n\downarrow) & \text{by } g \in \mathcal{C} \\ & \equiv & g(t_1\downarrow,\ldots,t_i[\bot]_q\downarrow \otimes u_g\downarrow,\ldots,t_n\downarrow) & \text{by the induction hypothesis} \\ & \equiv & g(t_1\downarrow,\ldots,t_i[\bot]_q\downarrow,\ldots,t_n\downarrow) \otimes u_g\downarrow & \text{by } i = RA(g) \text{ and } \textbf{Lemma 5} \\ & \equiv & g(t_1,\ldots,t_i[\bot]_q,\ldots,t_n)\downarrow \otimes u_g\downarrow & \text{by } g \in \mathcal{C} \\ & \equiv & t_g[\bot]_p\downarrow \otimes u_g\downarrow. & \end{array}$$

∎

In [20, 21], the notion of downward position is defined recursively; however, the mutual recursion of the definition is not terminating and thus the downward positions may not be uniquely defined for a TRS. To make this fact explicit, we introduce a notion of downward argument map and that of compatibility of the map with a TRS.

DEFINITION 7. *[downward argument map/downward position]*

1. *A downward argument map DP is a partial map from $\mathcal{D}$ to $\mathbb{N}$ such that for any $f \in \mathrm{dom}(DP)$, if $i = DP(f)$ then (1) $1 \le i \le \mathrm{arity}(f)$, and (2) if $f : \tau_1 \times \cdots \times t_n \to \tau_0$ then $\tau_i = \tau_0$ and τ_0 is monomorphic.*
2. *Let p be a position in a term t. The set $\mathrm{DPos}(t)$ of downward positions in t is defined as follows: (i) $\epsilon \in \mathrm{DPos}(t)$ (ii) if $t \equiv g(t_1,\ldots,t_n)$ with $g \in \mathcal{C}$, $i = RA(g)$, and $p \in \mathrm{DPos}(t_i)$ then $i.p \in \mathrm{DPos}(t_i)$. (iii) $t \equiv f(t_1,\ldots,t_n)$ with $f \in \mathcal{D}$, $i = DP(f)$, and $p \in \mathrm{DPos}(t_i)$ then $i.p \in \mathrm{DPos}(t_i)$.*

DEFINITION 8. *[compatible downward argument map] A downward argument map DP is compatible with a set $\mathcal{R}$ of rewrite rules if for any $f \in \mathrm{dom}(DP)$ with $i = DP(f)$ and for any $f(l_1,\ldots,l_n) \to r \in \mathcal{R}$, l_i is a linear variable occurrence of $f(l_1,\ldots,l_n)$ and there exists a position $p \in \mathrm{DPos}(r)$ such that $l_i \equiv r/p$ and r/p is a linear variable occurrence in r.*

Contrary to the definition in [21] in which l_i ($\equiv r/p$) is allowed to be an arbitrary term when $p \neq \epsilon$, we impose a restriction that l_i must be a variable; this condition is imposed in their previous paper [20] that proposes induction on term partition.

Example 2 *Let $\mathcal{S} = \{\mathrm{Nat}\}$, $\mathcal{F} = \{\ \mathsf{plus}^{\mathrm{Nat}\times\mathrm{Nat}\to\mathrm{Nat}}, \mathsf{s}^{\mathrm{Nat}\to\mathrm{Nat}}, \mathsf{0}^{\mathrm{Nat}}\ \}$ and*

$$\mathcal{R} = \left\{ \begin{array}{lcl} \mathsf{plus}(\mathsf{0},y) & \to & y \\ \mathsf{plus}(\mathsf{s}(x),y) & \to & \mathsf{s}(\mathsf{plus}(x,y)) \end{array} \right\}.$$

Then $\bot^{\mathrm{Nat}} \equiv \mathsf{0}$ and $\mathrm{RA}(\mathsf{s}) = 1$. A map DP with $\mathrm{DP}(\mathsf{plus}) = 2$ is a downward argument map compatible with $\mathcal{R}$.

Example 3 *Let $\mathcal{S} = \{\mathrm{Nat}\}$, $\mathcal{F} = \{\ \mathsf{f}^{\mathrm{Nat}\times\mathrm{Nat}\to\mathrm{Nat}}, \mathsf{g}^{\mathrm{Nat}\times\mathrm{Nat}\to\mathrm{Nat}}, \mathsf{s}^{\mathrm{Nat}\to\mathrm{Nat}}, \mathsf{0}^{\mathrm{Nat}}\ \}$ and*

$$\mathcal{R} = \left\{ \begin{array}{lcllcl} \mathsf{f}(\mathsf{0},y) & \to & y & \mathsf{g}(x,\mathsf{0}) & \to & x \\ \mathsf{f}(\mathsf{s}(x),y) & \to & \mathsf{s}(\mathsf{g}(y,x)) & \mathsf{g}(x,\mathsf{s}(y)) & \to & \mathsf{s}(\mathsf{f}(y,x)) \end{array} \right\}.$$

Then functions $\{\mathsf{f} \mapsto 2, \mathsf{g} \mapsto 1\}$ and $\emptyset$ are both downward argument maps compatible with $\mathcal{R}$. In terms of [20, 21], it may possibly be (1) 2 is a downward position of f and 1 is a downward position of g, and (2) both of f and g do not have downward positions. This is why we introduced the notion of downward argument maps as remarked above.

LEMMA 9. *[preservation of a downward position] Suppose that DP is compatible with $\mathcal{R}$. Let z be a fresh variable.*

1. *Let $p \in \mathrm{DPos}(s_g)$ and $s_g \to_{\mathcal{R}} t_g$. Then either (1) $s_g/p \equiv t_g/q$ and $s_g[z]_p \to_{\mathcal{R}} t_g[z]_q$ or (2) $p = q$, $s_g[z]_p \equiv t_g[z]_q$, and $s_g/p \to_{\mathcal{R}} t_g/q$.*
2. *Let $p \in \mathrm{DPos}(s_g)$ and $s_g \xrightarrow{*}_{\mathcal{R}} t_g$. Then there exists $q \in \mathrm{DPos}(t_g)$ such that $s_g[z]_p \xrightarrow{*}_{\mathcal{R}} t_g[z]_q$ and $s_g/p \xrightarrow{*}_{\mathcal{R}} t_g/q$.*

PROOF. 1. Let the redex occurrence of $s_g \to_{\mathcal{R}} t_g$ be p'. If $p' \mid p$ then apparently (1) holds and if $p' \geq p$ then apparently (2) holds. It remains to show the case $p' < p$. Then there exists $f(l_1, \ldots, l_n) \to r \in \mathcal{R}$ and a substitution σ such that $s_g/p' \equiv f(l_1, \ldots, l_n)\sigma$. By $p \in \mathrm{DPos}(s_g)$, $p'.i \leq p$ with $i = DP(f)$, $l_i \equiv x \in V$ is a linear in $f(l_1, \ldots, l_n)$, and there exists a unique $u \in \mathrm{DPos}(r)$ such that $r/u \equiv x$. Then we have $p = p'.i.q'$ for some q'. Let $q = p'.u.q'$. Then $s_g/p \equiv t_g/q$. Since $p \in \mathrm{DPos}(s_g[r\sigma]_{p'}) = \mathrm{DPos}(t_g)$, $q' \in \mathrm{DPos}(x\sigma)$, and $u \in \mathrm{DPos}(r)$, we have $q = p.u.q' \in \mathrm{DPos}(t_g)$. Furthermore, since $l_i \equiv x \in V$ and x is a linear variable in $f(l_1, \ldots, l_n)$ and r, we have $s_g[z]_p \to_{\mathcal{R}} t_g[z]_q$. 2. It follows from 1. ∎

LEMMA 10. *[decomposition at a downward position] Suppose that DP is compatible with $\mathcal{R}$, $t_g, u_g \in \mathrm{T}(\mathcal{F})$ are monomorphic, and $p \in \mathrm{DPos}(t_g)$. Then $t_g[u_g]_p{\downarrow} \equiv t_g[\bot]_p{\downarrow} \otimes u_g{\downarrow}$.*

PROOF. By **Lemma 9**, there exist s_g, v_g, q such that $t_g[u_g]_p{\downarrow} \equiv s_g[v_g]_q$, $q \in \mathrm{DPos}(s_g)$, $t_g[z]_p \xrightarrow{*}_{\mathcal{R}} s_g[z]_q$, and $u_g \xrightarrow{*}_{\mathcal{R}} v_g$. By sufficient completeness, $s_g[v_g]_q \in \mathrm{T}(\mathcal{C})$ and thus $v_g, s_g[\bot]_q \in \mathrm{T}(\mathcal{C})$ and hence $t_g[\bot]_p{\downarrow} \equiv s_g[\bot]_q$, and $u_g{\downarrow} \equiv v_g$. Furthermore, since $q \in \mathrm{DPos}(s_g)$ and $s_g[\bot]_q \in \mathrm{T}(\mathcal{C})$, it follows $q \in \mathrm{RPos}(s_g)$ by the definition of downward position. Hence, by **Lemma 6**, we have $s_g[v_g]_q \equiv s_g[\bot]_q \otimes v_g$. Therefore, $t_g[u_g]_p{\downarrow} \equiv t_g[\bot]_p{\downarrow} \otimes u_g{\downarrow}$. ∎

Example 4 (counterexample) *The lemma above does not hold for the definition of downward position in [21]. Let $\mathcal{S} = \{\mathrm{Nat}\}$, $\mathcal{F} = \{\mathsf{f}^{\mathrm{Nat}\to\mathrm{Nat}}, \mathsf{s}^{\mathrm{Nat}\to\mathrm{Nat}}, \mathsf{0}^{\mathrm{Nat}}\}$, and*

$$\mathcal{R} = \left\{ \begin{array}{lcl} \mathsf{f}(\mathsf{0}) & \to & \mathsf{s}(\mathsf{s}(\mathsf{0})) \\ \mathsf{f}(\mathsf{s}(x)) & \to & \mathsf{s}(\mathsf{s}(x)) \end{array} \right\}.$$

Then we have $\mathsf{f}(\mathsf{s}(\mathsf{0})){\downarrow} \equiv \mathsf{s}(\mathsf{s}(\mathsf{0}))$ and $\mathsf{f}(\mathsf{0}){\downarrow} \otimes \mathsf{s}(\mathsf{0}){\downarrow} \equiv \mathsf{s}(\mathsf{s}(\mathsf{0})) \otimes \mathsf{s}(\mathsf{0}) \equiv \mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{0})))$. Thus $\mathsf{f}(\mathsf{s}(\mathsf{0})){\downarrow} \not\equiv \mathsf{f}(\mathsf{0}){\downarrow} \otimes \mathsf{s}(\mathsf{0})$.

We now describe very roughly how the downward positions can be used to identify the common subterms that can be generalized.

Example 5 *Let $\mathcal{S}, \mathcal{F}, \mathcal{R}$ be as in Example 2. Consider a conjecture e and its generalization e' like this:*

$$e = \mathsf{plus}(s[x]_p, x) \doteq \mathsf{plus}(t[x]_q, x), \qquad e' = \mathsf{plus}(s[x]_p, y) \doteq \mathsf{plus}(t[x]_q, y).$$

*Obviously, if the equation e' is an inductive theorem then the equation e is an inductive theorem (because e is a particular instance of e'). We explain, using the decomposition at a downward position 2, that the other implication also holds. Suppose the equation e is an inductive theorem. Then, by definition, $\mathsf{plus}(s\sigma_g[u_g]_p, u_g) \xleftrightarrow{*}_{\mathcal{R}} \mathsf{plus}(t\sigma_g[u_g]_q, u_g)$ for any ground term u_g and ground substitution σ_g. This means $\mathsf{plus}(s\sigma_g[u_g]_p, u_g){\downarrow} \equiv \mathsf{plus}(t\sigma_g[u_g]_q, u_g){\downarrow}$. Thus, by **Lemma 10**, $\mathsf{plus}(s\sigma_g[u_g]_p, \mathsf{0})){\downarrow} \otimes u_g{\downarrow} \equiv \mathsf{plus}(t\sigma_g[u_g]_q, \mathsf{0}))){\downarrow} \otimes u_g{\downarrow}$, which implies $\mathsf{plus}(s\sigma_g[u_g]_p, \mathsf{0})){\downarrow} \equiv$*

$\mathsf{plus}(t\sigma_g[u_g]_q, 0))\downarrow$. *Then, for any* w_g, $\mathsf{plus}(s\sigma_g[u_g]_p, 0))\downarrow \otimes w_g\downarrow \equiv \mathsf{plus}(t\sigma_g[u_g]_q, 0))\downarrow \otimes w_g\downarrow$. *By* ***Lemma 10****, this implies* e' *is also an inductive theorem.*

Throughout the paper, if no confusion arises, we assume that the downward argument map DP is compatible with the TRS $\mathcal{R}$.

Next, we focus on the dual notion of downward position called upward position. The notion of upward argument position *UP* is the same as the one given in [20, 21]. We, however, additionally introduce a notion of upward position in a term which will be used to extend the definition of contextual positions.

DEFINITION 11.*[upward argument position/upward position] Let* $f \in \mathcal{D}$ *with* $f : \tau_1 \times \cdots \times \tau_n \to \tau$ *and* $1 \leq i \leq n$ *such that* $\tau_i = \tau$ *and* τ *is monomorphic.*

1. *The index* i *is called a upward argument position of* f *(*$UP(f)$*) if for any* $f(l_1, \ldots, l_n) \to r \in \mathcal{R}$, *either* $l_i \equiv \bot^\tau$ *or* $l_i \equiv u[x]_p \in \mathrm{T}(\mathcal{C}, V)$ *and* $r \equiv u[l[x]_i]_p$, *where* $l \equiv f(l_1, \ldots, l_n)$, $p \in \mathrm{RPos}(u)$, *and* x *is a linear variable in* l. *Note that* $p \neq \epsilon$; *for, otherwise* $l \equiv r$ *and contradicts termination of* $\mathcal{R}$.
2. *The set* $\mathrm{UPos}(t)$ *of upward positions in* t *is defined as follows:* $\mathrm{UPos}(t) = \{i\} \cup \{i.p \mid p \in \mathrm{UPos}(t_i)\}$ *if* $t \equiv f(t_1, \ldots, t_n)$ *with* $f \in D$ *and* $i = UP(f)$; $\mathrm{UPos}(t) = \emptyset$ *otherwise.*

The dual property of **Lemma 10** holds for upward positions.

LEMMA 12.*[decomposition at a upward position] Let* $t_g, u_g \in \mathrm{T}(\mathcal{F})$ *be monomorphic terms.*

1. *Let* $t_g \equiv f(t_1, \ldots, t_n)$ *with* $t_j \in \mathrm{T}(\mathcal{C})$ *for all* $1 \leq j \leq n$. *If* $i = UP(f)$ *and* p *be the grp of* t_i *then* $t_g \xrightarrow{*}_{\mathcal{R}} t_i[t_g[\bot]_i]_p$.
2. *If* $i = UP(t_g(\epsilon))$ *then* $t_g[u_g]_i\downarrow \equiv u_g\downarrow \otimes t_g[\bot]_i\downarrow$.
3. *If* $p \in \mathrm{UPos}(t_g)$ *then* $t_g[u_g]_p\downarrow \equiv u_g\downarrow \otimes t_g[\bot]_p\downarrow$.

PROOF.

1. By induction on $|t_i|$.
2. Use confluence, sufficient completeness of $\mathcal{R}$ and 1.
3. By induction on p. Use 2.

∎

Next, we focus on the notion of contextual argument positions. The definition is extended from the original one given in [20, 21].

DEFINITION 13.*[contextual argument position] Let* $f \in \mathcal{D}$ *with* $f : \tau_0 \times \cdots \times \tau_n \to \tau$ *with monomorphic* τ *and* $1 \leq i \leq \mathrm{arity}(f)$ *such that* τ_i *is monomorphic.*

1. *The index* i *is called a down-contextual argument position of* f *(*$DCP(f)$*) if for any* $f(l_1, \ldots, l_n) \to r \in \mathcal{R}$, *either* $l_i \equiv \bot$ *and* $r \equiv \bot$ *hold or* $l_i \equiv u[x]_p \in \mathrm{T}(\mathcal{C}, V)$ *and* $r/q \equiv l[x]_i$, *where* $l \equiv f(l_1, \ldots, l_n)$, $p \in \mathrm{RPos}(u)$, $q \in \mathrm{UPos}(r)$, *and* x *is a linear variable in* l *and* r. *Note that* $p \neq \epsilon$; *for, otherwise* $r/q \equiv l$ *and contradicts termination of* $\mathcal{R}$.
2. *The index* i *is called an up-contextual argument position of* f *(*$UCP(f)$*) if for any* $f(l_1, \ldots, l_n) \to r \in \mathcal{R}$, *either* $l_i \equiv \bot$ *and* $r \equiv \bot$ *hold or* $l_i \equiv u[x]_p \in \mathrm{T}(\mathcal{C}, V)$ *and* $r/q \equiv l[x]_i$, *where* $l \equiv f(l_1, \ldots, l_n)$, $p \in \mathrm{RPos}(u)$, $q \in \mathrm{DPos}(r)$, *and* x *is a linear variable in* l *and* r. *Note that* $p \neq \epsilon$; *for, otherwise* $r/q \equiv l$ *and contradicts termination of* $\mathcal{R}$.

The original definition of contextual positions use the conditions $q = UP(r(\epsilon))$ and $q = DP(r(\epsilon))$ instead of $q \in \mathrm{UPos}(r)$ and $q \in \mathrm{DPos}(r)$, respectively. Furthermore, when $l_i \equiv \bot$ and $r \equiv \bot$, $\mathrm{sort}(l_i) = \mathrm{sort}(r)$ (and hence $l_i \equiv r$) is required in the original definition. Since $UP(r(\epsilon)) \in \mathrm{UPos}(r)$ and $DP(r(\epsilon)) \in \mathrm{DPos}(r)$, our definition enlarges the scope of the contextual positions.

Example 6 *Let* $\mathcal{S} = \{\mathrm{Nat}, \mathrm{List}\}$, $\mathcal{F} = \{\mathsf{dbl}^{\mathrm{Nat}\to\mathrm{Nat}}, \mathsf{len}^{\mathrm{List}\to\mathrm{Nat}}, \mathsf{sum}^{\mathrm{List}\to\mathrm{Nat}}, \mathsf{plus}^{\mathrm{Nat}\times\mathrm{Nat}\to\mathrm{Nat}}, \mathsf{cons}^{\mathrm{Nat}\times\mathrm{List}\to\mathrm{List}}, \mathsf{nil}^{\mathrm{List}}, \mathsf{s}^{\mathrm{Nat}\to\mathrm{Nat}}, \mathsf{0}^{\mathrm{Nat}}\}$, *and*

$$\mathcal{R} = \left\{ \begin{array}{lcllcl} \mathsf{len}(\mathsf{nil}) & \to & \mathsf{0} & \mathsf{dbl}(\mathsf{0}) & \to & \mathsf{0} \\ \mathsf{len}(\mathsf{cons}(x, xs)) & \to & \mathsf{s}(\mathsf{len}(xs)) & \mathsf{dbl}(\mathsf{s}(x)) & \to & \mathsf{s}(\mathsf{s}(\mathsf{dbl}(x))) \\ \mathsf{plus}(x, \mathsf{0}) & \to & x & \mathsf{sum}(\mathsf{nil}) & \to & \mathsf{0} \\ \mathsf{plus}(x, \mathsf{s}(y)) & \to & \mathsf{s}(\mathsf{plus}(x, y)) & \mathsf{sum}(\mathsf{cons}(x, xs)) & \to & \mathsf{plus}(x, \mathsf{sum}(xs)) \end{array} \right\}$$

Then we have $1 = UCP(\mathsf{len})$, $1 = UCP(\mathsf{dbl})$, *and* $1 = DCP(\mathsf{sum})$. *In the original definition in [20, 21], however, none of these are defined.*

LEMMA 14. *[decomposition at contextual positions] Let* $t_g \in \mathrm{T}(\mathcal{F})$, $u_g, v_g \in \mathrm{T}(\mathcal{C})$ *be monomorphic terms and* $f = t_g(\epsilon)$.

1. *If* $i = DCP(f)$ *then* $t_g[\bot]_i\downarrow \equiv \bot$.
2. *If* $i = DCP(f)$ *then* $t_g[u_g \otimes v_g]_i\downarrow \equiv t_g[v_g]_i\downarrow \otimes t_g[u_g]_i\downarrow$.
3. *If* $i = UCP(f)$ *then* $t_g[\bot]_i\downarrow \equiv \bot$.
4. *If* $i = UCP(f)$ *then* $t_g[u_g \otimes v_g]_i\downarrow \equiv t_g[u_g]_i\downarrow \otimes t_g[v_g]_i\downarrow$.

PROOF.

1. Straightforward.
2. By induction on $|u_g|$. Use **Lemma 6** and **Lemma 12**.
3. Same as 1 except using $i = UCP(f)$ instead of $i = DCP(f)$.
4. Same as 2 except using **Lemma 10** instead of **Lemma 12**.

∎

4 Term Partition and Sound Generalization

Based on the characterization of five types of argument positions, Urso and Kounalis ([20, 21]) developed techniques useful in inductive theorem proving—namely, term partition and sound generalization. These techniques rely on the following observation.

DEFINITION 15. *[term partition[20, 21]] Let* $\mathcal{R}$ *be a sufficiently complete, confluent, terminating TRS.* $\langle s_0, s_1\rangle$ *is said to be a term partition of* s *if (1)* s_0 *and* s_1 *have the same monomorphic sort* τ *and (2) for any ground substitution* θ_g, $s_0\theta_g\downarrow \otimes s_1\theta_g\downarrow \equiv s\theta_g\downarrow$.

PROPOSITION 16. *[term partition theorem (Theorem 1 of [20])] Let* $\mathcal{R}$ *be a sufficiently complete, confluent, terminating TRS. Suppose* $\langle s_0, s_1\rangle$ *is a term partition of* s *and* $\langle t_0, t_1\rangle$ *is a term partition of* t. *Then for each* $i \in \{0, 1\}$, *if* $\mathcal{R} \vdash_{ind} s_i \doteq t_i$ *then we have* $\mathcal{R} \vdash_{ind} s \doteq t$ *iff* $\mathcal{R} \vdash_{ind} s_{1-i} \doteq t_{1-i}$.

In [20, 21], Urso and Kounalis introduced a notion of prominent paths (called *top path* and *bottom path*) based on the five types of argument positions of functions and a method to compute some term partitions based on these paths.

DEFINITION 17. *[top/bottom paths[20, 21]] Let t be a monomorphic term. The set* $\mathrm{TPath}(t)$ *of top paths in a term t and the set* $\mathrm{BPath}(t)$ *of bottom paths in a term t are defined as follows:*

$$\mathrm{TPath}(t) = \begin{cases} \{\epsilon\} \cup \{i.p \mid p \in \mathrm{TPath}(t_i)\} & \text{if } t \equiv f(t_1,\ldots,t_n),\ i = RA(f) \\ \{\epsilon\} \cup \{i.p \mid p \in \mathrm{TPath}(t_i)\} & \text{if } t \equiv f(t_1,\ldots,t_n),\ i = UP(f) \\ \{\epsilon\} \cup \{i.p \mid p \in \mathrm{BPath}(t_i)\} & \text{if } t \equiv f(t_1,\ldots,t_n),\ i = DCP(f) \\ \{\epsilon\} \cup \{i.p \mid p \in \mathrm{TPath}(t_i)\} & \text{if } t \equiv f(t_1,\ldots,t_n),\ i = UCP(f) \\ \{\epsilon\} & \text{otherwise} \end{cases}$$

$$\mathrm{BPath}(t) = \begin{cases} \{\epsilon\} \cup \{i.p \mid p \in \mathrm{BPath}(t_i)\} & \text{if } t \equiv f(t_1,\ldots,t_n),\ i = RA(f) \\ \{\epsilon\} \cup \{i.p \mid p \in \mathrm{BPath}(t_i)\} & \text{if } t \equiv f(t_1,\ldots,t_n),\ i = DP(f) \\ \{\epsilon\} \cup \{i.p \mid p \in \mathrm{TPath}(t_i)\} & \text{if } t \equiv f(t_1,\ldots,t_n),\ i = DCP(f) \\ \{\epsilon\} \cup \{i.p \mid p \in \mathrm{BPath}(t_i)\} & \text{if } t \equiv f(t_1,\ldots,t_n),\ i = UCP(f) \\ \{\epsilon\} & \text{otherwise} \end{cases}$$

Clearly, $\mathrm{TPath}(t)$ *and* $\mathrm{BPath}(t)$ *are totally ordered w.r.t.* $\leq$ *and the greatest element in* $\mathrm{TPath}(t)$ *and* $\mathrm{BPath}(t)$ *are called the maximum top path and the maximum bottom path and denoted by $TP(t)$ and $BP(t)$, respectively.*

DEFINITION 18. *[head/tail parts[20, 21]] Let t be a monomorphic term. For each $p \in \mathrm{TPath}(t)$, its head context $Ctop_{t,p}$ as well as for each $q \in \mathrm{BPath}(t)$, its tail context $Cbot_{t,q}$ are defined as follows:*

$$Ctop_{t,p} = \begin{cases} \Box & \text{if } p = \epsilon \\ t[Ctop_{t_i,p'}]_i & \text{if } p = i.p',\ t \equiv f(t_1,\ldots,t_n), \text{ and } i = RA(f) \\ Ctop_{t_i,p'} & \text{if } p = i.p',\ t \equiv f(t_1,\ldots,t_n), \text{ and } i = UP(f) \\ t[Cbot_{t_i,q'}]_i & \text{if } p = i.q',\ t \equiv f(t_1,\ldots,t_n), \text{ and } i = DCP(f) \\ t[Ctop_{t_i,p'}]_i & \text{if } p = i.p',\ t \equiv f(t_1,\ldots,t_n), \text{ and } i = UCP(f) \end{cases}$$

$$Cbot_{t,q} = \begin{cases} \Box & \text{if } q = \epsilon \\ Cbot_{t_i,q'} & \text{if } q = i.q',\ t \equiv f(t_1,\ldots,t_n) \text{ and } i = RA(f) \\ Cbot_{t_i,q'} & \text{if } q = i.q',\ t \equiv f(t_1,\ldots,t_n) \text{ and } i = DP(f) \\ t[Ctop_{t_i,p'}]_i & \text{if } q = i.p',\ t \equiv f(t_1,\ldots,t_n) \text{ and } i = DCP(f) \\ t[Cbot_{t_i,q'}]_i & \text{if } q = i.q',\ t \equiv f(t_1,\ldots,t_n) \text{ and } i = UCP(f) \end{cases}$$

The head part is given by $top(t,p) \equiv Ctop_{t,p}[t/p]$ and the tail part is by $bot(t,q) \equiv Cbot_{t,q}[t/q]$.

The development of the term patition based on prominent paths is solely based on the characterization lemmas for five types of argument positions. Thus this term partition can be corrected and extended based on our revised definition of monomorphic signature and argument positions given in the previous section. We refer to [20, 21] the definition of $ntp(t,p)$ and $nbt(t,q)$ in the following proposition.

PROPOSITION 19. *[term partition via prominent path (Theorem 36 of [21])] Let $\mathcal{R}$ be a sufficiently complete, terminating, and confluent TRS. Let t be a monomorphic term. (1) For*

each top path p in t, $\langle top(t,p), ntp(t,p)\rangle$ is a term partition of t. (2) For each bottom path q in t, $\langle nbt(t,q), bot(t,q)\rangle$ is a term partition of t.

The sound generalization is obtained from **Proposition 19**.

PROPOSITION 20.*[sound generalization theorem (Theorem 37 of [21])] Let $\mathcal{R}$ be a sufficiently complete, terminating, and confluent TRS. Let $s \doteq t$ be a monomorphic equation and x be a fresh variable.*

1. *Let p be a top path in s and q a top path in t. Suppose that $s/p \equiv t/q$ and $top(s,p) \equiv top(t,q)$. Then $\mathcal{R} \vdash_{ind} s \doteq t$ iff $\mathcal{R} \vdash_{ind} s[x]_p \doteq t[x]_q$.*
2. *Let p be a bottom path in s and q a bottom path in t. Suppose that $s/p \equiv t/q$ and $bot(s,p) \equiv bot(t,q)$. Then $\mathcal{R} \vdash_{ind} s \doteq t$ iff $\mathcal{R} \vdash_{ind} s[x]_p \doteq t[x]_q$.*

5 Conclusion

We presented an example showing that the sound generalization proposed in [21] does not work without a condition imposed in their previous paper [20] that proposes induction based on term partition. We restored a condition in the definition of one of the argument positions and gave the corrected proof of the characterization of the position. Based on this, the correctness of sound generalization [21] was recovered. We note that all examples of sound genereralization presented in [21] still works under the restored condition. We also extended the technique by eliminating one of the restriction of monomorphic signature, localizing a part of the conditions for target term rewriting systems, and extending the notion of contextual positions. The corrected part of sound generalization is implemented in our experimental induction prover based on rewriting induction [1].

Despite the relaxation, some strong restrictions of monomorphicness are still imposed on the sound generalization. Finding other types of sound generalization applicable for non-monomorphic equations remains as a future work. Another future work is obtaining a lemma discovery method other than term partition and sound generalization via deeper analysis of monomorphicness.

Acknowledgments

Thanks are due to anonymous referees for helpful comments. This work was partially supported by a grant from JSPS, No. 20500002.

References

[1] T. Aoto. Designing a rewriting induction prover with an increased capability of non-orientable equations. In *Proc. of Symbolic Computation in Software Science Austrian-Japanese Workshop*, volume 08-08 of *RISC Technical Report*, pages 1–15, 2008.

[2] F. Baader and T. Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.

[3] A. Bouhoula, E. Kounalis, and M. Rusinowitch. Automated mathematical induction. *Journal of Logic and Computation*, 5(5):631–668, 1995.

[4] R. Boyer and J. Moore. *A Computational Logic*. Academic Press, 1979.
[5] A. Bundy. The automation of proof by mathmatical induction. In *Handbook of Automated Reasoning*, volume 1, pages 845–908. MIT Press, 2001.
[6] A. Bundy, D. Basin, D. Hutter, and A. Ireland. *Rippling: Meta-Level Guidance for Mathematical Reasoning*. Cambridge University Press, 2005.
[7] H. Comon. Inductionless induction. In *Handbook of Automated Reasoning*, volume 1, pages 913–962. MIT Press, 2001.
[8] B. Gramlich. Strategic issues, problems and challenges in inductive theorem proving. *Electronic Notes in Theoretical Computer Science*, 125:5–43, 2005.
[9] G. Huet and J.-M. Hullot. Proof by induction in equational theories with constructors. *Journal of Computer and System Sciences*, 25(2):239–266, 1982.
[10] G. Huet and D. C. Oppen. Equations and rewrite rules: a survey. Technical report, Stanford University, Stanford, CA, USA, 1980.
[11] D. Hutter and C. Sengler. INKA: The next generation. In *Proc. of the 13th International Conference on Automated Deduction*, pages 288–292, 1996.
[12] A. Ireland and A. Bundy. Productive use of failure in inductive proof. *Journal of Automated Reasoning*, 16(1–2):79–111, 1996.
[13] D. Kapur, J. Giesl, and M. Subramaniam. Induction and decision procedures. *Revista de la real academia de ciencas (RACSAM) Serie A: Matematicas*, 98(1):154–180, 2004.
[14] D. Kapur, P. Narendran, and H. Zhang. Automating inductionless induction using test sets. *Journal of Symbolic Computation*, 11(1–2):81–111, 1991.
[15] D. Kapur and M. Subramaniam. Lemma discovery in automating induction. In *Proc. of the 13th International Conference on Automated Deduction*, volume 1104 of *LNCS*, pages 538–552. Springer-Verlag, 1996.
[16] D. Kapur and H. Zhang. An overview of rewrite rule laboratory (RRL). *Journal of Computer Mathematics with Applications*, 29(2):91–114, 1995.
[17] M. Kaufmann, P. Manolios, and J. S. Moore. *Computer-Aided Reasoning: ACL2 Case Studies*. Kluwer Academic Publishers, 2000.
[18] S. Shimazu, T. Aoto, and Y. Toyama. Automated lemma generation for rewriting induction with disproof. In *Proc. of the 8th JSSST Workshop on Programming and Programming Languages*, pages 75–89, 2006. In Japanese.
[19] S. Stratulat. A general framework to build contextual cover set induction provers. *Journal of Symbolic Computation*, 32:403–445, 2001.
[20] P. Urso and E. Kounalis. Term partition for mathematical induction. In *Proc. of the 14th International Conference on Rewriting Techniques and Applications*, volume 2706 of *LNCS*, pages 352–366, 2003.
[21] P. Urso and E. Kounalis. Sound generalizations in mathematical induction. *Theoretical Computer Science*, 323:443–471, 2004.
[22] T. Walsh. A divergence critic for inductive proof. *Journal of Artificial Intelligence Research*, 4:209–235, 1996.

Foundations of Software Technology and Theoretical Computer Science (Bangalore) 2008.
Editors: R. Hariharan, M. Mukund, V. Vinay; pp 25-36

Some Sieving Algorithms for Lattice Problems

V. Arvind and Pushkar S. Joglekar
Institute of Mathematical Sciences
C.I.T Campus,Chennai 600 113, India
{arvind,pushkar}@imsc.res.in

ABSTRACT. We study the algorithmic complexity of lattice problems based on the sieving technique due to Ajtai, Kumar, and Sivakumar [AKS01]. Given a k-dimensional subspace $M \subseteq \mathbb{R}^n$ and a full rank integer lattice $\mathcal{L} \subseteq \mathbb{Q}^n$, the *subspace avoiding problem* SAP, defined by Blömer and Naewe [BN07], is to find a shortest vector in $\mathcal{L} \setminus M$. We first give a $2^{O(n+k\log k)}$ time algorithm to solve *the subspace avoiding problem*. Applying this algorithm we obtain the following results.

1. We give a $2^{O(n)}$ time algorithm to compute i^{th} successive minima of a full rank lattice $\mathcal{L} \subset \mathbb{Q}^n$ if i is $O(\frac{n}{\log n})$.
2. We give a $2^{O(n)}$ time algorithm to solve a restricted *closest vector problem CVP* where the inputs fulfil a promise about the distance of the input vector from the lattice.
3. We also show that unrestricted CVP has a $2^{O(n)}$ exact algorithm if there is a $2^{O(n)}$ time exact algorithm for solving CVP with additional input $v_i \in \mathcal{L}, 1 \leq i \leq n$, where $\|v_i\|_p$ is the i^{th} successive minima of $\mathcal{L}$ for each i.

We also give a new approximation algorithm for SAP and the *Convex Body Avoiding problem* which is a generalization of SAP. Several of our algorithms work for *gauge* functions as metric, where the gauge function has a natural restriction and is accessed by an oracle.

1 Introduction

Fundamental algorithmic problems concerning integer lattices are the shortest vector problem (SVP) and the closest vector problem(CVP). Given a lattice $\mathcal{L} \subset \mathbb{R}^n$ by a basis, the shortest vector problem (SVP) is to find a shortest nonzero vector in $\mathcal{L}$ w.r.t. some metric given by a *gauge* function in general (usually the ℓ_p norm for some p). Likewise, the closest vector problem (CVP) takes as input a lattice $\mathcal{L} \subset \mathbb{R}^n$ and vector $v \in \mathbb{R}^n$ and asks for a $u \in \mathcal{L}$ closest to v w.r.t. a given metric. These problems have polynomial-time approximation algorithms based on the celebrated LLL algorithm for basis reduction [LLL82].

The fastest known exact deterministic algorithms for SVP and CVP have running time $2^{O(n\log n)}$ [Kan87] (also see [Bl00]). More recently, Ajtai, Kumar and Sivakumar in a seminal paper [AKS01] gave a $2^{O(n)}$ time *randomized* exact algorithm for SVP. Subsequently, in [AKS02] they gave a $2^{O(n)}$ time randomized approximation algorithm for CVP. Their algorithms are based on a generic sieving procedure (introduced by them) that exploits the underlying geometry. Recently, Blömer and Naewe [BN07] gave a different $2^{O(n)}$ time randomized approximation algorithm for CVP, also based on the AKS sieving technique.

For $1 \leq i \leq n$, the i^{th} *successive minima* $\lambda_i(\mathcal{L})$ is defined as the smallest r such that a ball of radius r around origin contains at least i linearly independent lattice vectors. The successive minimas $\lambda_i(\mathcal{L})$ are important lattice parameters. A classical problem is the *successive minima problem* SMP of finding for a given lattice $\mathcal{L}$, n linearly independent vectors

FSTTCS 2008
IARCS Annual Conference on
Foundations of Software Technology and Theoretical Computer Science
http://drops.dagstuhl.de/opus/volltexte/2008/1738

$v_1, v_2, \ldots, v_n \in \mathcal{L}$ such that $\|v_i\|$ is at most $\lambda_i(\mathcal{L})$. This problem clearly subsumes the *shortest independent vectors problem* SIVP where one wants to find linearly independent vectors $v_1, v_2, \ldots, v_n \in \mathcal{L}$ such that $\|v_i\| \leq \lambda_n(\mathcal{L})$. Given a k-dimensional subspace $M \subseteq \mathbb{R}^n$ and a full rank integer lattice $\mathcal{L} \subseteq \mathbb{Q}^n$, the *subspace avoiding problem* SAP, is to find a shortest vector in $\mathcal{L} \setminus M$. The paper [BN07] gives $2^{O(n)}$ time approximation algorithm for these problems.

No exact $2^{O(n)}$ time randomized algorithm is known for CVP or SMP. Recently, Micciancio has shown [Mi08] that CVP is polynomial-time equivalent to several lattice problems, including SIVP and SMP, under deterministic polynomial time rank-preserving reductions. This perhaps explains the apparent difficulty of finding a $2^{O(n)}$ time exact algorithm for CVP or SMP, because SVP reduces to all of these problems but no reduction is known in the other direction. In particular, the reductions in [Mi08] yield $2^{O(n \log n)}$ time exact algorithms for SAP, SMP and SIVP, whereas [BN07] gives $2^{O(n)}$ time randomized approximation algorithm for these problems.

Our results

In this paper we consider some natural restrictions of these problems that can be exactly solved in $2^{O(n)}$ time. We obtain these results giving a $2^{O(n+k \log k)}$ algorithm to solve SAP where n is the rank of the lattice and k is the dimension of the subspace.

As our first result we show that given a full rank lattice $\mathcal{L} \subset \mathbb{Q}^n$ there is $2^{O(n)}$ time randomized algorithm to compute linearly independent vectors $v_1, v_2, \ldots, v_i \in \mathcal{L}$ such that $\|v_i\| = \lambda_i(\mathcal{L})$ if i is $O(\frac{n}{\log n})$. Given a full rank lattice $\mathcal{L} \subset \mathbb{Q}^n$ and $v \in \mathbb{Q}^n$ we also give a $2^{O(n)}$ time algorithm to solve CVP$(\mathcal{L}, v)$ if the input $(v, \mathcal{L})$ fulfils the promise $d(v, \mathcal{L}) \leq \frac{\sqrt{3}}{2}\lambda_{O(\frac{n}{\log n})}(\mathcal{L})$.

We show that CVP can be solved in $2^{O(n)}$ time if there is a $2^{O(n)}$ time algorithm to compute a closest vector to v in $\mathcal{L}$ where $v \in \mathbb{Q}^n$, $\mathcal{L} \subset \mathbb{Q}^n$ is a full rank lattice and $v_1, v_2, \ldots, v_n \in \mathcal{L}$ such that $\|v_i\|_p$ is equal to i^{th} successive minima of $\mathcal{L}$ for $i = 1$ to n are given as an additional input to the algorithm. As a consequence, we can assume that successive minimas are given for free as an input to the algorithm for CVP. We believe that using basis reduction techniques from [Kan87] one might be able to exploit the information about successive minimas of the lattice to get a better algorithm for CVP.

We give a new $2^{O(n+k \log 1/\epsilon)}$ time randomized algorithm to solve $1+\epsilon$ approximation of SAP, where n is rank of the lattice and k is the dimension of subspace. We get better approximation guarantee than the one in [BN07] parametrised on k. We also consider a generalization of SAP (the *convex body avoiding* problem) and give a singly exponential approximation algorithm for the problem.

2 Preliminaries

A lattice $\mathcal{L}$ is a discrete additive subgroup of R^n, n is called dimension of the lattice. For algorithmic purposes we can assume that $\mathcal{L} \subseteq \mathbb{Q}^n$, and even in some cases $\mathcal{L} \subseteq \mathbb{Z}^n$. A lattice is usually specified by a basis $B = \{b_1, \cdots, b_m\}$, where $b_i \in \mathbb{Q}^n$ and b_i's are linearly independent. m is called the rank of the lattice. If the rank is n the lattice is said to be a *full rank* lattice. Although most results in the paper hold for general lattices, for convenience we

mainly consider only full-rank lattices. For $x \in \mathbb{Q}^n$ let size(x) denote the number of bits for the standard binary representation as an n-tuple of rationals. Let size($\mathcal{L}$) denote $\sum_i$ size(b_i). Next we recall the definition of gauge functions.

DEFINITION 1.[Si45] *A function $f : \mathbb{R}^n \to \mathbb{R}$ is called a* gauge function *if it satisfies following properties:*

1. $f(x) > 0$ *for all* $x \in \mathbb{R}^n \setminus \{0\}$ *and* $f(x) = 0$ *if* $x = 0$.
2. $f(\lambda x) = \lambda f(x)$ *for all* $x \in \mathbb{R}^n$ *and* $\lambda \in \mathbb{R}$.
3. $f(x + y) \leq f(x) + f(y)$ *for all* $x, y \in \mathbb{R}^n$.

For $v \in \mathbb{R}^n$ we denote $f(v)$ by $\|v\|_f$ and call it norm of v with respect to the gauge function f. It is easy to see that any l_p norm satisfies all the above properties. Thus gauge functions generalize the usual l_p norms. A gauge function f defines a natural metric d_f on $\mathbb{R}^n$ by setting $d_f(x, y) = f(x - y)$ for $x, y \in \mathbb{R}^n$. For $x \in \mathbb{R}^n$ and $r > 0$, let $B_f(x, r)$ denote the f-ball of radius r with center x with respect to the gauge function f, defined as $B_f(x, r) = \{y \in \mathbb{R}^n | f(x - y) \leq r\}$. We denote the metric balls with respect to usual l_p norm by $B_p(x, r)$. Unless specified otherwise we always consider balls in $\mathbb{R}^n$. The next well-known proposition characterizes the class of all gauge functions.

PROPOSITION 2.[Si45] *Let $f : \mathbb{R}^n \to \mathbb{R}$ be any gauge function then a unit radius ball around origin with respect to f is a n dimensional bounded O-symmetric convex body. Conversely, for any n dimensional bounded O-symmetric convex body C, there is a gauge function $f : \mathbb{R}^n \to \mathbb{R}$ such that $B_f(0, 1) = C$.*

Given an f-ball of radius r around origin with respect to a gauge function f, from the Proposition 2 it follows that $B_f(0, r)$ is an O-symmetric convex body. It is easy to check that for any $r > 0$ and any constant c we have $\text{vol}(B_f(0, cr)) = c^n \text{vol}(B_f(0, r))$, where vol($C$) denotes the volume of the corresponding convex body C (see e.g. [Si45]).

We now place a natural restriction on gauge functions. A gauge function f, given by oracle access, is a *nice gauge function* if it satisfies the following property: For some polynomial $p(n)$, $B_2(0, 2^{-p(n)}) \subseteq B_f(0, 1) \subseteq B_2(0, 2^{p(n)})$, i.e. there exists a Euclidean sphere of radius $2^{-p(n)}$ inside the convex body $B_f(0, 1)$, and $B_f(0, 1)$ is contained inside a Euclidean sphere of radius $2^{p(n)}$. Note that if f is a nice gauge function and $v \in \mathbb{Q}^n$ we have size($f(v)$)=poly(n,size(v)). For a nice gauge function f we can sample points from convex body $B_f(0, r)$ almost uniformly at random in poly(size(r),n) time using the Dyer-Frieze-Kannan algorithm [DFK91]. It is easy to check that all l_p norms $p \geq 1$ define nice gauge functions. The i^{th} successive minima of a lattice $\mathcal{L}$ with respect to ℓ_p norm is smallest $r > 0$ such that $B_p(0, r)$ contains at least i linearly independent lattice vectors. It is denoted by $\lambda_i^p(\mathcal{L})$.

Remarks: In this paper we consider lattice problems with respect to nice gauge functions. Let $\mathcal{L}$ be a lattice with basis $\{b_1, b_2, \ldots, b_n\}$ and f be a nice gauge function. Suppose B is a full rank $n \times n$ matrix with columns $b_1, b_2, \ldots, b_n$. Note that the linear transformation B^{-1} maps lattice $\mathcal{L}$ isomorphically to the standard lattice $\mathbb{Z}^n$. Furthermore, it is easy to see that the set $C = B^{-1}(B_f(0, 1))$ is an O-symmetric convex body. Hence, by Proposition 2 it follows that $C = B_g(0, 1)$ for some gauge function g. As f is a nice gauge function, it easily follows that g is also a nice gauge function.

Thus, our algorithms that work for nice gauge functions can be stated for the the standard lattice $\mathbb{Z}^n$ and a nice gauge function g. However, some of our results hold only for ℓ_p norms. Thus, to keep uniformity we allow our algorithms to take arbitrary lattices as input even when the metric is give by a nice gauge function.

3 A Sieving Algorithm for SAP

In this section we present a different analysis of the AKS sieving [AKS01, Re04] applied to the Subspace Avoiding Problem (SAP). Our analysis is quite different from that due to Blömer and Naewe [BN07] and gives us improved running time for computing a $1+\epsilon$ approximate solution.

Recall that an input instance of the subspace avoiding problem (SAP) consists of $(\mathcal{L}, M)$ where $\mathcal{L} \subset \mathbb{Q}^n$ is a full rank lattice and $M \subset \mathbb{R}^n$ is a subspace of dimension k. The SAP problem is to find a vector $v \in \mathcal{L} \setminus M$ with least norm with respect to a nice gauge function f.

We give an intuitive outline of our approximation algorithm: Our analysis of AKS sieving will use the fact that the sublattice $\mathcal{L} \cap M$ of $\mathcal{L}$ is of rank k. We will use the AKS sieving procedure to argue that we can sample $2^{O(n+k\log(1/\epsilon))}$ points from *some* coset of $\mathcal{L} \cap M$ in $2^{O(n+k\log(1/\epsilon))}$ time. We can then apply a packing argument in the coset (which is only k-dimensional) to obtain points in the coset that are close to each other. Then, with a standard argument following the original AKS result [AKS01] we can conclude that their differences will contain a good approximation.

Suppose, without loss of generality, that the input lattice $\mathcal{L} \subseteq \mathbb{R}^n$ is n-dimensional given by a basis $\{b_1, \cdots, b_n\}$, so that $\mathcal{L} = \sum_{i=1}^{n} \mathbb{Z} \cdot b_i$. Let us fix a nice gauge function f and let $v \in \mathcal{L}$ denote a shortest vector in $\mathcal{L} \setminus M$ with respect to gauge function f, i.e. $f(x)$ for $x \in \mathcal{L} \setminus M$ attains minimum value at $x = v$. Let $s = \text{size}(\mathcal{L}, \text{M})$ denote the input size (which is the number of bits for representing the vectors b_i and the basis for M). As v is a shortest vector in $\mathcal{L} \setminus M$ and f is a nice gauge function it is quite easy to see that $size(f(v))$ is bounded by a polynomial in s. Thus, we can scale the lattice $\mathcal{L}$ to ensure that $2 \leq f(v) \leq 3$. More precisely, we can compute polynomially many scaled lattices from $\mathcal{L}$, so that $2 \leq f(v) \leq 3$ holds for at least one scaled lattice. Thus, we can assume that $2 \leq f(v) \leq 3$ holds for the lattice $\mathcal{L}$.

We first describe the AKS sieving procedure [AKS01] for any gauge function, analyze its running time and explain its key properties. The following lemma is crucially used in the algorithm.

LEMMA 3. *[Sieving Procedure] Let $f : \mathbb{R}^n \to \mathbb{R}$ be any gauge function. Then there is a sieving procedure that takes as input a finite set of points $\{\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3, \ldots, \mathbf{v}_N\} \subseteq B_f(0, r)$, and in $N^{O(1)}$ time it outputs a subset of indices $S \subset [N]$ such that $|S| \leq 5^n$ and for each $i \in [N]$ there is a $j \in S$ with $f(\mathbf{v}_i - \mathbf{v}_j) \leq r/2$.*

Proof. The sieving procedure is exactly as described in Regev's lecture notes [Re04]. The sieving procedure is based on a simple greedy strategy. We start with $S = \emptyset$ and run the following step for all elements $v_i, 1 \leq i \leq N$. At the i^{th} step we consider v_i. If $f(v_i - v_j) > r/2$ for all $j \in S$ include i in the set S and increment i. After completion, for all $i \in [N]$

there is a $j \in S$ such that $f(v_i - v_j) \leq r/2$. The bound on $|S|$ follows from a packing argument combined with the fact that $\text{vol}(B_f(0, cr)) = c^n \text{vol}(B_f(0, r))$ for any $r > 0$ and a constant $c > 0$. More precisely, for any two points $v_i, v_j \in S$ we have $f(v_i - v_j) > r/2$. Thus, all the convex bodies $B_f(v_i, r/4)$ for $v_i \in S$ are mutually disjoint and are contained in $B_f(0, r + r/4)$. Also note that $\text{vol}(B_f(0, dr)) = d^n \text{vol}(B_f(0, r))$ for any constant $d > 0$. It follows that $5^n \text{vol}(B_f(v_i, r/4)) \geq \text{vol}(B_f(0, r + r/4))$. Hence, $|S| \leq 5^n$. The second property of S is guaranteed by the sieving procedure. ■

Next, our algorithm follows the usual AKS random sampling procedure. Let $R = n \cdot max_i \|b_i\|_f$. It is clear that size($R$) is polynomial in s since f is a nice gauge function. Let $B_f(0, 2)$ denote the f-ball of radius 2 around the origin. Since we have an oracle for membership in $B_f(0, 2)$ and f is a nice gauge function we can almost uniformly sample from $B_f(0, 2)$ using the Dyer-Frieze-Kannan algorithm [DFK91]. Let $x_1, x_2, \cdots, x_N$ denote such a random sample, for $N = 2^{c \cdot (n + k \log(1/\epsilon))} \cdot \log R$ where the constant $c > 0$ will be suitably chosen. Now, using the lattice $\mathcal{L}$ we can round off the points x_i. More precisely, we express $x_i = \Sigma_j \alpha_{ij} b_j$ for rationals α_{ij}. Then, from each vector x_i we compute the vector $y_i = \Sigma_j \beta_{ij} b_j$, where $0 \leq \beta_{ij} < 1$, by adding appropriate integral multiples of the b_j's to the expression for x_i. Thus, the points $y_1, \cdots, y_N$ are in the interior of the fundamental parallelepiped of $\mathcal{L}$, and each $x_i - y_i \in \mathcal{L}$. We denote this by $y_i = x_i (\text{mod } \mathcal{L})$. We now have the set of N pairs $P = \{(x_i, y_i) \mid i \in [N]\}$, where $x_i - y_i$ are lattice points. Since y_i lie inside the fundamental parallelepiped we have $\|y_i\|_f \leq n \cdot max_i \|b_i\|_f = R$ for $i = 1$ to N.

Now, we apply the AKS sieving procedure in Lemma 3 to the set $\{y_1, y_2, \cdots, y_N\}$. The result is a subset $S \subset [N]$ of at most 5^n indices such that for each $i \in [N]$ there is some $j \in S$ such that $f(y_i - y_j) \leq R/2$. We remove from P all (x_j, y_j) for $j \in S$ and replace each remaining $(x_i, y_i) \in P$ by a corresponding $(x_i, y_i - (y_j - x_j))$, where $j \in S$ is the first index such that $f(y_i - y_j) \leq R/2$. After the sieving round, the set P has the property that for each $(x_i, z_i) \in P$ we have $x_i - z_i \in \mathcal{L}$ and $f(x_i - z_i) \leq 4 + R/2$, and P has shrunk in size by at most 5^n. We continue with $O(\log\ R)$ sieving rounds so that we are left with a set P with $N - O(\log\ R)5^n$ pairs (x_i, z_i) such that $x_i - z_i \in \mathcal{L}$ and $f(x_i - z_i) \leq 8$. We can ensure that $|P| \geq 2^{c'(n + k \log(1/\epsilon))}$ for an arbitrary constant c' by appropriately choosing constant c. The vectors, $x_i - z_i$ for $(x_i, z_i) \in P$ follows some distribution among lattice points inside $B_f(0, 8)$. Next, we need following simple proposition.

PROPOSITION 4. *Let $\mathcal{L} \subset \mathbb{R}^n$ be a rank n lattice, $v \in \mathcal{L}$ such that $2 \leq f(v) \leq 3$ for a nice gauge function f. Consider the convex regions $C = B_f(-v, 2) \cap B_f(0, 2)$ and $C' = B_f(v, 2) \cap B_f(0, 2)$. Then $C' = C + v$ and $vol(C) = vol(C') = \Omega(\frac{vol(B_f(0,2))}{2^{O(n)}})$.*

Proposition 4 is easy to prove since $B_f(-v/2, 1/2) \subseteq C, B_f(v/2, 1/2) \subseteq C'$. Note that we have picked $x_1, \ldots, x_N$ uniformly at random from $B_f(0, 2)$,where $N = 2^{c \cdot (n + k \log(1/\epsilon))} \cdot \log R$. By Proposition 4, the point x_i is in C with probability at least $2^{-O(n)}$. Hence by choosing the constant c large enough we can ensure that with high probability there is a subset $Z \subseteq P$ such that $|Z| \geq 2^{c_1(n + k \log(1/\epsilon))}$ for a constant c_1 and for all $(x_i, z_i) \in Z$, $x_i \in C$. We now prove the main theorem of this section.

THEOREM 5. *Let $\mathcal{L} \subset \mathbb{Q}^n$ be a full rank lattice and let $v \in \mathcal{L} \setminus M$ such that $2 \leq f(v) \leq 3$ for a given gauge function f and $f(v) \leq f(x)$ for all $x \in \mathcal{L} \setminus M$. Let $\epsilon > 0$ be an arbitrary constant. Then there is a randomized algorithm that in time $2^{O(n+klog(1/\epsilon))}.\text{poly}(\text{size}(\mathcal{L}))$ computes a set P of pairs (x_i, z_i) such that $|P| \geq 2^{c' \cdot (n+k\log(1/\epsilon))}$ for a constant c' and $f(x_i - z_i) \leq 8$ for all $(x_i, z_i) \in P$. Moreover, $z_i - x_i \in \mathcal{L}$ are such that with probability $1 - 2^{-O(n)}$ there is a pair of points $(x_i, z_i), (x_j, z_j) \in P$ such that $v + u = (x_i - z_i) - (x_j - z_j)$ for a vector $u \in \mathcal{L} \cap M$ with $f(u) \leq \epsilon$.*

Proof.

Consider the set P of pairs (x_i, z_i), obtained after the AKS sieving as described above, such that $|P| \geq 2^{c'(n+k\log(1/\epsilon))}$, and $f(x_i - z_i) \leq 8$ for all $(x_i, z_i) \in P$. We know that by choosing c large enough we can ensure that with high probability there is $Z \subseteq P$ such that $|Z| \geq 2^{c_1(n+k\log(1/\epsilon))}$ for any constant c_1 and for all $(x_i, z_i) \in Z$, $x_i \in C$.

Note that $\mathcal{L} \cap M$ is a rank k sublattice of $\mathcal{L}$. We will now analyze Z using the cosets of the sublattice $\mathcal{L} \cap M$.

Write Z as a partition $Z = \bigcup_{j=1}^{m} Z_j$, where for each Z_j there is a distinct coset $(\mathcal{L} \cap M) + v_j$ of $\mathcal{L} \cap M$ in $\mathcal{L}$ such that $z_i - x_i \in (\mathcal{L} \cap M) + v_j$ for all $(x_i, z_i) \in Z_j$. Let $Z'_j = \{z_i - x_i \mid (x_i, z_i) \in Z_j\}$. Suppose $u_j \in Z'_j \subseteq (\mathcal{L} \cap M) + v_j$ for $j = 1$ to m.

CLAIM 6. *[Coset sampling] By choosing constant c_1 large enough we can ensure that there is an index t, $1 \leq t \leq m$ such that $|Z_t| \geq 2^{c_2(n+k\log(1/\epsilon))}$ for any constant c_2.*

Proof of Claim Note that u_i and u_j for $i \neq j$ lie in different cosets of $\mathcal{L} \cap M$. So $u_i - u_j \notin M$. Since v is a shortest f-vector in $\mathcal{L} \setminus M$ with $2 \leq f(v) \leq 3$, we have $f(u_i - u_j) \geq 2$. Hence unit radius f-balls around u_i's are disjoint. Note that $B_f(u_i, 1) \subset B_f(0, 9)$ for $i = 1$ to m. Since $\text{vol}(B_f(0,9))/\text{vol}(B_f(0,1)) \leq 2^{dn}$ for some constant d, we have $m \leq 2^{dn}$. We have $|Z| \geq 2^{c_1(n+k\log(1/\epsilon))}$ and Z is partitioned as $Z = \bigcup_{j=1}^{m} Z_j$. So it is clear that by choosing c_1 large enough we can ensure that there is an index t, $1 \leq t \leq m$ such that $|Z_t| \geq 2^{c_2(n+k\log(1/\epsilon))}$ for any constant c_2. ∎

By renumbering the indices assume that $Z_t = \{(x_1, z_1), \ldots, (x_q, z_q)\}$, $q \geq 2^{c_2(n+k\log(1/\epsilon))}$. Let $\beta_i = z_i - x_i$ for $(x_i, z_i) \in Z_t$. Thus, each such β_i lies in the same coset $(\mathcal{L} \cap M) + v_\ell$.

CLAIM 7. *[Packing argument] By choosing the constant c_2 large enough we can ensure that there exists $(x_i, z_i), (x_j, z_j) \in Z_t, i \neq j$ such that $f(\beta_i - \beta_j) \leq \epsilon$.*

Proof of Claim Suppose for all $(x_i, z_i), (x_j, z_j) \in Z_t, i \neq j$ $f(\beta_i - \beta_j) \geq \epsilon$. We also have $f(\beta_i - \beta_j) \leq 16$ for $i, j \in [q]$. Let $\gamma_i = \beta_i - v_\ell \in \mathcal{L} \cap M \subset M$ for $i = 1$ to q. It is clear that $f(\gamma_i - \gamma_j) = f(\beta_i - \beta_j)$ for $i, j \in [q]$. Let $\{b_1, \ldots, b_k\}$ be an orthonormal basis of M. Consider the linear transformation $T : \text{M} \rightarrow \mathbb{R}^k$ such that $T(b_i) = e_i$ for $i = 1$ to k, where $\{e_1, e_2, \ldots, e_k\}$ is a standard basis of $\mathbb{R}^k$. Let $\delta_i = T(\gamma_i)$ for $i = 1$ to q. By standard linear algebra it follows that T preserves distances between points with respect to any norm. In particular, we have $f(\gamma_i - \gamma_j) = f(\delta_i - \delta_j)$ for $i, j \in [q]$. So we have $\epsilon/2 \leq f(\delta_i - \delta_j) \leq 16$. As $\delta_i \in \mathbb{R}^k$ for $i \in [q]$, it follows that *k-dimensional* balls of radius $\epsilon/2$ around δ_i's are mutually disjoint. By a packing argument it follows that $|Z_t| \leq \frac{(16+\epsilon/2)^k}{(\epsilon/2)^k} = 2^{f(k\log(1/\epsilon))}$ for a constant f. This is a contradiction since choosing c_2 large enough we can ensure that $|Z_t| \geq 2^{c_2(n+k\log(1/\epsilon))} > 2^{f(k\log(1/\epsilon))}$.

We now complete the proof with a standard argument from [AKS01, Re04] using a modified distribution.

We have $(x_i, z_i), (x_j, z_j) \in Z_t \subset Z, i \neq j, x_i, x_j \in C$ such that $f(\beta_i - \beta_j) \leq \epsilon$ and $\beta_i - \beta_j \in \mathcal{L} \cap M$. Now, we apply the argument as explained in Regev's notes [Re04] to reason with a modified distribution of the x_i. Note that in the sieving procedure described before Theorem 5, each x_i is picked independently and uniformly at random from $B_f(0,2)$. Now, notice that we can replace the original distribution of x_i with a modified distribution in which we output x_i if it lies in $B_f(0,2) \setminus (C \cup C')$ and if $x_i \in C$ it outputs either x_i or $x_i + v$ with probability $1/2$ each. Similarly, if $x_i \in C' = C + v$ it outputs either x_i or $x_i - v$ with probability $1/2$ each. By Proposition 4 it follows that this modified distribution is also uniform on $B_f(0,2)$ (indeed, this distribution is required only for the purpose of analysis). Furthermore, we can replace each x_i by the modified distribution just before it is used in the algorithm for the first time. The reason we can do this is because the distribution of y_i's remains same even if we replace x_i by the modified distribution because $y_i = x_i (\mathrm{mod} \mathcal{L})$ and $v \in \mathcal{L}$. This is explained further in Regev's notes [Re04]. Now recall that we have $(x_i, z_i), (x_j, z_j) \in Z$ with $x_i, x_j \in C$ and $f(\beta_i - \beta_j) \leq \epsilon$. Putting it together with the above argument, it follows that with good probability the points (x_i, z_i) and $(x_j + v, z_j)$ are in the set P, where P is the set of pairs left after the sieving. This is easily seen to imply that with high probability we are likely to see the vector $v + (\beta_i - \beta_j)$ as the difference of $z_i - x_i$ and $z_j - x_j$ for some two pairs $(x_i, z_i), (x_j, z_j) \in P$. The theorem now follows since $f(\beta_i - \beta_j) \leq \epsilon$. ■

By choosing M as the 0-dimensional subspace we get a $2^{O(n)}$ algorithm for SVP with respect to any nice gauge function. As an immediate consequence of Theorem 5 we get a $1 + \epsilon$ approximation algorithm for SAP problem that runs in time $2^{O(n + k \log \frac{1}{\epsilon})} \cdot poly(size(\mathcal{L}, M))$.

Remarks: The $1 + \epsilon$ approximation algorithm in [BN07] for SAP has running time $2^{O(n \log \frac{1}{\epsilon})} \cdot poly(size(\mathcal{L}, M)))$. Our algorithm has running time $2^{O(n + k \log \frac{1}{\epsilon})}$ for computing $1 + \epsilon$ approximate solution. Put another way, for $k = o(n)$ we get a $2^{O(n)}$ time algorithm for obtaining $1 + 2^{-n/k}$ approximate solutions to SAP.

There is a crucial difference in our analysis of the AKS sieving and that given in [BN07]. In [BN07] it is shown that with probability $1 - 2^{-O(n)}$ the sieving procedure outputs a $1 + \epsilon$ approximate solution $u \in \mathcal{L} \setminus M$.

On the other hand, we show in Claim 6 that with probability $1 - 2^{-O(n)}$ the sieving procedure samples $2^{O(n + k \log(1/\epsilon)}$ lattice points in *some* coset of the sublattice $\mathcal{L} \cap M$ in $\mathcal{L}$. Then we argue that with probability $1 - 2^{-O(n)}$ the sample contains a lattice point u in $\mathcal{L} \cap M + v$ such that such that $d(u, v)$ is small, for some shortest vector v in $\mathcal{L} \setminus M$. We argue this in Claim 7 by a packing argument in the coset of $\mathcal{L} \cap M$. As $\mathcal{L} \cap M$ has rank k, the packing argument in k dimensions gives the improved running time for our approximation algorithm for the problem.

The fact that the AKS sampling contains many points from the same coset of $\mathcal{L} \cap M$ also plays crucial role in our exact algorithm for SAP shown in Theorem 12.

COROLLARY 8. *Given a rank n lattice $\mathcal{L}$ and a k-dimensional subspace $M \subset \mathbb{R}^n$, there is $1+\epsilon$ randomized approximation algorithm for SAP (for any nice gauge function) with running time $2^{O(n+k\log\frac{1}{\epsilon})} \cdot poly(size(\mathcal{L}, M))$.*

Proof. The algorithm will examine all $(z_i - x_i) - (z_j - x_j)$ for $(x_i, z_i), (x_j, z_j) \in P$ obtained after sieving and output that element in $\mathcal{L} \setminus M$ of minimum f-value. The proof of correctness and running time guarantee follows immediately from Theorem 5. ∎

4 Convex Body Avoiding Problem

In this section we consider a generalization of SAP: given a lattice $\mathcal{L}$ and a convex body C the problem is to find a shortest vector (w.r.t. ℓ_p norm) in $\mathcal{L} \setminus C$. We consider convex bodies C that are bounded and O-symmetric. We refer to this problem as the *Convex body Avoiding Problem* (CAP).

A set $S \subseteq \mathbb{R}^n$ is *O-symmetric* if $x \in S$ if and only if $-x \in S$. Notice that a subspace $M \subseteq \mathbb{R}^n$ is convex and O-symmetric (but not bounded).

The input to CAP is the lattice $\mathcal{L}$ and the convex body C, where C is given by a membership oracle. An algorithm can query the oracle for any $x \in \mathbb{R}^n$ to test if $x \in C$.

We give an approximation algorithm to solve CAP.

THEOREM 9. *Given an integer lattice $\mathcal{L}$ of rank n and an O-symmetric convex body C in $\mathbb{R}^n$ given by a membership oracle, there is $1+\epsilon$ factor approximation algorithm to solve CAP (w.r.t. any ℓ_p norm) with running time $2^{O(n)\cdot\log(1/\epsilon)} \cdot poly(size(\mathcal{L}))$.*

Proof. It suffices to solve the problem for the case when C is n-dimensional. To see this, suppose C is contained in some k-dimensional subspace M of $\mathbb{R}^n$. We can find a basis for M with high probability by sampling vectors from C using the polynomial-time almost uniform sampling algorithm described in [DFK91]. Next, we compute the sublattice $\mathcal{L} \cap M$ and find a $(1+\epsilon)$ approximate solution u for the k-dimensional convex body avoidance for the lattice $\mathcal{L} \cap M$ and C. We also solve the SAP instance $(\mathcal{L}, M)$ and find a $(1+\epsilon)$ approximate solution $v \in \mathcal{L} \setminus M$ using Theorem 5. The shorter of vectors u and v is clearly a $(1+\epsilon)$ approximate solution for the input CAP instance.

Thus, we can assume C is n-dimensional. Let v be a shortest vector in $\mathcal{L} \setminus C$ which, as before, we can assume satisfies $2 \leq \|v\|_p \leq 3$ by considering polynomially many scalings of the lattice and the convex body. As in Theorem 5, we pick random points $x_1, \cdots, x_N$ from $B_p(0,2)$ for $N = 2^{cn\log(1/\epsilon)} \cdot poly(s)$. The constant $c > 0$ will be suitably chosen later. Let $y_i = x_i (\text{mod } \mathcal{L})$ for $i = 1$ to N. We apply several rounds of the AKS sieving on the set $\{(x_1, y_1), \cdots, (x_N, y_N)\}$ until we are left with a set S of $2^{c_1 n\log(1/\epsilon)}$ pairs (x_i, z_i) such that $\|x_i - z_i\|_p \leq 8$. From proposition 4 it follows easily that with good probability we have $Z \subseteq S$ such that $|Z| \geq 2^{c_2 n\log(1/\epsilon)}$ and for all $(x_i, z_i) \in Z$ we have $x_i \in D \cup D'$ where $D = B_p(0,2) \cap B_p(-v,2)$ and $D' = B_p(0,2) \cap B_p(v,2)$. Note that the the constant c_2 can be chosen as large as we like by appropriate choice of c. Let $Z' = \{z_i - x_i \mid (x_i, z_i) \in Z\}$. Now consider ℓ_p ball of radius $\epsilon/2$ centered at each lattice point $\beta \in Z'$. It is clear that for all $\beta \in Z'$, $B_p(\beta, \epsilon/2) \subseteq B_p(0, 8+\epsilon/2)$. If for all $\beta \in Z'$ ℓ_p balls $B_p(\beta, \epsilon/2)$ are mutually disjoint, by packing argument we get $|Z'| \leq \frac{(8+\epsilon/2)^n}{(\epsilon/2)^n} = 2^{c' n\log(1/\epsilon)}$ for a constant c'. We choose constant

c appropriately to ensure that $c_2 > c'$. This implies that there exists tuples $(x_i, z_i), (x_j, z_j) \in Z$ such that $\|\beta_i - \beta_j\| \leq \epsilon$, where $\beta_i = z_i - x_i$ and $\beta_j = z_j - x_j$. Let $\beta = \beta_i - \beta_j$. We claim that it is not possible that both $\beta + v, \beta - v$ lie inside the convex body C. Because this implies $v - \beta \in C$ since C is O-symmetric. Therefore $v = \frac{(\beta+v)+(v-\beta)}{2} \in C$, which contradicts with assumption $v \notin C$. So without loss of generality assume that $\beta + v \notin C$. Note that without loss of generality we can also assume that $x_i \in D'$ with good probability. Now, we apply the argument as explained in [Re04] to reason with a modified distribution of the x_i. As $x_i \in D'$ we can replace x_i by $x_i - v$. It is easy to see that after sieving with good probability there exists tuples $(x_i, z_i), (x_j, z_j) \in S$ such that $r_{i,j} = (z_i - x_i) - (z_j - x_j) = v + \beta_i - \beta_j$. Hence, $r_{i,j} = v + \beta \notin C$ and, clearly, $\|r_{i,j}\|_p \leq (1 + \epsilon)\|v\|_p$ since $\|\beta_i - \beta_j\|_p \leq \epsilon$. It is easy to see that the algorithm runs in time $2^{O(n \log(1/\epsilon))} poly(size(\mathcal{L}))$. This completes the proof of the theorem. ∎

5 Applications

The results of this section are essentially applications of ideas from Theorem 5 and Section 3.

First we describe an exact algorithm for SAP for ℓ_p norms. We prove our result for full rank lattices, but it is easy to see that the result holds for general lattices as well. Let $\mathcal{L} \subset \mathbb{Q}^n$ be a full rank integer lattice given by a basis $\{b_1, \cdots, b_n\}$ and let $M \subseteq \mathbb{R}^n$ is a subspace of dimension $k < n$. For any ℓ_p norm we give a randomized $2^{O(n+k \log k)} poly(s)$ time algorithm to find a shortest vector in $\mathcal{L} \setminus M$, where $s = size(\mathcal{L}, M)$. Our exact algorithm uses the same sieving procedure and analysis described in the proof of Theorem 5 in Section 3. As before, by considering polynomially many scalings of the lattice, we can assume that a shortest vector $v \in \mathcal{L} \setminus M$ satisfies $2 \leq \|v\|_p \leq 3$. We now describe the algorithm.

1. Let $N = 2^{cn} \log(n.max_i \|b_i\|_p)$. Pick $x_1, x_2, \cdots, x_N$ uniformly at random from $B_p(0, 2)$.
2. Let $y_i = x_i (\text{mod } \mathcal{L})$. Apply AKS sieving to the set $\{(x_1, y_1), \cdots, (x_N, y_N)\}$ as described in Section 3 until $\|x_i - z_i\|_p \leq 8$ for each pair (x_i, z_i) left after the sieving.
3. Let $P = \{(x_i, z_i) | i \in T\}, T \subset [N]$ be the set of tuples left after the sieving procedure. For all $i, j \in T$ compute lattice points $v_{i,j} = (z_i - x_i) - (z_j - x_j)$.
4. Let $w_{i,j}$ be a closest lattice vector to $v_{i,j}$ in the rank k lattice $\mathcal{L} \cap M$ (found using Kannan's exact CVP algorithm [Kan87]), and let $r_{i,j} = v_{i,j} - w_{i,j}$. Output a vector of least nonzero ℓ_p norm among all the vectors $r_{i,j}$ for $i, j \in T$.

First we prove the correctness of the algorithm.

LEMMA 10. *For an appropriate choice of the constant c in the algorithm, it outputs a shortest nonzero vector in $\mathcal{L} \setminus M$ with respect to ℓ_p norm.*

Proof. Let v be a shortest vector in $\mathcal{L} \setminus M$. Consider the set of pairs $P = \{(x_i, z_i) | i \in T\}, T \subset [N]$, that remains after the sieving procedure in Step 3 of the algorithm. If we choose ϵ as a constant in Theorem 5, it follows that there is a constant c such that with probability $1 - 2^{-O(n)}$ there exists $(x_i, z_i), (x_j, z_j) \in P$ such that $v + u = \beta_i - \beta_j$ for some $u \in \mathcal{L} \cap M$ where $\beta_i = z_i - x_i$ and $\beta_j = z_j - x_j$. Hence, in Step 3 of the algorithm we have some $v_{i,j} = v + u$ for some vector $u \in \mathcal{L} \cap M$, i.e. $v_{i,j}$ and v lie in same coset of $\mathcal{L} \cap M$.

Let $w_{i,j} \in \mathcal{L} \cap M$ be a closest vector to $v_{i,j}$. So we have $d(v_{i,j}, w_{i,j}) \leq d(v_{i,j}, u) = \|v\|_p$, i.e. $\|v_{i,j} - w_{i,j}\|_p \leq \|v\|_p$. But since we have $v_{i,j} \notin \mathcal{L} \cap M$ and $w_{i,j} \in \mathcal{L} \cap M$ clearly $v_{i,j} - w_{i,j} \notin$

$\mathcal{L} \cap M$ and since v is a shortest vector in $\mathcal{L} \setminus M$, this implies $\|v_{i,j} - w_{i,j}\|_p = \|v\|_p$. So with probability $1 - 2^{-O(n)}$ the algorithm will output (in Step 4) a vector $r_{i,j}$ with $\|r_{i,j}\|_p = \|v\|_p$. This proves the correctness of the algorithm. ■

Next we argue that the running time of the algorithm is $2^{O(n+k\log k)} \cdot poly(s)$ where s is the input size. In Step 1 of the algorithm we are sampling $N = 2^{O(n)}$ points from $B_p(0,2)$, a ball of radius 2 with respect to l_p norm. Since $B_p(0,2)$ is a convex body, the task can be accomplished using Dyer-Frieze-Kannan algorithm [DFK91] in time $2^{O(n)} \cdot poly(s)$. It easily follows that the sieving procedure in Step 2 can be performed in $2^{O(n)}$ time. Note that $\mathcal{L} \cap M$ is a rank k lattice and a basis for it can be computed efficiently. We need the following easy lemma from [Mi08].

LEMMA 11.[Mi08, Lemma 1] *There is a polynomial-time algorithm that takes as input a lattice $\mathcal{L} \subset \mathbb{Q}^n$ and a subspace $M \subset \mathbb{R}^n$ of dimension $k < n$ outputs a basis for rank k lattice $\mathcal{L} \cap M$.*

From the above lemma it is clear that a basis for $\mathcal{L} \cap M$ can be efficiently computed in polynomial time. In Step 4 of the algorithm we are solving $2^{O(n)}$ many instances of CVP for the rank k lattice $\mathcal{L} \cap M$. For $i, j \in S$ a closest vector to $v_{i,j}$ in the rank k lattice $\mathcal{L} \cap M$ can be computed in $2^{O(k\log k)}$ time using Kannan's algorithm for CVP [Kan87]. Hence the Step 4 takes $2^{O(n+k\log k)}$ time. Therefore the overall running time of the algorithm is $2^{O(n+k\log k)} \cdot poly(s)$. Note that by repeating above algorithm $2^{O(n)}$ times we can make the success probability of the algorithm exponentially close to 1.

THEOREM 12. *Given a full rank lattice $\mathcal{L} \subset \mathbb{Q}^n$ and a subspace $M \subseteq \mathbb{R}^n$ of dimension $k < n$, There is a randomized algorithm to finds $v \in \mathcal{L} \setminus M$ with least possible l_p norm. The running time of the algorithm is $2^{O(n+k\log k)}$ times a polynomial in the input size and it succeeds with probability $1 - 2^{-cn}$ for an arbitrary constant c.*

Blömer and Naewe [BN07] gave $2^{O(n)}$ time $1+\epsilon$ factor approximation algorithms to solve the SMP and SIVP problems. As a simple consequence of Theorem 12 we get a $2^{O(n)}$ time randomized algorithm to "partially" solve SMP: we can compute the first $O(\frac{n}{\log n})$ successive minima in $2^{O(n)}$ time. More precisely, we can compute a set of i linearly independent vectors $\{v_1, v_2, \ldots, v_i\} \subset \mathcal{L}$ such that $\|v_j\|_p = \lambda_j^p(\mathcal{L})$ for $j = 1$ to i if i is $O(\frac{n}{\log n})$.

Given a lattice $\mathcal{L}$, let $M = 0 \subset \mathbb{R}^n$ be the zero-dimensional subspace in $\mathbb{R}^n$ and consider the SAP instance $(\mathcal{L}, M)$. Clearly, v_1 is a shortest vector in $\mathcal{L} \setminus M$. Hence, by Theorem 12 we can compute v_1 in $2^{O(n)}$ time. Now, inductively assume that we have computed linearly independent vectors $v_1, v_2, \ldots, v_k \in \mathcal{L}$ such that $\|v_j\|_p = \lambda_j^p(\mathcal{L})$. Consider the instance $(\mathcal{L}, M)$ of SAP where M is the space generated by $v_1, \ldots, v_k$ and compute $v \in \mathcal{L} \setminus M$ using Theorem 12 in time $2^{O(n+k\log k)}$. It is clear that $\|v\|_p = \lambda_{k+1}^p(\mathcal{L})$ and as $v \notin M$ the vectors $v_1, v_2, \ldots, v_k, v$ are linearly independent. If k is $O(\frac{n}{\log n})$ it is clear that algorithm takes $2^{O(n)}$ time. This proves Corollary 13.

COROLLARY 13. *Given a full rank lattice $\mathcal{L} \subset \mathbb{Q}^n$ and a positive integer $i \leq \frac{cn}{\log n}$ for a constant c, there is a randomized algorithm with running time $2^{O(n)} \cdot poly(size(\mathcal{L}))$ to*

compute linearly independent vectors $v_1, v_2, \ldots, v_i \in \mathcal{L}$ *such that* $\|v_j\|_p = \lambda_j^p(\mathcal{L})$ *for* $j = 1$ *to* i.

The CVP problem is polynomial-time reducible to SAP, as noted in [BN07]. Micciancio [Mi08] has shown that CVP, SAP and SMP are all polynomial-time equivalent. Our algorithm computes $v \in \mathcal{L} \setminus M$ with least norm by solving $2^{O(n)}$ instances of CVP. We have basically given a randomized $2^{O(n)}$ time Turing reduction from SAP to CVP. An interesting property of our reduction is that we are solving instance $(\mathcal{L}, M)$ of SAP by solving $2^{O(n)}$ many CVP instances $(\mathcal{L} \cap M, v)$ where $\mathcal{L} \cap M$ is a rank k lattice, where k is dimension of M. In contrast, for the CVP instance (N, v) produced by the SAP to CVP reduction in [BN07] the lattice N has rank $O(n)$.

As a consequence of this property of our reduction we obtain Corollary 14 which states that it suffices to look for a $2^{O(n)}$ randomized exact algorithm for CVP that can access all successive minimas of the input lattice.

COROLLARY 14. *Suppose for all* m *there is a* $2^{O(m)}$ *randomized exact algorithm for* CVP *that takes as input a* CVP *instance* (M, v) *where* M *is full rank lattice of rank* m *and* $v \in \mathbb{R}^m$ *(along with the* extra *input* $v_i \in M$ *such that* $|v_i|_p = \lambda_i^p(M)$ *for* $i = 1$ *to* m *where* $\lambda_i^p(M)$ *is* i^{th} *successive minima in* M*). Then, in fact, there is a* $2^{O(n)}$ *randomized exact algorithm for solving* CVP *on any rank* n *lattice.*

Proof. By [Mi08], CVP is polynomial-time equivalent to SMP (the successive minima problem). Consider the full rank lattice $\mathcal{L} \subset \mathbb{Q}^n$ as input to SMP. It suffices to compute linearly independent vectors $v_1, \ldots, v_n \in \mathcal{L}$ with $\|v_i\|_p = \lambda_i^p(\mathcal{L})$ for $i = 1$ to n in $2^{O(n)}$ time. We proceed as in the proof of Corollary 13. Inductively assume that we have computed linearly independent vectors $v_1, \ldots, v_k \in \mathcal{L}$ with $\|v_i\|_p = \lambda_i^p(\mathcal{L})$. Let M be the space generated by $v_1, \ldots, v_k$. As in proof of Theorem 12 we can solve the SAP instance $(\mathcal{L}, M)$ by solving $2^{O(n)}$ many instances of CVP $(\mathcal{L} \cap M, v')$. Note that $\mathcal{L} \cap M$ is rank k lattice and it is clear that $\|v_i\|_p \lambda_i^p(\mathcal{L} \cap M)$ for $i = 1$ to k. Hence we can solve these instances in $2^{O(n)}$ time (although $\mathcal{L} \cap M$ is not full rank lattice, but it is not difficult to convert all these instances of CVP to full rank by applying a suitable linear transformation). This takes time $2^{O(n+k)}$ which is at most $2^{O(n)}$. Hence, it is clear that we can compute linearly independent vectors $v_1, \ldots, v_n \in \mathcal{L}$ such that $\|v_i\|_p = \lambda_i^p(\mathcal{L})$ in time $n \cdot 2^{O(n)}$. ∎

In the next corollary we give a $2^{O(n)}$ time algorithm to solve certain CVP instances $(\mathcal{L}, v)$ for any ℓ_p norm. We prove the result only for ℓ_2 norm and it is easy to generalize it for general ℓ_p norms. Let $\lambda_i(\mathcal{L})$ denote i th successive minima of the lattice $\mathcal{L}$ with respect to ℓ_2 norm.

COROLLARY 15. *Let* $(\mathcal{L}, v)$ *be a* CVP *instance such that* $\mathcal{L}$ *is full rank with the promise that* $d(v, \mathcal{L}) < \sqrt{3}/2\lambda_t(\mathcal{L})$, $t \leq \frac{cn}{\log n}$. *Then there is a* $2^{O(n)} \cdot poly(size(\mathcal{L}))$ *time randomized algorithm that solves such a* CVP *instance exactly.*

Proof. By Corollary 13 we first compute $\lambda_t(\mathcal{L})$. We now use ideas from Kannan's CVP to SVP reduction [Kan87]. Let $b_1, b_2, \cdots, b_n$ be a basis for $\mathcal{L}$. We obtain new vectors $c_i \in \mathbb{Q}^{n+1}$ for $i = 1$ to n by letting $c_i^T = (b_i^T, 0)$. Likewise, define $u \in \mathbb{Q}^{n+1}$ as $u^T = (v^T, \lambda_t/2)$. Let $\mathcal{M}$ be the lattice generated by the $n + 1$ vectors $u, c_1, c_2, \cdots c_n$. Compute the vectors $v_j \in \mathcal{M}$

such that $\|v_j\|_2 = \lambda_j(\mathcal{M})$ for $j = 1$ to t using Corollary 13 in time $2^{O(n)} \cdot poly(size(\mathcal{L}))$. Write vectors v_j as $v_j = u_j + \alpha_j u$, $u_j \in \mathcal{L}(c_1, \cdots, c_n)$ and $\alpha_j \in \mathbb{Z}$. Clearly, $|\alpha_j| \leq 1$ since u has $\lambda_t/2$ as its $(n+1)^{th}$ entry. As $d(v, \mathcal{L}) < \sqrt{3}/2\lambda_t(\mathcal{L})$ we have $d(u, \mathcal{M}) < \lambda_t(\mathcal{L})$. Hence, there is at least one index i, $1 \leq i \leq t$ such that $|\alpha_i| = 1$. Consider the set $S = \{u_i \mid 1 \leq i \leq t, |\alpha_i| = 1\}$and let u_j be the shortest vector in S. Writing $u_j = (w_j^T, 0)$, it is clear that the vector $-w_j \in \mathcal{L}$ is closest vector to v if $\alpha_j = 1$ and w_j is a closest vector to v if $\alpha_j = -1$. ■

References

[AKS01] M. AJTAI, R. KUMAR, D. SIVAKUMAR, A sieve algorithm for the shortest lattice vector. *In Proceedings of the 30th Annual ACM Symposium on Theory of Computing,* 266-275, 2001.

[AKS02] M. AJTAI, R. KUMAR, D. SIVAKUMAR, Sampling short lattice vectors and the closest lattice vector problem. *In Proceedings of the 17th IEEE Annual Conference on Computational Complexity-CCC,* 53-57, 2002.

[Bl00] J. BLÖMER, Closest vectors, successive minima, and dual HKZ-bases of lattices. *In Proceedings of th 17th ICALP,* Lecture Notes in Computer Science 1853, 248-259, Springer, 2000.

[BN07] J. BLÖMER, S. NAEWE Sampling Methods for Shortest Vectors, Closest Vectors and Successive Minima of lattices. *In Proceedings of ICALP,* 65-77, 2007.

[DFK91] M. DYER, A. FRIEZE, R. KANNAN A random polynomial time algorithm for approximating the volume of convex bodies. *Journal of the ACM ,* 38(1):1-17, 1991.

[Kan87] R. KANNAN Minkowski's convex body theorem and integer programing. *Mathematics of Operational Rearch ,*12(3):415-440, 1987.

[LLL82] A. K. LENSTRA, H. W. LENSTRA, JR. AND L. LOVASZ, Factoring Polynomials with Rational Coefficients, *Mathematische Annalen,* 261:515-534, 1982.

[MG02] D. MICCIANCIO, S. GOLDWASSER, *Complexity of Lattice Problems. A Cryptographic Perspective,* Kluwer Academic Publishers, 2002.

[Mi08] D. MICCIANCIO, Efficient reductions among lattice problems,*SODA*,2008,84-93

[Re04] O. REGEV, Lecture Notes — Lattices in Computer Science, lecture 8. Available at the website: http://www.cs.tau.ac.il/ odedr/teaching/lattices_fall_2004/index.html.

[Si45] C. L. SIEGEL Lectures on Geometry of Numbers. *Springer-Verlag publishing company*, 1988.

Foundations of Software Technology and Theoretical Computer Science (Bangalore) 2008.
Editors: R. Hariharan, M. Mukund, V. Vinay; pp 37-48

Analyzing Asynchronous Programs with Preemption

Mohamed Faouzi Atig, Ahmed Bouajjani, Tayssir Touili
LIAFA, CNRS and University Paris Diderot
France
{atig,abou,touili}@liafa.jussieu.fr

ABSTRACT. Multiset pushdown systems have been introduced by Sen and Viswanathan as an adequate model for asynchronous programs where some procedure calls can be stored as tasks to be processed later. The model is a pushdown system supplied with a multiset of pending tasks. Tasks may be added to the multiset at each transition, whereas a task is taken from the multiset only when the stack is empty. In this paper, we consider an extension of these models where tasks may be of different priority level, and can be preempted at any point of their execution by tasks of higher priority. We investigate the control point reachability problem for these models. Our main result is that this problem is decidable by reduction to the reachability problem for a decidable class of Petri nets with inhibitor arcs. We also identify two subclasses of these models for which the control point reachability problem is reducible respectively to the reachability problem and to the coverability problem for Petri nets (without inhibitor arcs).

1 Introduction

In the last few year, a lot of effort has been devoted to the verification problem for models of concurrent programs (see, e.g., [17, 7, 15, 19, 4, 3, 2, 23, 13, 1]). Multiset Pushdown Systems (MPDS) have been introduced in [22] as an adequate model for asynchronous programs. These programs constitute an important class of program widely used in the management of concurrent interactions with the environment, e.g., in building networked software systems, distributed systems, etc. In these programs, procedure calls can be either synchronous, which means that the caller waits as usual until the callee returns, or asynchronous, which means that the callee is rather stored as a task to be processed later. Repetitively, pending tasks are chosen and executed until completion, which may generate other pending tasks.

The MPDS model consists of a pushdown system supplied with a multiset store containing pending tasks. When (and only when) the stack is empty, a task is taken from the multiset and put into the stack. Then, the system starts executing the task using pushdown transition rules which, in addition to usual push and pop operations (modeling synchronous procedure calls) can generate new tasks (modeling asynchronous procedure calls). Notice that in this model, both the stack and the multiset store are of unbounded sizes. The control point reachability problem has been proved to be decidable in [22], and an efficient procedure for deciding this problem has been developed in [12].

In this paper, we consider a wider class of programs where tasks may have different priority levels (assuming that there is a finite number of such levels), and that at any point in time tasks are executed according to their priority level ordering. This means that tasks can be preempted by tasks of higher priority level: When a task γ of level i generates a

FSTTCS 2008
IARCS Annual Conference on
Foundations of Software Technology and Theoretical Computer Science
http://drops.dagstuhl.de/opus/volltexte/2008/1739

task γ' of level $j > i$, the task γ is preempted and must wait until the task γ' as well as all its descendants (i.e., tasks it created) of level greater than i are done. We consider that in general the task γ' may also have descendant of level less or equal to i; these tasks are stored among the other pending tasks of their level for later execution.

To reason about this class of programs, we introduce the model of k-MPDS corresponding to MPDS with $k+1$ priority levels and preemption (i.e., 0-MPDS coincides with the model of [22]). We address the control point/configuration reachability problem in these models. Our main result is that both of these problems are decidable. The proof is not trivial. The main difficulty to face is that a preempted task can be resumed only when there are no pending tasks of higher level. This involves a kind of test to 0 of some counters (which count the number of pending tasks at each priority level). We show that in fact these reachability problems can be reduced to the reachability problem in a class of Petri nets with inhibitor arcs shown to be decidable by Reinhardt in [21].

Then, we consider two classes of k-MPDS obtained by introducing some restrictions either on the way priority levels are assigned to newly created tasks, or on the allowed kind of communication through return values between asynchronous calls. The first subclass of models we consider, called hierarchical MPDS, corresponds to systems where each created task is assigned a priority level which is at least as high as the one of its caller. We show that this inheritance-based policy of priority assignment leads to a model for which both the control point and the configuration reachability problem can be reduced to the reachability problem in Petri nets without inhibitor arcs.

The second subclass we consider, called restricted MPDS, corresponds to systems where return values are taken into account (1) for synchronous calls at any level, (2) for asynchronous calls at level 0, and (3) between tasks of different levels when a preemption or a resumption occurs. This means that returns values by asynchronous calls within levels greater than 0 are not taken into account (i.e., they are abstracted away), but these calls may have an influential side effect by creating new tasks at any level, and this is taken into account in our model. We prove that for the corresponding models to this class of programs the control point and the configuration reachability problems are reducible to the corresponding problems in Petri nets using Parikh image computations of context-free languages. This means in particular that the control point problem (which the relevant problem for proving safety properties) for these models can be reduced to the coverability problem in Petri nets. As far as we know, our results are not covered by any existing result in the literature.

2 Preliminaries

Words and Languages. Let Σ be a finite alphabet. We denote by Σ^* (resp. Σ^+) the set of all *words* (resp. non empty words) over Σ, and by ϵ the empty word. A language is a (possibly infinite) set of words. Given two disjoint finite alphabets Σ and Σ' and a language L over $\Sigma \cup \Sigma'$, the projection of L on Σ, denoted L_Σ, is the set of words $a_1 \dots a_n \in \Sigma^*$ such that $\left(\Sigma'^* a_1 \Sigma'^* a_2 \Sigma'^* \cdots \Sigma'^* a_n \Sigma'^*\right) \cap L \neq \emptyset$.

Finite State Automata. A Finite State Automaton (FSA) is a tuple $\mathcal{S} = (S, \Sigma, \delta, s^i, s^f)$ where S is a finite set of states, Σ is a finite alphabet, $\delta \subseteq S \times (\Sigma \cup \{\epsilon\}) \times S$ is a set of rules, $s^i \in S$ is an initial state, and $s^f \in S$ is a final state. Let $L(\mathcal{S})$ denotes the language accepted by $\mathcal{S}$.

Multi-sets. Let Σ be a finite alphabet. A Multi-set over Σ is a function $M : \Sigma \to \mathbb{N}$. We denote by $M[\Sigma]$ the collection of all multi-sets over Σ and by $\emptyset$ the empty multi-set. Given two multi-sets M and M', we write $M' \leq M$ iff $M'(a) \leq M(a)$ for every $a \in \Sigma$; and $M + M'$ (resp. $M - M'$ if $M' \leq M$) to denote the multi-set where $(M + M')(a) = M(a) + M'(a)$ (resp. $(M - M')(a) = M(a) - M'(a)$) for every $a \in \Sigma$. For a word $w \in \Sigma^*$, $\lfloor w \rfloor$ is the multi-set formed by counting the number of symbols occurring in w; and for a language $L \subseteq \Sigma^*$, $\lfloor L \rfloor = \{\lfloor w \rfloor : w \in L\}$. A set $\mathcal{M} \subseteq M[\Sigma]$ is semi-linear iff there is a FSA $\mathcal{S}$ s.t. $\mathcal{M} = \lfloor L(\mathcal{S}) \rfloor$.

Context-Free Grammars. A Context-Free Grammar (CFG) is a tuple $G = (V, \Sigma, R, S)$ where V is a set of non terminal symbols, Σ is an input alphabet, $S \in V$ is the start symbol (called also axiom), and $R \subseteq V \times (V \cup \Sigma)^*$ is a finite set of production rules. Given two words $u, v \in (V \cup \Sigma)^*$, we write $u \vdash_G v$ iff $\exists (\alpha, \beta) \in R$ such that $u = u_1 \alpha u_2$ and $v = u_1 \beta u_2$ for some $u_1, u_2 \in (V \cup \Sigma)^*$. We denote by $\vdash_G^*$ the transitive and reflexive closure of $\vdash_G$ and by $L(G) = \{w \in \Sigma^* \mid S \vdash_G^* w\}$ the context free language generated by G.

Labeled Pushdown Systems. A Labeled Pushdown System (LPDS) is a tuple $\mathcal{P} = (Q, \Sigma, \Gamma, \delta)$ where Q is a finite set of states, Σ is an input alphabet, Γ is a stack alphabet, and δ is a finite set of transition rules of the form: $q\gamma \overset{a}{\hookrightarrow} q'w'$ where $q, q' \in Q$, $\gamma \in \Gamma$, $a \in \Sigma \cup \{\epsilon\}$, and $w \in \Gamma^*$. A *configuration* of $\mathcal{P}$ is a tuple (q, σ, w) where $q \in Q$ is a state, $\sigma \in \Sigma^*$ is an input word, and $w \in \Gamma^*$ is a stack content. We define the binary relation $\Rightarrow_{\mathcal{P}}$ between configurations as follows: $(q, a\sigma, w\gamma) \Rightarrow_{\mathcal{P}} (q', \sigma, ww')$ iff $q\gamma \overset{a}{\hookrightarrow} q'w' \in \Delta$. The *transition relation* $\Rightarrow_{\mathcal{P}}^*$ is the reflexive transitive closure of $\Rightarrow_{\mathcal{P}}$.

Given a LPDS $\mathcal{P}$, a pair of states $q_1, q_2 \in Q$, and a stack symbol $\gamma \in \Gamma$, we define $L_{\mathcal{P}}(q_1, q_2, \gamma)$ as the set of words $\{\sigma \in \Sigma^* \mid (q_1, \sigma, \gamma) \Rightarrow_{\mathcal{P}}^* (q_2, \epsilon, \epsilon)\}$. It is well-known that $L_{\mathcal{P}}(q_1, q_2, \gamma)$ is a context-free language, and conversely, every context-free language can be defined as a trace language of some LPDS.

Finally, we recall a result due to Parikh [18] which will be used later in the paper.

PROPOSITION 1. *Given a LPDS $\mathcal{P} = (Q, \Sigma, \Gamma, \delta)$, two states $q_1, q_2 \in Q$, and a stack symbol $\gamma \in \Gamma$, it is possible to construct a FSA $\mathcal{S}$ such that $\lfloor L_{\mathcal{P}}(q_1, q_2, \gamma) \rfloor = \lfloor L(\mathcal{S}) \rfloor$.*

Petri Nets with Inhibitor arcs. A Petri net with inhibitor arcs is a pair $\mathcal{N} = (P, T)$ where P is a finite set of places, and $T \subseteq 2^P \times P^* \times P^*$ is a finite set of transitions. Given a transition $t = (I, w, w')$, we define the relation $\xrightarrow{t} \subseteq M[P] \times M[P]$ as follows: $W \xrightarrow{t} W'$ iff $W \geq \lfloor w \rfloor$, $W' = W + \lfloor w' \rfloor - \lfloor w \rfloor$ and $W(p) = 0$ for every $p \in I$. We define the transition relation $\rightarrow_{\mathcal{N}}$ on multi-sets over P by the union of the $\xrightarrow{t}$, i.e., $\rightarrow_{\mathcal{N}} = \bigcup_{t \in T} \xrightarrow{t}$. The transition relation $\rightarrow_{\mathcal{N}}^*$ is the reflexive transitive closure of $\rightarrow_{\mathcal{N}}$.

A Petri net with *weak* inhibitor arcs is a Petri-net with inhibitor arcs (P, T) such that there is a function $f : P \to \mathbb{N} \setminus \{0\}$ such that $\forall p, p' \in P$, $f(p) \leq f(p') \Rightarrow (\forall (I, w, w') \in T, p' \in I \Rightarrow p \in I)$. A Petri net can be seen as a subclass of Petri nets with inhibitor arcs where all the transitions $(I, w, w') \in T$ are such that $I = \emptyset$. In this case, the transitions T can be described in $P^* \times P^*$.

The reachability (resp. coverability) problem for a Petri net with inhibitor arcs N is the problem of deciding for two given multi-sets W' and W whether $W \rightarrow_{\mathcal{N}}^* W'$ (resp. there is a multi-set $W'' \geq W'$ such that $W \rightarrow_{\mathcal{N}}^* W''$). Reachability and coverability problems for Petri

nets with inhibitor arcs are undecidable [10]. Fortunately, they become decidable for Petri nets Petri nets with weak inhibitor arcs.

THEOREM 2. *Reachability and coverabilty problems for Petri nets with weak inhibitor arcs are decidable* [21]. *Moreover, the reachability (resp. coverability) problem for Petri nets is decidable and EXSPACE-hard (resp. EXSPACE-complete)*[14, 20, 16, 5].

3 Multi-set Pushdown Systems with Preemption

3.1 Definition of the Model

We introduce multiset pushdown systems with preemptive task generation according to a finite number of priority classes. The model of MPDS defined in [22] corresponds to the particular case where all tasks have the same priority (and therefore preemption never occurs).

DEFINITION 3. *Let k be a natural number. A k-Multi-set Pushdown System with Preemption (k-MPDS) is a tuple $\mathcal{A} = (Q, \Gamma_0, \ldots, \Gamma_k, \Delta, \Delta', q_0, \gamma_0)$ where Q is a finite set of states, $\Gamma = \bigcup_{0 \le j \le k} \Gamma_j$ is a finite set of multi-set symbols, $q_0 \in Q$ is the initial state, $\gamma_0 \in \Gamma_0$ is the initial task, and the sets $\Delta \subseteq \bigcup_{j=0}^{k} (Q \times \Gamma_j) \times (Q \times \Gamma_j^* \times (\Gamma \cup \{\epsilon\}))$ and $\Delta' \subseteq Q \times Q \times \Gamma$ form together the transition rules.*

For presentation matter, transitions in Δ (resp. Δ') will be represented respectively by $q\gamma \hookrightarrow q'w' \rhd \gamma'$ (resp. $q \hookrightarrow q' \lhd \gamma'$) with $q, q' \in Q$, $\gamma \in \Gamma_j$, $w' \in \Gamma_j^*$, and $\gamma' \in \Gamma$ for some $j \in \{0, \ldots, k\}$. Intuitively, rules of the form $q\gamma \hookrightarrow q'w' \rhd \gamma'$ correspond, in addition to the usual pushdown operations (popping γ and then pushing w' while changing the control state from q to q'), to generate the task γ'. Rules of the form $q \hookrightarrow q' \lhd \gamma$ correspond to move the control state from q to q' and to start executing the pending task γ if the priority level of the topmost symbol in the stack is strictly less than the priority level of γ.

A configuration of $\mathcal{A}$ is a tuple $(q, w, M_0, \ldots, M_k)$ where $q \in Q$, $w \in \Gamma_0^* \times \cdots \times \Gamma_k^*$, and $M_j \in M[\Gamma_j]$, $0 \le j \le k$, is a multiset representing the waiting tasks of priority j. The content of the stack w is always of the form $w_0 w_1 \ldots w_i$ where for every $j \in \{0, \ldots, i\}$, $w_j \in \Gamma_j^*$ is the tasks of priority j that are waiting in the stack. The initial configuration of $\mathcal{A}$ is $(q_0, \epsilon, \lfloor \gamma_0 \rfloor, \emptyset, \ldots, \emptyset)$. The transition relation $\Rightarrow_{\mathcal{A}}$ is defined as the union of the binary relations $\rightarrow_{0 \le j \le k}$, $\hookrightarrow_{0 \le j \le k}$, and $\rightsquigarrow_{0 \le j < k}$ defined as follows:

- **Move with task creation ($\hookrightarrow_j$):** $(q, w\gamma_j, M_0, \ldots, M_j, \emptyset, \ldots, \emptyset) \hookrightarrow_j (q', ww', M_0, \ldots, M_i + \lfloor \gamma_i \rfloor, \ldots, M_j, \emptyset, \ldots, \emptyset)$ iff $(q\gamma_j \hookrightarrow q'w' \rhd \gamma_i) \in \Delta$, $\gamma_j \in \Gamma_j$, $\gamma_i \in \Gamma_i \cup \{\epsilon\}$, and $i \le j$. Such transitions correspond to move the control state from q to q', pop γ_j from the top of the stack, push w' into the stack, and generate the task γ_i.
- **Move with task preemption ($\rightsquigarrow_j$):** $(q, w\gamma_j, M_0, \ldots, M_j, \emptyset, \ldots, \emptyset) \rightsquigarrow_j (q', ww'\gamma_i, M_0, \ldots, M_j, \emptyset, \ldots, \emptyset)$ iff $(q\gamma_j \hookrightarrow q'w' \rhd \gamma_i) \in \Delta$, $\gamma_j \in \Gamma_j$, $\gamma_i \in \Gamma_i$, and $i > j$. Such transitions correspond to move the control state from q to q', pop γ_j from the top of the stack, push w' into the stack, and to start executing the task γ_i.
- **Treatment of a new task ($\rightarrow_j$):** $(q, w, M_0, \ldots, M_j + \lfloor \gamma_j \rfloor, \emptyset, \ldots, \emptyset) \rightarrow_j (q', w\gamma_j, M_0, \ldots, M_j, \emptyset, \ldots, \emptyset)$ iff $(q \hookrightarrow q' \lhd \gamma_i) \in \Delta'$, $\gamma_j \in \Gamma_j$, and $w \in \Gamma_0^* \times \cdots \times \Gamma_{j-1}^*$. Such transitions correspond to move the control state from q to q' and to start executing the pending task γ_j if its priority level is strictly greater than the priority level of the topmost symbol in the stack.

Finally, let $\Rightarrow^*_{\mathcal{A}}$ denotes the reflexive and transitive closure of the binary relation $\Rightarrow_{\mathcal{A}}$.

3.2 Reachability Problems

The configuration (resp. control state) reachability problem is to determine, given a k-MPDS $\mathcal{A} = (Q, \Gamma_0, \ldots, \Gamma_k, \Delta, \Delta', q_0, \gamma_0)$ and a configuration $(q, w, M_0, \ldots, M_k)$ (resp. a control state q), whether $(q_0, \epsilon, \lfloor\gamma_0\rfloor, \emptyset, \ldots, \emptyset) \Rightarrow^*_{\mathcal{A}} (q, w, M_0, \ldots, M_k)$ (resp. $(q_0, \epsilon, \lfloor\gamma_0\rfloor, \emptyset, \ldots, \emptyset) \Rightarrow^*_{\mathcal{A}} (q, w', M'_0, \ldots, M'_k)$ for some w' and $M'_0, \ldots, M'_k$). The *empty stack* configuration (resp. control state) reachability problem is to determine, given a k-MPDS $\mathcal{A} = (Q, \Gamma_0, \ldots, \Gamma_k, \Delta, \Delta', q_0, \gamma_0)$ and a control state q, whether $(q_0, \epsilon, \lfloor\gamma_0\rfloor, \emptyset, \ldots, \emptyset) \Rightarrow^*_{\mathcal{A}} (q, \epsilon, \emptyset, \ldots, \emptyset)$ (resp. $(q_0, \epsilon, \lfloor\gamma_0\rfloor, \emptyset, \ldots, \emptyset) \Rightarrow^*_{\mathcal{A}} (q, \epsilon, M'_0, \emptyset, \ldots, \emptyset)$ for some M'_0).

LEMMA 4. *The configuration (resp. control state) reachability problem is polynomially reducible to empty stack configuration (resp. control state) reachability problem for k-MPDSs and vice-versa.*

From now, we sometimes use configuration (resp. control state) reachability problem to denote the empty stack configuration (resp. control state) reachability problem.

3.3 Sub-classes of Multi-set Pushdown Systems with preemption

Two subclasses of our models can be defined by restricting either (1) the way the priorities are assigned to newly created tasks, or (2) the way tasks returns values after their executions.

The first class we define, called Hierarchical k-MPDS, corresponds to systems where created tasks inherit is a priority which at least as high as the one of their parents.

DEFINITION 5. *A* Hierarchical k-MPDS *(k-HMPDS) $\mathcal{A} = (Q, \Gamma_0, \ldots, \Gamma_k, \Delta, \Delta', q_0, \gamma_0)$ is a k-MPDS such that $\Delta \subseteq \bigcup_{j=0}^{k} (Q \times \Gamma_j) \times (Q \times \Gamma_j^* \times (\bigcup_{l \geq j} \Gamma_l \cup \{\epsilon\}))$.*

The second class we consider, called Restricted k-MPDS, corresponds to systems where communication between tasks through shared memory can only happen (1) for tasks of level 0, or (2) between tasks at different levels (at the preemption and resumption operations). In other words, intra-level communication cannot occur between asynchronous tasks of level greater or equal to 1 (but value passing at synchronous procedure calls and returns is not restricted). Formally, the restriction we consider can be modeled by the fact that for each level $j \geq 1$, there is a designated state q_j such that tasks of level j can be treated only if the control state of the system is q_j.

DEFINITION 6. *A* Restricted k-MPDS *(k-RMPDS) is a tuple $\mathcal{R} = (Q, \Gamma_0, \ldots, \Gamma_k, \Delta, \Delta', q_0, \ldots, q_k, \gamma_0)$ where $\mathcal{A} = (Q, \Gamma_0, \ldots, \Gamma_k, \Delta, \Delta', q_0, \gamma_0)$ is an k-MPDS, $q_1, \ldots, q_k \in Q$ is a fixed sequence of states, and $\Delta' \subseteq (Q \times Q \times \Gamma_0) \cup (\bigcup_{j=1}^{k} (\{q_j\} \times \{q_j\} \times \Gamma_j))$.*

4 0-MPDSs vs Petri nets

In the case of 0-MPDS, the decidability of the control state reachability problem has been shown to be decidable in [22]. We present hereafter an alternative proof based on a reduction of this problem to the coverability problem for Petri nets. We show actually that 0-MPDS can be simulated by Petri nets and vice-versa (in some sense that will be made clear later). Our principal aim by showing this relationship between the two models is to introduce smoothly

ideas and constructions which constitute the basis of the constructions presented in the next sections that are our main contributions. Actually, the reduction we show provides also a more robust proof principle, since it allows us to establish the decidability not only for control state reachability problem but also for configuration reachability problem.

4.1 From 0-MPDSs to Petri nets

We prove that every 0-MPDS can be simulated by a Petri net in the following sense:

THEOREM 7. *Given a 0-MPDS $\mathcal{A} = (Q, \Gamma_0, \Delta, \Delta', q_0, \gamma_0)$, it is possible to construct a Petri net $\mathcal{N}$ such that $(q_0, \epsilon, \lfloor \gamma_0 \rfloor) \Rightarrow^*_{\mathcal{A}} (q, \epsilon, M)$ iff $\lfloor q_0 \rfloor + \lfloor \gamma_0 \rfloor \rightarrow^*_{\mathcal{N}} M + \lfloor q \rfloor$.*

Proof (Sketch): First, we observe that for any run $(q_0, \epsilon, \{\gamma_0\}) \Rightarrow^*_{\mathcal{A}} (q, \epsilon, M)$ of $\mathcal{A}$ there are $q_0, q'_0, \ldots, q_n, q'_n \in Q$, $M_1, \ldots, M_n \in M[\Gamma_0]$, and $\gamma_0, \ldots, \gamma_n \in \Gamma_0$ such that:

$$(q_0, \epsilon, \lfloor \gamma_0 \rfloor) \rightarrow_0 (q'_0, \gamma_0, \epsilon) \hookrightarrow^*_0 (q_1, \epsilon, M_1 + \lfloor \gamma_1 \rfloor) \rightarrow_0 (q'_1, \gamma_1, M_1) \hookrightarrow^*_0 (q_2, \epsilon, M_2 + \lfloor \gamma_2 \rfloor) \rightarrow_0 \cdots \hookrightarrow^*_0 (q_{n-1}, \epsilon, M_{n-1} + \lfloor \gamma_{n-1} \rfloor) \rightarrow_0 (q'_{n-1}, \gamma_{n-1}, M_{n-1}) \hookrightarrow^*_0 (q_n, \epsilon, M_n + \lfloor \gamma_n \rfloor) \rightarrow_0 (q'_n, \gamma_n, M_n) \hookrightarrow^*_0 (q, \epsilon, M)$$

(Notice that $\rightsquigarrow_0$ is never used since there are no preemptions for 0-MPDS models.)

Then, the first step of the reduction is to show using Proposition 1 that, for every $p, p' \in Q$ and $\gamma \in \Gamma_0$, the set $\mathcal{M}(p, p', \gamma) = \{M' \mid (p, \gamma, \emptyset) \hookrightarrow^*_0 (p', \epsilon, M')\}$ is a semi-linear set. In fact, a transition rule $p_1\gamma_1 \hookrightarrow p_2 w \rhd \gamma_2$ of Δ (and therefore of $\hookrightarrow_0$) can be seen as a transition rule $p_1\gamma_1 \overset{\gamma_2}{\hookrightarrow} p_2 w$ of the LPDS $\mathcal{P} = (Q, \Gamma_0, \Gamma_0, \delta)$. Thus, a word in the trace language $L_{\mathcal{P}}(p, p', \gamma)$ corresponds to the set of waiting tasks added to the multi-set during the execution of γ to its completion, i.e. $\lfloor L_{\mathcal{P}}(p, p', \gamma) \rfloor = \mathcal{M}(p, p', \gamma)$. Hence, it is possible to construct a finite state automaton $\mathcal{S}_{(p,p',\gamma)} = (S_{(p,p',\gamma)}, \Gamma_0, \delta_{(p,p',\gamma)}, s^i_{(p,p',\gamma)}, s^f_{(p,p',\gamma)})$ such that $\lfloor L(\mathcal{S}_{(p,p',\gamma)}) \rfloor = \mathcal{M}(p, p', \gamma)$.

In the second step, we prove that every run of $\mathcal{A}$ of the form: $(p_1, \epsilon, \lfloor \gamma \rfloor) \rightarrow_0 (p_2, \gamma, \emptyset) \hookrightarrow^*_0 (p_3, \epsilon, M')$ can be simulated by a computation of the Petri net $\mathcal{N}_{(p_1,p_3,\gamma)} = (P_{(p_1,p_3,\gamma)}, T_{(p_1,p_3,\gamma)})$ such that $P_{(p_1,p_3,\gamma)} = Q \cup (\bigcup_{p_2 \in Q} S_{(p_2,p_3,\gamma)}) \cup \Gamma_0$ and $T_{(p_1,p_3,\gamma)}$ is the smallest set of transitions containing:

- **Initialization:** A transition $(p_1\gamma, s^i_{(p_2,p_3,\gamma)})$ for every transition rule $(p_1 \hookrightarrow p_2 \lhd \gamma)$ in Δ'. Such a transition takes a token from each of the places p_1 and γ and puts a token in the place $s^i_{(p_2,p_3,\gamma)}$. This allows to simulate the move $(p_1, \epsilon, \lfloor \gamma \rfloor) \rightarrow_0 (p_2, \gamma, \emptyset)$.
- **Simulation of $\mathcal{S}_{(p_2,p_3,\gamma)}$:** A transition $(s, s'\gamma')$ (resp. (s, s')) for every (s, γ', s') (resp. (s, ϵ, s')) in $\delta_{(p_2,p_3,\gamma)}$. Such transitions allow the simulation of computation of the form $(p_2, \gamma, \emptyset) \hookrightarrow^*_0 (p_3, \epsilon, M')$.

LEMMA 8. *Given a multi-set $M' \in M[\Gamma_0]$, states $p_1, p_2, p_3 \in Q$, and a task $\gamma \in \Gamma_0$, $(p_1, \epsilon, \lfloor \gamma \rfloor) \rightarrow_0 (p_2, \gamma, \emptyset) \hookrightarrow^*_0 (p_3, \epsilon, M')$ iff $\lfloor p_1 \rfloor + \lfloor \gamma \rfloor \rightarrow^*_{N_{(p_1,p_3,\gamma)}} M' + \lfloor s^f_{(p_2,p_3,\gamma)} \rfloor$.*

Since any run that reaches a configuration (q, ϵ, M) can be decomposed as a sequence of runs of the form: $(p_1, \epsilon, \lfloor \gamma \rfloor) \rightarrow_0 (p_2, \gamma, \emptyset) \hookrightarrow^*_0 (p_3, \epsilon, M')$, then $(q_0, \epsilon, \lfloor \gamma_0 \rfloor) \Rightarrow^*_{\mathcal{A}} (q, \epsilon, M)$ can be simulated by the following Petri net $\mathcal{N} = (P, T)$ where: $P = \bigcup_{p_1,p_3 \in Q, \gamma \in \Gamma_0} P_{(p_1,p_3,\gamma)}$ is a finite set of places, and T is the smallest set of transitions such that: $T_{(p_1,p_3,\gamma)} \subseteq T$ and $(s^f_{(p_2,p_3,\gamma)}, p_3)$ is in T for every $p_1, p_2, p_3 \in Q$ and $\gamma \in \Gamma_0$. Hence, Theorem 7 follows immediately from the following lemma:

LEMMA 9. *Given a state $q \in Q$ and a multi-set $M \in M[\Gamma_0]$, $(q_0, \epsilon, \lfloor \gamma_0 \rfloor) \Rightarrow^*_{\mathcal{A}} (q, \epsilon, M)$ iff $W' = M + \lfloor q \rfloor$ is reachable by the Petri net $\mathcal{N}$ from $W = \lfloor \gamma_0 \rfloor + \lfloor q_0 \rfloor$.* □

The following fact follows immediately from Proposition 4 and Theorem 7.

COROLLARY 10. *Configuration and control state reachability for 0-MPDSs are decidable.*

4.2 From Petri nets to 0-MPDSs

We show that every Petri net can be simulated by a 0-MPDS in the following sense:

THEOREM 11. *Given a Petri net $\mathcal{N} = (P, T)$, it is possible to construct a 0-MPDS $\mathcal{A}$ with a special state q_0 such that $W \rightarrow^*_{\mathcal{N}} W'$ iff $(q_0, \epsilon, W) \Rightarrow^*_{\mathcal{A}} (q_0, \epsilon, W')$.*

This can be done by adapting the construction given in [22] to prove the lower bound on the complexity for control state reachability problem for 0-MPDS.

By Theorem 11 and the fact that the set of reachable multi-sets for Petri nets is in general not semi-linear [11], it is possible to show that:

COROLLARY 12. *The set of reachable multi-set configurations $\{M \mid (q_0, \epsilon, \lfloor \gamma_0 \rfloor) \Rightarrow^*_{\mathcal{A}} (q, \epsilon, M)\}$ by an 0-MPDS $\mathcal{A} = (Q, \Gamma_0, \Delta, \Delta', q_0, \gamma_0)$ is in general not semi-linear.*

5 Reachability Analysis for k-MPDSs

In this section, we prove that the configuration (resp. the control state) reachability problem for k-MPDSs is decidable by reduction to the reachability (resp. coverability) problem for Petri nets with weak inhibitor arcs.

THEOREM 13. *Configuration and control state reachability are decidable for k-MPDSs.*

Proof (Sketch): We consider here that $k \geq 1$ (since $k = 0$ has been already considered in the previous section). To simplify the presentation of the proof, we consider first the case of $k = 1$. The generalization to any $k \geq 1$ is given later.

Case $k = 1$: Let $\mathcal{A} = (Q, \Gamma_0, \Gamma_1, \Delta, \Delta', q_0, \gamma_0)$ be an 1-MPDS. Let $\Rightarrow_1 = \rightarrow_1 \cup \hookrightarrow_1$ and $\Rightarrow_0 = \rightsquigarrow_0 \cup \Rightarrow_1 \cup \hookrightarrow_0$ be two transition relations. Thanks to Proposition 4, we consider w.l.o.g configurations of the form $(q, \epsilon, M, \emptyset)$ with $M \in M[\Gamma_0]$. We observe that $(q, \epsilon, M, \emptyset)$ is reachable by $\mathcal{A}$ iff there are some $q_0, q'_0, q_1, \ldots, q_n \in Q$, $\gamma_0, \ldots, \gamma_{n-1} \in \Gamma_0$, and $M_1, \ldots, M_n \in M[\Gamma_0]$ such that $q_n = q$, $M_n = M$, and:

$$\begin{aligned}\textit{Path 0: } & (q_0, \epsilon, \lfloor \gamma_0 \rfloor, \emptyset) \rightarrow_0 (q'_0, \gamma_0, \emptyset, \emptyset) \Rightarrow^*_0 (q_1, \epsilon, M_1 + \lfloor \gamma_1 \rfloor, \emptyset) \rightarrow_0 (q'_1, \gamma_1, M_1, \emptyset) \Rightarrow^*_0 \\ & (q_2, \epsilon, M_2 + \lfloor \gamma_2 \rfloor, \epsilon) \rightarrow_0 (q'_2, \gamma_2, M_2, \emptyset) \cdots (q'_{n-1}, \gamma_{n-1}, M_{n-1}, \emptyset) \Rightarrow^*_0 (q_n, \epsilon, M_n, \emptyset)\end{aligned}$$

Indeed, any computation of $\mathcal{A}$ of the form $(p, \gamma, \emptyset, \emptyset) \Rightarrow^*_0 (p', \epsilon, N, \emptyset)$ for some $p, p' \in Q$, $\gamma \in \Gamma_0$, and $N, N' \in M[\Gamma_0]$, there are $p'_0, p_0, p''_1, p'_1, \ldots, p''_m \in Q$, $\gamma'_0, \ldots, \gamma'_{m-1} \in \Gamma_1$, $w'_0, w_1, w'_1, \ldots, w_m \in \Gamma_0^*$, and $N'_0, N_1, N'_1, \ldots, N_m \in M[\Gamma_0]$ such that:

$$\begin{aligned}\textit{Path 1: } & (p, \gamma, \emptyset, \emptyset) \hookrightarrow^*_0 (p'_0, w'_0, N'_0, \emptyset) \rightsquigarrow_0 (p_0, w_1\gamma'_0, N'_0, \emptyset) \Rightarrow^*_1 (p''_1, w_1, N_1, \emptyset) \hookrightarrow^*_0 \\ & (p'_1, w'_1, N'_1, \emptyset) \rightsquigarrow_0 (p_1, w_2\gamma'_1, N'_1, \emptyset) \Rightarrow^*_1 (p''_2, w_2, N_2, \emptyset) \hookrightarrow^*_0 (p'_2, w'_2, N'_2, \emptyset) \rightsquigarrow_0 \\ & (p_2, w_3\gamma'_2, N'_2, \emptyset) \Rightarrow^*_1 (p''_3, w_3, N_3, \emptyset) \hookrightarrow^*_0 \cdots \Rightarrow^*_1 (p''_m, w_m, N_m, \emptyset) \hookrightarrow^*_0 (p', \epsilon, N, \emptyset)\end{aligned}$$

Then the proof is structured as follows:

- For every $g, g' \in Q$ and $\gamma' \in \Gamma_1$, we construct a Petri net $\mathcal{N}'_{(g,g',\gamma')}$ with a special place c counting the number of pending tasks of priority 1, such that the set of reachable multi-sets when the place c is empty is precisely $\mathcal{M}_1(g, g', \gamma') = \{N' \in M[\Gamma_0] \mid (g, \gamma', \emptyset, \emptyset) \Rightarrow_1^* (g', \epsilon, N', \emptyset)\}$.
- For every $p, p' \in Q$ and $\gamma \in \Gamma_0$, we construct a Petri net $\mathcal{N}_{(p,p',\gamma)}$ with weak inhibitor arcs that characterizes the set $\mathcal{M}_0(p, p', \gamma) = \{N \in M[\Gamma_0] \mid (p, \gamma, \emptyset, \emptyset) \Rightarrow_0^* (p', \epsilon, N, \emptyset)\}$. The Petri net $\mathcal{N}_{(p,p',\gamma)}$ simulates the runs of the form *Path 1* by delegating the $\Rightarrow_1^*$ segments of these runs to the networks $\mathcal{N}'_{(g,g',\gamma')}$ introduced above.
  The difficulties to face in doing that are: (1) transitions $\rightsquigarrow_0$ are not always taken when the stack is empty (since at level 1 the context of the interrrupted task of level 0 is still present in the stack), and (2) the effect of $\Rightarrow_1^*$ computations on the multiset of pending tasks of level 0 must be computed precisely and this should be done only for such computations that reach at their end a configuration where the multiset of pending tasks of level 1 is empty. Since computations at level 1 can be as general as computations of any Petri net, the latter problem needs to be addressed using some notion of place emptyness testing. Then, inhibitor arcs are used to check at the end of $\Rightarrow_1^*$ computations that the place c of $\mathcal{N}'_{(g,g',\gamma')}$ is empty.
  To tackle the first issue, the idea is to reason about the whole computations of level 0 by inserting instead of the level 1 segments a tuple $(g_1, g_2, \gamma') \in Q \times Q \times \Gamma_1$ corresponding to the guess that an interruption by a task γ' of level 1 is able to bring the control state from g_1 to g_2. So, we build a pushdown system labelled by the generated task of level 0 as well as the guessed tuples (g_1, g_2, γ') defined as above. Then, the key observation is that the information represented in the traces of this LPDS can be represented by the traces of a finite state automata $\mathcal{S}$. Indeed, (1) like in the previous section, the ordering between tasks generated by level 0 computations between two given control states does not need to be kept, and (2) it is sufficient to know for each *Path 1* computation how many times each guessing pair (g_1, g_2, γ') occurs; the consistency of these occurrences within the computation (i.e., these guesses can indeed be inserted in the computation) can be checked using a finite control.
  Then, to simulate the computations of the form *Path 1*, the Petri net $\mathcal{N}_{(p,p',\gamma)}$ simulates in parallel the evolution of the control states and the finite-state automaton $\mathcal{S}$ by (1) generating a level 0 task whenever the transition of the automaton is labelled by this task, and (2) simulating the network $\mathcal{N}'(g_1, g_2, \gamma')$ whenever the transition of $\mathcal{S}$ is labelled by (g_1, g_2, γ') where γ' is a level 1 task (which is supposed to be the one which preempts the current level 0 task).
- The collection of all the networks $\mathcal{N}_{(p,p',\gamma)}$ with $p, p' \in Q$ and $\gamma \in \Gamma_0$ are used to build a network $\mathcal{N}_0$ with weak inhibitor arcs that simulates all the runs that reaches a configuration of the form $(q, \epsilon, M, \emptyset)$ (i.e. computations of the form *Path 2*).

Computing $\mathcal{N}'_{(g,g',\gamma')}$: Let $g, g' \in Q$ be a pair of states and $\gamma' \in \Gamma_1$ be a task of priority 1. Then, any computation of the form $(g, \gamma', \emptyset, \emptyset) \Rightarrow_1^* (g', N', M')$ can be seen as a computation of the 0-MPDS $\mathcal{A}_1$ which mimics the execution of $\mathcal{A}$ over tasks of priority 1. Formally, $\mathcal{A}_1$ is defined by the tuple $(Q, \Gamma_0 \cup \Gamma_1, \Delta_1, \Delta'_1, g, \gamma')$ where $\Delta_1 = \Delta$ and $\Delta'_1 = \Delta' \cap (Q \times Q \times \Gamma_1)$. Thus, $(g, \gamma', \emptyset, \emptyset) \Rightarrow_1^* (g', N', M')$ iff $(g, \gamma', \emptyset) \Rightarrow_{\mathcal{A}_1}^* (g', \epsilon, N' + M')$.

Then, by adapting the construction given in the previous section (see Theorem 7) to $\mathcal{A}_1$, we can construct a Petri net $\mathcal{N}'_{(g,g',\gamma')} = (P'_{(g,g',\gamma')}, T'_{(g,g',\gamma')})$ which has two special places c and $t_{g'}$. The place c is used to count the number of pending task of priority 1. A token in the place $t_{g'}$ means that the guessed control state when all tasks of priority level 1 are done is g'. The relation between $\mathcal{A}$ and $\mathcal{N}'_{(g,g',\gamma')}$ is given by the following lemma:

LEMMA 14. *Let $g, g' \in Q$ be a pair of states and $\gamma' \in \Gamma_1$ be a task of priority 1. Then, $(g, \gamma', \emptyset, \emptyset) \Rightarrow_1^* (g', \epsilon, N', \emptyset)$ iff $\lfloor g \rfloor + \lfloor \gamma' \rfloor + \lfloor t_{g'} \rfloor \rightarrow^*_{\mathcal{N}'_{(g,g',\gamma')}} \lfloor g' \rfloor + \lfloor t_{g'} \rfloor + N'$.*

Computing $\mathcal{N}_{(p,p',\gamma)}$: For every $p, p' \in Q$ and $\gamma \in \Gamma_0$, we construct a Petri net $\mathcal{N}_{(p,p',\gamma)}$ with weak inhibitor arcs that simulates computations of $\mathcal{A}$ of the form: $(p, \epsilon, \lfloor \gamma \rfloor, \emptyset) \rightarrow_0 (p'', \gamma, \emptyset, \emptyset) \Rightarrow_0^* (p', \epsilon, M', \emptyset)$. Then, let $\mathcal{P}' = (Q, \Sigma', \Gamma_0, \delta')$ be a LPDS with $\Sigma' = \{(g, g', \gamma') \mid g, g' \in Q, \gamma' \in \Gamma_1\} \cup \Gamma_0$. For every $g, g' \in Q$ and $\gamma' \in \Gamma_1$, (g, g', γ') is a new symbol which represents the set $\mathcal{M}_1(g, g', \gamma')$ (we have showed in the previous paragraph how these sets can be characterized by the Petri nets $\mathcal{N}'_{(g,g',\gamma')}$). The set δ' is defined as the smallest set of transition rules such that:

- If $g_1\gamma_1 \hookrightarrow g_2 w' \rhd \gamma_2 \in \Delta$, $\gamma_1 \in \Gamma_0$, and $\gamma_2 \in \Gamma_0 \cup \{\epsilon\}$, then $g_1\gamma_1 \overset{\gamma_2}{\hookrightarrow} g_2 w' \in \delta'$;
- If $g_1\gamma_1 \hookrightarrow g_2 w' \rhd \gamma' \in \Delta$, $\gamma_1 \in \Gamma_0$, and $\gamma_2 \in \Gamma_1$, then $g_1\gamma_1 \overset{(g_2,g_3,\gamma_2)}{\hookrightarrow} g_3 w' \in \delta'$ for every $g_3 \in Q$. Such a transition rule records in its label the fact that a guess is made at this point of the computation: The level 1 task γ_2 interrupts the level 0 task γ_0, and then the level 1 computation of the form $(g_2, \gamma_2, \emptyset, \emptyset) \Rightarrow_1^* (g_3, \epsilon, N, \emptyset)$ brings the control state from g_2 to g_3.

Thanks to Proposition 1, it is possible to construct a finite state automaton $\mathcal{S}_{(p'',p',\gamma)} = (S_{(p'',p',\gamma)}, \Sigma', \delta_{(p'',p',\gamma)}, s^i_{(p'',p',\gamma)}, s^f_{(p'',p',\gamma)})$ such that $\lfloor L(\mathcal{S}_{(p'',p',\gamma)}) \rfloor = \lfloor L_{\mathcal{P}'}(p'', p', \gamma) \rfloor$. Then, we can define a Petri net with weak inhibitor arcs $\mathcal{N}_{(p,p',\gamma)} = (P_{(p,p',\gamma)}, T_{(p,p',\gamma)})$ using the set of automata $\mathcal{S}_{(p'',p',\gamma)}$ as follows:

- $P_{(p,p',\gamma)} = \{\top, \bot\} \cup P \cup (\bigcup_{p'' \in Q} S_{(p'',p',\gamma)})$ is a finite set of places where $P = \bigcup_{g,g' \in Q, \gamma' \in \Gamma_1} P'_{(g,g',\gamma')}$. The places $\top$ and $\bot$ are flags that indicate if the simulation of $\mathcal{S}_{(p'',p',\gamma)}$ has been initiated or not. When the simulation starts, the place $\top$ is emptied and a token is put in $\bot$. This allows to ensure that the segments $\rightarrow_0 \circ \Rightarrow_0^*$ are simulated in a serial manner and do not interfer.
- The set of transitions $T_{(p,p',\gamma)}$ is the set of the following transitions:
  - **Initialization:** A transition $(\emptyset, \top p\gamma, \bot s^i_{(p'',p',\gamma)})$ for each transition rule $p \hookrightarrow p'' \lhd \gamma$ in Δ'. This transition simulates the move $(p, \epsilon, \lfloor \gamma \rfloor, \emptyset) \rightarrow_0 (p'', \gamma, \emptyset, \emptyset)$.
  - **Simulation of $\mathcal{S}_{(p'',p',\gamma)}$:** A transition rule $(\emptyset, \bot s, \bot s'\gamma')$ (resp. $(\emptyset, \bot s, \bot s')$) for each each transition (s, γ', s') (resp. (s, ϵ, s')) in $\delta_{(p'',p',\gamma)}$ with $\gamma' \in \Gamma_0$. A transition rule $(\emptyset, \bot s, g\, t_{g'}\, \gamma'\, s')$ for each transition $(s, (g, g', \gamma'), s')$ of $\mathcal{S}_{(p,p',\gamma)}$.
  - **Simulation of $\mathcal{N}'_{(g,g',\gamma')}$:** The set of transitions $T'_{(g,g',\gamma')}$ of the network $\mathcal{N}'_{(g,g',\gamma')}$ defined previously for each $g, g' \in Q$ and $\gamma' \in \Gamma_1$.
  - **Checking the guessed tuple** (g, g', γ')**:** A transition rule $(c, g' t_{g'}, \bot)$ for each $g' \in Q$. This rule checks if there are no more tasks of level 1 (i.e. $\lfloor c \rfloor = \emptyset$) and that the control state at the resumption of the preempted task of level 0 is g'.

LEMMA 15. $(p, \epsilon, \lfloor\gamma\rfloor, \emptyset) \rightarrow_0 (p'', \gamma, \emptyset, \emptyset) \Rightarrow_0^* (p', \epsilon, M', \emptyset)$ *iff the multi-set* $W' = M' + \lfloor s^f_{(p'',p',\gamma)}\rfloor + \lfloor\bot\rfloor$ *is reachable by* $\mathcal{N}_{(p,p',\gamma)}$ *from* $W = \lfloor\top\rfloor + \lfloor\gamma\rfloor + \lfloor p\rfloor$.

Computing the Petri net $\mathcal{N}_0$: We observe that any run $(q_0, \epsilon, \{\gamma_0\}, \emptyset) \Rightarrow_{\mathcal{A}}^* (q, \epsilon, M, \emptyset)$ of $\mathcal{A}$ can be simulated by a sequence of executions of the Petri nets $\mathcal{N}_{(p,p',\gamma)}$ with $p, p' \in Q$ and $\gamma \in \Gamma_0$. This can be obtained by defining the Petri net $\mathcal{N}_0 = (P_0, T_0)$ as follows:

- P_0 is a finite union of places $P_{(p,p',\gamma)}$ for every $p, p' \in Q$ and $\gamma \in \Gamma_0$.
- T_0 is the smallest set of transitions satisfying the following conditions:
  - $T_{(p,p',\gamma)} \subseteq T_0$ for every $p, p' \in Q$ and $\gamma \in \Gamma_0$,
  - $(\emptyset, \bot s^f_{(p'',p',\gamma)}, \top p') \in T_0$ for every $p'', p' \in Q$ and $\gamma \in \Gamma_0$, which makes possible the iteration of the executions of Petri nets of the form $\mathcal{N}_{(p,p',\gamma)}$.

Then, Theorem 13 follows immediately from the following lemma:

LEMMA 16. $(q_0, \epsilon, \lfloor\gamma_0\rfloor, \emptyset) \Rightarrow_{\mathcal{A}}^* (q, \epsilon, M, \emptyset)$ *iff* $\lfloor\gamma_0\rfloor + \lfloor q_0\rfloor + \lfloor\top\rfloor \rightarrow_{\mathcal{N}_0}^* M + \lfloor q\rfloor + \lfloor\top\rfloor$.

General Case: This construction can be extended to the case where we have an arbitrary number k of priorities. In this case, we compute a Petri net with *weak inhibitor arcs* as follows: we need k places $c_1, \ldots, c_k$, where c_i counts the number of tasks in the multiset of level i. Then, these places can be ordered as follows: $c_1 > c_2 > \cdots > c_k$. Indeed, in the computed Petri net with inhibitor arcs, we need to check whether $c_i = 0$ only if the other counters $c_j, j \geq i$ are also null. This ensures that the network we construct is a weak inhibitor arcs Petri net. □

6 Reachability Analysis for k-RMPDSs and k-HMPDSs

6.1 Reachability Analysis of Restricted k-MPDSs

In this section, we prove that the configuration and control state reachability problems for k-RMPDSs are decidable and reducible to the same problems for 0-MPDSs. In particular, we show that the control state reachability problem for k-RMPDSs is reducible to the coverability problem for Petri nets. This is based on the fact that it is possible to prove in this case that the sets $\mathcal{M}_1(g, g', \gamma') = \{N' \in M[\Gamma_0] \mid (g, \gamma', \emptyset, \emptyset) \Rightarrow_1^* (g', \epsilon, N', \emptyset)\}$ are semi-linear and can be computed as Parikh images of context free languages. Notice that for the case of (unrestricted) k-MPDS these sets are not semi-linear in general (see Corrollary 12).

THEOREM 17. *For any $k > 0$, control state and configuration reachability problems for k-RMPDSs are reducible to the same problems for some $(k-1)$-RMPDSs.*

Proof (Sketch): Let us fix a 1-RMPDS $\mathcal{R} = (Q, \Gamma_0, \Gamma_1, \Delta, \Delta', q_0, q_1, \gamma_0)$ and its transition relations $\Rightarrow_1 = \rightarrow_1 \cup \hookrightarrow_1$ and $\Rightarrow_0 = \rightsquigarrow_0 \cup \Rightarrow_1 \hookrightarrow_0$. Given $g, g' \in Q$ and $\gamma' \in \Gamma_1$, we construct a context free grammar $G_{(g,g',\gamma')}$ such that $\lfloor L(G_{(g,g',\gamma')})\rfloor = \mathcal{M}_1(g, g', \gamma') = \{N' \in M[\Gamma_0] \mid (g, \gamma', \emptyset, \emptyset) \Rightarrow_1^* (g', \epsilon, N', \emptyset)\}$. Indeed, let us observe that such computations can be decomposed as follows:

$$(g, \gamma', \emptyset, \emptyset) \hookrightarrow_1^* (q_1, \epsilon, N_0^1, N_1^1 + \lfloor\gamma_1\rfloor) \rightarrow_1 (q_1, \gamma_1, N_0^1, N_1^1) \hookrightarrow_1^* (q_1, \epsilon, N_0^2, N_1^2 + \lfloor\gamma_2\rfloor) \rightarrow_1 (q_1, \gamma_2, N_0^2, N_1^2) \hookrightarrow_1^* \cdots \hookrightarrow_1^* (q_1, \epsilon, N_0^n, N_1^n + \lfloor\gamma_n\rfloor) \rightarrow_1 (q_1, \gamma_n, N_0^n, N_1^n) \hookrightarrow_1^* (g', \epsilon, N', \emptyset)$$

Then, given $g_1, g_2 \in Q$ and $\gamma \in \Gamma_1$, it is possible to characterize the set of computations $(g_1, \gamma, \emptyset, \emptyset) \hookrightarrow_1^* (g_2, \epsilon, N_0, N_1)$ by a CFG $G'_{\langle g_1, g_2, \gamma\rangle}$ such that $\lfloor L(G'_{\langle g_1, g_2, \gamma\rangle})\rfloor =$

$\{N_0 + N_1 \mid (g_1, \gamma, \emptyset, \emptyset) \hookrightarrow_1^* (g_2, \epsilon, N_0, N_1)\}$ using a similar construction to the characterization of $\hookrightarrow_0^*$ computations in 0-MPDS (see proof of Theorem 7). Now, we use the set of CFGs $G'_{\langle g,g',\gamma'\rangle}$, $G'_{\langle g,q_1,\gamma'\rangle}$, $G'_{\langle q_1,q_1,\gamma\rangle}$, and $G'_{\langle q_1,g',\gamma\rangle}$ to built $G_{\langle g,g',\gamma'\rangle}$ as follows. Every computation $(g_1, \epsilon, \emptyset, \lfloor\gamma\rfloor) \rightarrow_1 (g_1, \gamma, \emptyset, \emptyset) \hookrightarrow_1^* (g_2, \epsilon, N_0, N_1)$ is simulated by $G_{(g,g',\gamma')}$: γ is rewritten by the axiom of $G'_{\langle g_1,g_2,\gamma\rangle}$, and then rules of $G'_{\langle g_1,g_2,\gamma\rangle}$ are applied. This can be done because the processing order of pending tasks of level 1 is not relevant due to the restriction in RMPDS models (two pending tasks of level 1 cannot communicate).

By Proposition 1, we can construct a finite state automaton $\mathcal{S}_{(g,g',\gamma')}$ such that $\lfloor L(\mathcal{S}_{(g,g',\gamma')})\rfloor = \lfloor L(G_{(g,g',\gamma')})\rfloor = \mathcal{M}_1(g, g', \gamma')$. Then, we construct a 0-MPDS $\mathcal{A}'$ over the alphabet Γ_0 that mimics any computation of $\mathcal{R}$ over tasks of priority 0, i.e. $g_1\gamma_1 \hookrightarrow g_2w' \rhd \gamma_2$ (resp. $g_1 \hookrightarrow g_2 \lhd \gamma_1$) is a rule of $\mathcal{A}'$ iff $g_1\gamma_1 \hookrightarrow g_2w' \rhd \gamma_2$ (resp. $g_1 \hookrightarrow g_2 \lhd \gamma_1$) is a transition rule of $\mathcal{R}$ and $\gamma_1, \gamma_2 \in \Gamma_0 \cup \{\epsilon\}$. On the other hand, any computation $(g, \gamma', \emptyset, \emptyset) \Rightarrow_1^* (g', \epsilon, N', \emptyset)$, with $g, g' \in Q$ and $\gamma' \in \Gamma_1$, can be simulated by a computations of $\mathcal{A}'$ that: (1) moves the control state from g to the initial state of $\mathcal{S}_{(g,g',\gamma')}$; (2) each transition (s, γ'', s') of $\mathcal{S}_{(g,g',\gamma')}$ by rule that moves the control state from s to s' and creates the task γ''; and (3) changes the control state from the final state $\mathcal{S}_{(g,g',\gamma')}$ to g'. Hence, $(q_0, \epsilon, \lfloor\gamma_0\rfloor, \emptyset) \Rightarrow_{\mathcal{R}}^* (q, \epsilon, M, \emptyset)$ iff $(q_0, \epsilon, \lfloor\gamma_0\rfloor) \Rightarrow_{\mathcal{A}'}^* (q, \epsilon, M)$ for any $q \in Q$ and $M \in M[\Gamma_0]$. □

6.2 Reachability Analysis of Hierarchical k-MPDSs

In this section, we study the reachability problem for Hierarchical k-MPDS. We show that control state and configuration reachability problems for k-HMPDSs are decidable using reachability for Petri nets without inhibitor arcs. Here, we use the fact that the set $\mathcal{M}_1(g, g', \gamma') = \{N' \in M[\Gamma_0] \mid (g, \gamma', \emptyset, \emptyset) \Rightarrow_1^* (g', \epsilon, N', \emptyset)\}$ is empty. Indeed, in that case, the only relevant information about level 1 computation segments when simulating *Path 1* computations is whether, given two states g and g′ and a task $\gamma \in \Gamma_1$, it is possible to have a run from $g\gamma$ which reaches a configuration with control state g' where no level 1 tasks are left. This can be solved as a reachability problem in a Petri net simulating the level 1 computations of a configuration where there are no pending tasks of priority 1.

THEOREM 18. *For any $k \geq 0$, the control state and configuration reachability problems for (k)-HMPDSs are reducible to the corresponding problems for $(k-1)$-HMPDSs using the reachability problem for Petri nets.*

7 Conclusion

We have investigated the reachability problem for a model of concurrent programs where tasks (1) can be dynamically created, (2) may have different levels of priority, and (3) may be preempted by tasks of higher priority level. We have shown that this problem is difficult but decidable. Our proof is based on a reduction to the reachability problem in a class of Petri nets with inhibitor arcs. We have also identified a class of models for which the (control point) reachability problem can be reduced to the reachability problem in Petri nets without inhibitor arcs, and another class of models for which the control point reachability problem can be reduced, using Parikh image computations of context-free languages, to the coverability problem in Petri nets. For the latter class, although the problem of solving

the control point reachability problem remains complex (EXPSPACE-hard), we believe that it can be handled in practice using efficient algorithms and tools for (1) computing CFL Parikh images using a Newton method based technique for solving polynomial equations in commutative Kleene algebras [6], and for (2) solving the coverability problem in Petri nets using forward reachability analysis and complete abstraction techniques [8, 9].

References

[1] M. F. Atig, A. Bouajjani, and T. Touili. On the reachability analysis of acyclic networks of pushdown systems. In *CONCUR'08*, LNCS, 2008.
[2] A. Bouajjani and J. Esparza. Rewriting models of boolean programs. In *RTA*, volume 4098 of *LNCS*, pages 136–150. Springer, 2006.
[3] A. Bouajjani, M. Müller-Olm, and T. Touili. Regular symbolic analysis of dynamic networks of pushdown systems. In *CONCUR'05*, LNCS, 2005.
[4] A. Bouajjani and T. Touili. On Computing Reachability Sets of Process Rewrite Systems. In *RTA'05*. LNCS, 2005.
[5] J. Esparza. Decidability and complexity of Petri net problems – an introduction. In G. Rozenberg and W. Reisig, editors, *Lectures on Petri Nets I: Basic Models. Advances in Petri Nets*, number 1491 in Lecture Notes in Computer Science, pages 374–428, 1998.
[6] J. Esparza, S. Kiefer, and M. Luttenberger. On fixed point equations over commutative semirings. In *STACS*, volume 4393 of *LNCS*, pages 296–307. Springer, 2007.
[7] J. Esparza and A. Podelski. Efficient algorithms for pre* and post* on interprocedural parallel flow graphs. In *POPL'00*. ACM, 2000.
[8] G. Geeraerts, J.-F. Raskin, and L. V. Begin. Expand, enlarge, and check: New algorithms for the coverability problem of wsts. In *FSTTCS*, volume 3328 of *LNCS*, 2004.
[9] G. Geeraerts, J.-F. Raskin, and L. V. Begin. Expand, enlarge and check... made efficient. In *CAV*, volume 3576 of *LNCS*, pages 394–407. Springer, 2005.
[10] M. Hack. Decision problems for petri nets and vector addition systems. Technical Report TR 161, 1976.
[11] J. E. Hopcroft and J. J. Pansiot. On the reachability problem for 5-dimensional vector addition systems. *Theor. Comput. Sci.*, 8:135–159, 1979.
[12] R. Jhala and R. Majumdar. Interprocedural analysis of asynchronous programs. In *POPL*, pages 339–350, 2007.
[13] V. Kahlon, F. Ivancic, and A. Gupta. Reasoning about threads communicating via locks. In *CAV*, volume 3576 of *LNCS*. Springer, 2005.
[14] R. Lipton. The reachability problem requires exponential time. Technical Report TR 66, 1976.
[15] D. Lugiez and P. Schnoebelen. The regular viewpoint on pa-processes. *Theor. Comput. Sci.*, 274(1-2):89–115, 2002.
[16] E. W. Mayr. An algorithm for the general petri net reachability problem. In *STOC '81*, pages 238–246, New York, NY, USA, 1981. ACM Press.
[17] R. Mayr. Decidability and Complexity of Model Checking Problems for Infinite-State Systems. Phd. thesis, Technical University Munich, 1998.
[18] R. Parikh. On context-free languages. *J. ACM*, 13(4):570–581, 1966.
[19] S. Qadeer, S. Rajamani, and J. Rehof. Procedure Summaries for Model Checking Multithreaded Software. In *POPL'04*. ACM, 2004.
[20] C. Rackoff. The covering and boundedness problem for vector addition systems. In *TCS*, 1978.
[21] K. Reinhardt. Reachability in petri nets with inhibitor arcs. Revised manuscript, 2006.
[22] K. Sen and M. Viswanathan. Model checking multithreaded programs with asynchronous atomic methods. In *CAV*, pages 300–314, 2006.
[23] S. L. Torre, P. Madhusudan, and G. Parlato. A robust class of context-sensitive languages. In *LICS*, pages 161–170. IEEE, 2007.

Foundations of Software Technology and Theoretical Computer Science (Bangalore) 2008.
Editors: R. Hariharan, M. Mukund, V. Vinay; pp 49-60

Runtime Monitoring of Metric First-order Temporal Properties

David Basin[1], Felix Klaedtke[1], Samuel Müller[2], Birgit Pfitzmann[3]

[1]ETH Zurich, Switzerland
{basin,felixkl}@inf.ethz.ch

[2]IBM Zurich Research Lab and ETH Zurich, Switzerland
sml@zurich.ibm.com

[3]IBM Watson Research Lab, USA
bpfitzm@us.ibm.com

ABSTRACT. We introduce a novel approach to the runtime monitoring of complex system properties. In particular, we present an online algorithm for a safety fragment of metric first-order temporal logic that is considerably more expressive than the logics supported by prior monitoring methods. Our approach, based on automatic structures, allows the unrestricted use of negation, universal and existential quantification over infinite domains, and the arbitrary nesting of both past and bounded future operators. Moreover, we show how to optimize our approach for the common case where structures consist of only finite relations, over possibly infinite domains. Under an additional restriction, we prove that the space consumed by our monitor is polynomially bounded by the cardinality of the data appearing in the processed prefix of the temporal structure being monitored.

1 Introduction

Runtime monitoring [1] is an approach to verifying system properties at execution time by using an online algorithm to check whether a system trace satisfies a temporal property. While novel application areas such as compliance or business activity monitoring [13, 15] require expressive property specification languages, current monitoring techniques are restricted in the properties they can handle. They either support properties expressed in propositional temporal logics and thus cannot cope with variables ranging over infinite domains [6,16,20, 23,29], do not provide both universal and existential quantification [4,12,17,23–25] or only in restricted ways [4,28,30], do not allow arbitrary quantifier alternation [4,22], cannot handle unrestricted negation [8,22,27,30], do not provide quantitative temporal operators [22,25], or cannot simultaneously handle past and future temporal operators [8,22–24,26,27].

In this paper, we present a runtime monitoring approach for an expressive safety fragment of metric first-order temporal logic (MFOTL) [8] that overcomes most of these limitations. The fragment consists of formulae of the form $\Box\,\phi$, where ϕ is bounded, i.e., its temporal operators refer only finitely into the future. Our monitor uses automatic structures [7] to finitely represent infinite structures, which allows for the unrestricted use of negation and

FSTTCS 2008
IARCS Annual Conference on
Foundations of Software Technology and Theoretical Computer Science
http://drops.dagstuhl.de/opus/volltexte/2008/1740

quantification in monitored formulae. Moreover, our monitor supports the arbitrary nesting of both (metric) past and bounded future operators. This means that complex properties can be specified more naturally than with only past operators.[1]

In a nutshell, our monitor works as follows: Given a MFOTL formula $\Box\,\phi$ over a signature S, where ϕ is bounded, we first transform ϕ into a first-order formula $\hat{\phi}$ over an extended signature $\hat{S}$, obtained by augmenting S with auxiliary predicates for every temporal subformula in ϕ. Our monitor then incrementally processes a temporal structure (D, τ) over S and determines for each time point i those elements in (D, τ) that violate ϕ. This is achieved by incrementally constructing a collection of automata that finitely represent the (possibly infinite) interpretations of the auxiliary predicates and by evaluating the transformed first-order formula $\neg\hat{\phi}$ over the extended $\hat{S}$-structure at every time point. In doing so, our monitor discards any information not required for evaluating $\neg\hat{\phi}$ at the current and future time points.

We also show how to adapt our monitoring approach to the common case where all relations are required to be finite and hence relational databases can serve as an alternative to automata. Under the additional (realistic) restriction that time increases after at most a fixed number of time points, our incremental construction ensures that our monitor requires only polynomial space in the cardinality of the data appearing in the processed prefix of the monitored temporal structure. This is in contrast to complexity results for other approaches, such as the logical data expiration technique proposed for 2-FOL [30]. While this logic is at least as expressive as MFOTL, the space required for monitoring (syntactically-restricted) 2-FOL formulae is non-elementary in the cardinality of the data in the processed prefix.

Overall, we see our contributions as follows. First, the presented monitor admits a substantially more expressive logic than previous monitoring approaches. In particular, by supporting arbitrary bounded MFOTL formulae, it significantly extends Chomicki's dynamic integrity checking approach for temporal databases [8]. Second, we extend runtime monitoring to automatic structures, which allows for the unrestricted use of negation and quantification in monitored formulae. Third, for the restricted setting where all relations are finite, we show how to implement our monitor using relational databases. Here, we extend the rewrite procedure of [11] to handle a larger class of temporal formulae. We then prove that, under an additional restriction, the space consumed by our monitor is polynomially bounded in the cardinality of the data appearing in the processed prefix of a monitored temporal structure. Finally, our work shows how to effectively combine ideas from different, but related areas, including database theory, runtime monitoring, model checking, and model theory.

This paper is an extended abstract. Full details are presented in [5].

2 Metric First-order Temporal Logic

In this section, we introduce metric first-order temporal logic (MFOTL) [8], which extends propositional metric temporal logic [19] in a standard way. In the forthcoming sections, we present a method for monitoring requirements formalized within MFOTL.

[1] It is unknown whether the past-only fragment of MFOTL is as expressive as the fragment with both past and bounded future operators and whether formulae in the past-only fragment can be expressed as succinctly as those in the future-bounded fragment.

Syntax and Semantics. Let $\mathbb{I}$ be the set of nonempty intervals over $\mathbb{N}$. We often write an interval in $\mathbb{I}$ as $[c,d)$, where $c \in \mathbb{N}$, $d \in \mathbb{N} \cup \{\infty\}$, and $c < d$, i.e., $[c,d) := \{a \in \mathbb{N} \mid c \leq a < d\}$. A *signature* S is a tuple $(\mathsf{C},\mathsf{R},a)$, where C is a finite set of constant symbols, R is a finite set of predicates disjoint from C, and the function $a : \mathsf{R} \to \mathbb{N}$ associates each predicate $r \in \mathsf{R}$ with an arity $a(r) \in \mathbb{N}$. For the rest of this paper, V denotes a countably infinite set of variables, where we assume that $\mathsf{V} \cap (\mathsf{C} \cup \mathsf{R}) = \emptyset$, for every signature $S = (\mathsf{C},\mathsf{R},a)$. In the following, let $S = (\mathsf{C},\mathsf{R},a)$ be a signature.

DEFINITION 1. *The* formulae *over S are inductively defined: (i) For $t, t' \in \mathsf{V} \cup \mathsf{C}$, $t \approx t'$ and $t \prec t'$ are formulae. (ii) For $r \in \mathsf{R}$ and $t_1, \ldots, t_{a(r)} \in \mathsf{V} \cup \mathsf{C}$, $r(t_1, \ldots, t_{a(r)})$ is a formula. (iii) For $x \in \mathsf{V}$, if θ and θ' are formulae then $(\neg\theta)$, $(\theta \wedge \theta')$, and $(\exists x.\, \theta)$ are formulae. (iv) For $I \in \mathbb{I}$, if θ and θ' are formulae then $(\bullet_I\, \theta)$, $(\bigcirc_I\, \theta)$, $(\theta\, \mathcal{S}_I\, \theta')$, and $(\theta\, \mathcal{U}_I\, \theta')$ are formulae.*

To define the semantics of MFOTL, we need the following notions: A *(first-order) structure* D over S consists of a domain $|D| \neq \emptyset$ and interpretations $c^D \in |D|$ and $r^D \subseteq |D|^{a(r)}$, for each $c \in \mathsf{C}$ and $r \in \mathsf{R}$. A *temporal (first-order) structure* over S is a pair (D,τ), where $D = (D_0, D_1, \ldots)$ is a sequence of structures over S and $\tau = (\tau_0, \tau_1, \ldots)$ is a sequence of natural numbers (time stamps), where:

1. The sequence τ is monotonically increasing (i.e., $\tau_i \leq \tau_{i+1}$, for all $i \geq 0$) and makes progress (i.e., for every $i \geq 0$, there is some $j > i$ such that $\tau_j > \tau_i$).
2. D has constant domains, i.e., $|D_i| = |D_{i+1}|$, for all $i \geq 0$. We denote the domain by $|D|$ and require that $|D|$ is linearly ordered by the relation $<$.
3. Each constant symbol $c \in \mathsf{C}$ has a rigid interpretation, i.e., $c^{D_i} = c^{D_{i+1}}$, for all $i \geq 0$. We denote the interpretation of c by c^D.

A *valuation* is a mapping $v : \mathsf{V} \to |D|$. We abuse notation by applying a valuation v also to constant symbols $c \in \mathsf{C}$, with $v(c) = c^D$. For a valuation v, a variable vector $\bar{x} = (x_1, \ldots, x_n)$, and $\bar{d} = (d_1, \ldots, d_n) \in |D|^n$, $v[\bar{x}/\bar{d}]$ is the valuation that maps x_i to d_i, for i such that $1 \leq i \leq n$, and the valuation of the other variables is unaltered.

DEFINITION 2. *Let (D,τ) be a temporal structure over S, with $D = (D_0, D_1, \ldots)$ and $\tau = (\tau_0, \tau_1, \ldots)$, θ a formula over S, v a valuation, and $i \in \mathbb{N}$. We define $(D,\tau,v,i) \models \theta$ as follows:*

$$
\begin{array}{lll}
(D,\tau,v,i) \models t \approx t' & \textit{iff} & v(t) = v(t') \\
(D,\tau,v,i) \models t \prec t' & \textit{iff} & v(t) < v(t') \\
(D,\tau,v,i) \models r(t_1, \ldots, t_{a(r)}) & \textit{iff} & (v(t_1), \ldots, v(t_{a(r)})) \in r^{D_i} \\
(D,\tau,v,i) \models (\neg\theta_1) & \textit{iff} & (D,\tau,v,i) \not\models \theta_1 \\
(D,\tau,v,i) \models (\theta_1 \wedge \theta_2) & \textit{iff} & (D,\tau,v,i) \models \theta_1 \textit{ and } (D,\tau,v,i) \models \theta_2 \\
(D,\tau,v,i) \models (\exists x.\, \theta_1) & \textit{iff} & (D,\tau,v[x/d],i) \models \theta_1, \textit{ for some } d \in |D| \\
(D,\tau,v,i) \models (\bullet_I\, \theta_1) & \textit{iff} & i > 0,\ \tau_i - \tau_{i-1} \in I, \textit{ and } (D,\tau,v,i-1) \models \theta_1 \\
(D,\tau,v,i) \models (\bigcirc_I\, \theta_1) & \textit{iff} & \tau_{i+1} - \tau_i \in I \textit{ and } (D,\tau,v,i+1) \models \theta_1 \\
(D,\tau,v,i) \models (\theta_1\, \mathcal{S}_I\, \theta_2) & \textit{iff} & \textit{for some } j \leq i,\ \tau_i - \tau_j \in I,\ (D,\tau,v,j) \models \theta_2, \\
 & & \textit{and } (D,\tau,v,k) \models \theta_1, \textit{ for all } k \in [j+1, i+1) \\
(D,\tau,v,i) \models (\theta_1\, \mathcal{U}_I\, \theta_2) & \textit{iff} & \textit{for some } j \geq i,\ \tau_j - \tau_i \in I,\ (D,\tau,v,j) \models \theta_2, \\
 & & \textit{and } (D,\tau,v,k) \models \theta_1, \textit{ for all } k \in [i, j)
\end{array}
$$

Note that the temporal operators are augmented with lower and upper bounds. A temporal formula is only satisfied if it is satisfied within the bounds given by the temporal operator, which are relative to the current time stamp τ_i.

Terminology and Notation. We use standard syntactic sugar such as the standard conventions concerning the binding strength of operators to omit parentheses (e.g., temporal operators bind weaker than Boolean connectives and quantifiers) and we use standard temporal operators (e.g., $\blacklozenge_I \theta := \mathit{true}\, \mathcal{S}_I\, \theta$, where *true* abbreviates $\exists x.\, x \approx x$). Note that the non-metric variants of the temporal operators are easily defined (e.g., $\Box\, \theta := \Box_{[0,\infty)}\, \theta$).

We call formulae of the form $t \approx t'$, $t \prec t'$, and $r(t_1, \ldots, t_{a(r)})$ *atomic*, and formulae with no temporal operators *first-order*. The outermost connective (i.e., Boolean connective, quantifier, or temporal operator) occurring in a formula θ is called the *main connective* of θ. A formula that has a temporal operator as its main connective is a *temporal* formula. A formula θ is *bounded* if the interval I of every temporal operator $\mathcal{U}_I$ occurring in θ is finite.

MFOTL denotes the set of MFOTL formulae and FOL the set of first-order formulae. For $\theta \in \mathsf{MFOTL}$, we define its immediate temporal subformulae $\mathit{tsub}(\theta)$ to be: (i) $\mathit{tsub}(\alpha)$ if $\theta = \neg\alpha$ or $\theta = \exists x.\, \alpha$; (ii) $\mathit{tsub}(\alpha) \cup \mathit{tsub}(\beta)$ if $\theta = \alpha \wedge \beta$; (iii) $\{\theta\}$ if θ is a temporal formula; and (iv) $\emptyset$ otherwise. E.g., for $\theta := (\bullet\, \alpha) \wedge ((\bigcirc\, \beta)\, \mathcal{S}_{[1,9)}\, \gamma)$, we have that $\mathit{tsub}(\theta) = \{\bullet\, \alpha, (\bigcirc\, \beta)\, \mathcal{S}_{[1,9)}\, \gamma\}$.

If $\theta \in \mathsf{MFOTL}$ has the free variables given by the vector $\bar{x} = (x_1, \ldots, x_n)$, we define the set of satisfying assignments at time instance i as

$$\theta^{(D,\tau,i)} := \left\{ \bar{d} \in |D|^n \;\middle|\; (D, \tau, v[\bar{x}/\bar{d}], i) \models \theta, \text{ for some valuation } v \right\}.$$

For $\theta \in \mathsf{FOL}$, we write $(D_i, v) \models \theta$ instead of $(D, \tau, v, i) \models \theta$ and θ^{D_i} for $\theta^{(D,\tau,i)}$. Note that $(D_i, v) \models \theta$ agrees with the standard definition of satisfaction in first-order logic.

3 Monitoring by Reduction to First-order Queries

To effectively monitor MFOTL formulae, we restrict both the formulae and the temporal structures under consideration. We discuss these restrictions in §3.1 and describe monitoring in §3.2–§3.5.

3.1 Restrictions

Throughout this section, let (D, τ) be a temporal structure over the signature $S = (\mathsf{C}, \mathsf{R}, a)$ and ψ the formula to be monitored. We make the following restrictions on ψ and D. First, we require ψ to be of the form $\Box\, \phi$, where ϕ is bounded. It follows that ψ describes a safety property [3]. Note though that not all safety properties can be expressed by formulae of this form [9]. This is in contrast to propositional linear temporal logic, where every safety property can be expressed as $\Box\, \beta$, where β contains only past-time operators [21].

Second, we require that each structure in D is automatic [18]. Roughly speaking, this means that each structure in D can be finitely represented by a collection of automata over finite words. Let us briefly recall some background on automatic structures [7, 18]. Let Σ be an alphabet and # a symbol not in Σ. The *convolution* of the words $w_1, \ldots, w_k \in \Sigma^*$ with $w_i = w_{i1} \cdots w_{i\ell_i}$ is the word

$$w_1 \otimes \cdots \otimes w_k := \begin{bmatrix} w'_{11} \\ \vdots \\ w'_{k1} \end{bmatrix} \cdots \begin{bmatrix} w'_{1\ell} \\ \vdots \\ w'_{k\ell} \end{bmatrix} \in \left(\left(\Sigma \cup \{\#\} \right)^k \right)^*,$$

where $\ell = \max\{\ell_1, \ldots, \ell_k\}$ and $w'_{ij} = w_{ij}$, for $j \leq \ell_i$ and $w'_{ij} = \#$ otherwise. The padding symbol # is added to the words w_i to ensure that all of them have the same length.

DEFINITION 3. *A structure A over a signature $S = (\mathsf{C}, \mathsf{R}, a)$ is* automatic *if there is a regular language $\mathcal{L}_{|A|} \subseteq \Sigma^*$ and a surjective function $\nu : \mathcal{L}_{|A|} \to |A|$ such that the languages $\mathcal{L}_{\approx} := \{u \otimes v \mid u, v \in \mathcal{L}_{|A|} \text{ with } \nu(u) = \nu(v)\}$ and $\mathcal{L}_r := \{u_1 \otimes \cdots \otimes u_{a(r)} \mid u_1, \ldots, u_{a(r)} \in \mathcal{L}_{|D|} \text{ with } (\nu(u_1), \ldots, \nu(u_{a(r)})) \in r^A\}$, for each $r \in \mathsf{R}$, are regular.*

An *automatic representation* of the automatic structure A consists of (i) the function $\nu : \mathcal{L}_{|A|} \to |A|$, (ii) a family of words $(w_c)_{c \in \mathsf{C}}$ with $w_c \in \mathcal{L}_{|A|}$ and $\nu(w_c) = c^A$, for all $c \in \mathsf{C}$, and (iii) a collection $(\mathcal{A}_{|A|}, \mathcal{A}_{\approx}, (\mathcal{A}_r)_{r \in \mathsf{R}})$ of automata that recognize the languages $\mathcal{L}_{|A|}$, $\mathcal{L}_{\approx}$, and $\mathcal{L}_r$, for all $r \in \mathsf{R}$. In the following, we assume that for an automatic structure, we always have an automatic representation for it at hand. A relation $r^A \subseteq |A|^k$ is *regular* if the language $\{u_1 \otimes \cdots \otimes u_k \mid u_1, \ldots, u_k \in \mathcal{L}_{|A|} \text{ with } (\nu(u_1), \ldots, \nu(u_k)) \in r\}$ is regular. Note that an automaton reads the components of the convolution of a representative of $\bar{a} \in |A|^k$ synchronously.

In addition to the requirement that each structure in D is automatic, we require that D has a constant domain representation. This means that the domain of each D_i is represented by the same regular language $\mathcal{L}_{|D|}$ and each word in $\mathcal{L}_{|D|}$ represents the same element in $|D|$, i.e., each automatic representation has the same function $\nu : \mathcal{L}_{|D|} \to |D|$. Finally, we assume that $|D| = \mathbb{N}$ and that $<$ is the standard ordering on $\mathbb{N}$. This is without loss of generality whenever the function ν is injective, i.e., every element in $|D|$ has only one representative in $\mathcal{L}_{|D|}$. Furthermore, note that every automatic structure has an automatic representation in which the function ν is injective [18].

Note that for a first-order formula θ, we can effectively construct an automaton that represents the set θ^{D_i}. Moreover, various basic arithmetical relations are first-order definable in the structure $(\mathbb{N}, <)$ and thus regular. For example, the successor relation $\{(x, y) \in \mathbb{N}^2 \mid y = x + 1\}$ and the relation $\{(x, y) \in \mathbb{N}^2 \mid x + d \leq y\}$, for any $d \in \mathbb{N}$, are regular.

Before presenting our monitoring method, we give two examples of system properties expressed in the MFOTL fragment that our monitor can handle. First, the property "whenever the program variable *in* stores the input x, then x must be stored in the program variable *out* within 5 time units" can be expressed by $\Box \forall x.\, in(x) \rightarrow \Diamond_{[0,6)}\, out(x)$. Second, the property "the value of the program variable v increases by 1 in each step from an initial value 0 until it becomes 5 and then it stays constant" can be formalized as $\Box(\neg(\bullet\, true) \rightarrow v(0)) \wedge (\exists i.\, v(i) \wedge i \prec 5 \rightarrow \bigcirc v(i+1)) \wedge (v(5) \rightarrow \bigcirc v(5))$. Note that we use relations that are singletons to model program variables.

3.2 Overview of the Monitoring Method

To monitor the formula $\Box \phi$ over a temporal structure (D, τ), we incrementally build a sequence of structures $\hat{D}_0, \hat{D}_1, \ldots$ over an extended signature $\hat{S}$. The extension depends on the temporal subformulae of ϕ. For each time point i, we determine the elements that violate ϕ by evaluating a transformed formula $\neg\hat{\phi} \in \mathsf{FOL}$ over $\hat{D}_i$. Observe that with future operators, we usually cannot do this yet when time point i occurs. Our monitor, which we present in §3.5, therefore maintains a list of unevaluated subformulae for past time points. In the following, we first describe how we extend S and transform ϕ. Afterwards, we explain how we incrementally build $\hat{D}_i$. Finally, we present our monitor and prove its correctness.

3.3 Signature Extension and Formula Transformation

In addition to the predicates in R, the extended signature $\hat{S}$ contains an auxiliary predicate p_α for each temporal subformula α of ϕ. For subformulae of the form $\beta\, \mathcal{S}_I\, \gamma$ and $\beta\, \mathcal{U}_I\, \gamma$, we introduce further predicates, which store information that allows us to incrementally update the auxiliary relations.

DEFINITION 4. *Let $\hat{S} := (\hat{\mathsf{C}}, \hat{\mathsf{R}}, \hat{a})$ be the signature with $\hat{\mathsf{C}} := \mathsf{C}$ and $\hat{\mathsf{R}}$ is the union of the sets R, $\{p_\alpha \mid \alpha$ temporal subformula of $\phi\}$, $\{r_\alpha \mid \alpha$ subformula of ϕ of the form $\beta\, \mathcal{S}_I\, \gamma$ or $\beta\, \mathcal{U}_I\, \gamma\}$, and $\{s_\alpha \mid \alpha$ subformula of ϕ of the form $\beta\, \mathcal{U}_I\, \gamma\}$. For $r \in \mathsf{R}$, let $\hat{a}(r) := a(r)$. If α is a temporal subformula with n free variables, then $\hat{a}(p_\alpha) := n$, and $\hat{a}(r_\alpha) := n + 1$ and $\hat{a}(s_\alpha) := n + 2$, if r_α and s_α exist. We assume that $p_\alpha, r_\alpha, s_\alpha \notin \mathsf{C} \cup \mathsf{R} \cup \mathsf{V}$.*

We transform MFOTL formulae over the signature S into first-order formulae over the extended signature $\hat{S}$ as follows.

DEFINITION 5. *For $\theta \in$ MFOTL, we define (i) $\hat{\theta} := \neg\hat{\beta}$ if θ is of the from $\neg\beta$, (ii) $\hat{\theta} := \hat{\beta} \wedge \hat{\gamma}$ if θ is of the form $\beta \wedge \gamma$, (iii) $\hat{\theta} := \exists y.\, \hat{\beta}$ if θ is of the form $\exists y.\, \beta$, (iv) $\hat{\theta} := p_\theta(\bar{x})$ if θ is a temporal formula with the vector of free variables $\bar{x}$, and (v) $\hat{\theta} := \theta$ if θ is an atomic formula.*

We assume throughout this section, without loss of generality, that each subformula of ϕ has the vector of free variables $\bar{x} = (x_1, \ldots, x_n)$. The formula transformation has the following properties, which are easily shown by an induction over the formula structure.

LEMMA 6. *Let θ be a subformula of ϕ. For all $i \in \mathbb{N}$, the following properties hold:*

(i) If $p_\alpha^{\hat{D}_i} = \alpha^{(D,\tau,i)}$ for all $\alpha \in tsub(\theta)$, then $\hat{\theta}^{\hat{D}_i} = \theta^{(D,\tau,i)}$.

(ii) If $p_\alpha^{\hat{D}_i}$ is regular for all $\alpha \in tsub(\theta)$, then $\hat{\theta}^{\hat{D}_i}$ is regular.

3.4 Incremental Extended Structure Construction

We now show how the auxiliary relations in the $\hat{D}_i$s are incrementally constructed. Their instantiations are computed recursively both over time and over the formula structure, where evaluations of subformulae may also be needed from future time points. We later show that this is well-defined and can be evaluated incrementally.

For $c \in \mathsf{C}$ and $r \in \mathsf{R}$, we define $c^{\hat{D}_i} := c^{D_i}$ and $r^{\hat{D}_i} := r^{D_i}$. We address the auxiliary relations for each type of main temporal operator separately.

Previous and Next. For $\alpha = \bullet_I\, \beta$ with $I \in \mathbb{I}$, we define $p_\alpha^{\hat{D}_i}$ as $\hat{\beta}^{\hat{D}_{i-1}}$ if $i > 0$ and $\tau_i - \tau_{i-1} \in I$, and $p_\alpha^{\hat{D}_i} := \emptyset$ otherwise. Intuitively, a tuple $\bar{a}$ is in $p_\alpha^{\hat{D}_i}$ if $\bar{a}$ satisfies β at the previous time point $i - 1$ and the difference of the two successive time stamps is in the interval I.

LEMMA 7. *Let $\alpha = \bullet_I\, \beta$. For $i > 0$, if $p_\delta^{\hat{D}_{i-1}}$ is regular and $p_\delta^{\hat{D}_{i-1}} = \delta^{(D,\tau,i-1)}$ for all $\delta \in tsub(\beta)$, then $p_\alpha^{\hat{D}_i}$ is regular and $p_\alpha^{\hat{D}_i} = \alpha^{(D,\tau,i)}$. Moreover, $p_\alpha^{\hat{D}_0}$ is regular and $p_\alpha^{\hat{D}_0} = \alpha^{(D,\tau,0)}$.*

PROOF. For $i = 0$, the lemma obviously holds. For $i > 0$, the regularity of $p_\alpha^{\hat{D}_i}$ follows from the assumption that the relations $p_\delta^{\hat{D}_{i-1}}$ are regular and Lemma 6(ii). The equality of the two sets follows from Lemma 6(i) and the semantics of the temporal operator $\bullet_I$. ∎

For $\alpha = \bigcirc_I \beta$ with $I \in \mathbb{I}$, we define $p_\alpha^{\hat{D}_i}$ as $\hat{\beta}^{\hat{D}_{i+1}}$ if $\tau_{i+1} - \tau_i \in I$, and $p_\alpha^{\hat{D}_i} := \emptyset$ otherwise. Note that the definition of $p_\alpha^{\hat{D}_i}$ depends on the relations of the next structure D_{i+1} and on the auxiliary relations for $\delta \in \mathit{tsub}(\beta)$ of the next extended structure $\hat{D}_{i+1}$. Hence, the monitor instantiates $p_\alpha^{\hat{D}_i}$ with a delay of at least one time step.

LEMMA 8. *Let $\alpha = \bigcirc_I \beta$. If $p_\delta^{\hat{D}_{i+1}}$ is regular and $p_\delta^{\hat{D}_{i+1}} = \delta^{(D,\tau,i+1)}$ for all $\delta \in \mathit{tsub}(\beta)$, then $p_\alpha^{\hat{D}_i}$ is regular and $p_\alpha^{\hat{D}_i} = \alpha^{(D,\tau,i)}$.*

Since and Until. We first address the past-time operator $\mathcal{S}_I$ with $I = [c, d) \in \mathbb{I}$. Assume that $\alpha = \beta \, \mathcal{S}_I \, \gamma$. We start with the initialization and update of the auxiliary relations for r_α. We define $r_\alpha^{\hat{D}_0} := \hat{\gamma}^{\hat{D}_0} \times \{0\}$ and for $i > 0$, we define

$$r_\alpha^{\hat{D}_i} := \big(\hat{\gamma}^{\hat{D}_i} \times \{0\}\big) \cup \big\{ (\bar{a}, y) \in \mathbb{N}^{n+1} \;\big|\; \bar{a} \in \hat{\beta}^{\hat{D}_i},\ y < d,\ \text{and } (\bar{a}, y') \in r_\alpha^{\hat{D}_{i-1}},\ \text{for } y' = y - \tau_i + \tau_{i-1} \big\}.$$

Intuitively, a pair $(\bar{a}, y)$ is in $r_\alpha^{\hat{D}_i}$ if $\bar{a}$ satisfies α at time point i independent of the lower bound c, where the "age" y indicates how long ago the formula γ was satisfied by $\bar{a}$. If $\bar{a}$ satisfies γ at the time point i, it is added to $r_\alpha^{\hat{D}_i}$ with the age 0. For $i > 0$, we additionally update the tuples $(\bar{a}, y) \in r_\alpha^{\hat{D}_{i-1}}$. First, $\bar{a}$ must satisfy β at the time point i. Second, the age is adjusted by the difference of the time stamps τ_{i-1} and τ_i. Third, the new age must be less than d, otherwise it is too old to satisfy α.

The arithmetic constraint $y' = y - \tau_i + \tau_{i-1}$ in the definition of $r_\alpha^{\hat{D}_i}$ for $i > 0$ is first-order definable in D. Note that $\tau_i + \tau_{i-1}$ is a constant value. Now it is not hard to see that $r_\alpha^{\hat{D}_i}$ is regular if all its components are regular.

With the relation $r_\alpha^{\hat{D}_i}$, we can determine the elements that satisfy α at the time point i. We define $p_\alpha^{\hat{D}_i} := \big\{ \bar{a} \in \mathbb{N}^n \;\big|\; (\bar{a}, y) \in r_\alpha^{\hat{D}_i}, \text{for some } y \geq c \big\}$.

LEMMA 9. *Let $\alpha = \beta \, \mathcal{S}_{[c,d)} \, \gamma$. Assume that $p_\delta^{\hat{D}_j}$ is regular and $p_\delta^{\hat{D}_j} = \delta^{(D,\tau,j)}$, for all $j \leq i$ and $\delta \in \mathit{tsub}(\beta) \cup \mathit{tsub}(\gamma)$. Then the following properties hold:*

(i) The relation $r_\alpha^{\hat{D}_i}$ is regular and for all $\bar{a} \in \mathbb{N}^n$ and $y \in \mathbb{N}$,

$$(\bar{a}, y) \in r_\alpha^{\hat{D}_i} \quad \textit{iff} \quad \begin{array}{c} \textit{there is a } j \in [0, i+1) \textit{ such that } y = \tau_i - \tau_j < d\,, \bar{a} \in \gamma^{(D,\tau,j)}\,, \\ \textit{and } \bar{a} \in \beta^{(D,\tau,k)},\ \textit{for all } k \in [j+1, i+1)\,. \end{array}$$

(ii) The relation $p_\alpha^{\hat{D}_i}$ is regular and $p_\alpha^{\hat{D}_i} = \alpha^{(D,\tau,i)}$.

Note that the definition of $r_\alpha^{\hat{D}_i}$ only depends on the relation $r_\alpha^{\hat{D}_{i-1}}$, if $i > 0$, and on the relations in $\hat{D}_i$ for which the corresponding predicates occur in the subformulae of $\hat{\beta}$ or $\hat{\gamma}$. Furthermore, the definition of $p_\alpha^{\hat{D}_i}$ only depends on $r_\alpha^{\hat{D}_i}$.

We now address the bounded future-time operator $\mathcal{U}_I$ with $I = [c, d) \in \mathbb{I}$ and $d \in \mathbb{N}$. Assume that $\alpha = \beta \, \mathcal{U}_I \, \gamma$. For all $i \in \mathbb{N}$, let $\ell_i := \max\{ j \in \mathbb{N} \mid \tau_{i+j} - \tau_i < d \}$. We call ℓ_i the lookahead offset at time point i. For convenience, let $\ell_{-1} := 0$. To instantiate the relation $p_\alpha^{\hat{D}_i}$, only the relations $p_\delta^{\hat{D}_i}, \ldots, p_\delta^{\hat{D}_{i+\ell_i}}$ are relevant, where $\delta \in \mathit{tsub}(\beta) \cup \mathit{tsub}(\gamma)$. The definition of $p_\alpha^{\hat{D}_i}$ is based on the auxiliary relations $r_\alpha^{\hat{D}_i}$ and $s_\alpha^{\hat{D}_i}$, which we first show how to initialize and update.

We define $r_\alpha^{\hat{D}_i}$ as the union of the sets N_r and U_r. N_r contains the tuples that are new in the sense that they are obtained from data at the time points $i+\ell_{i-1},\dots,i+\ell_i$; U_r contains the updated data from the time points $i,\dots,i+\ell_{i-1}-1$. Formally, we define

$$N_r := \left\{(\bar{a},j) \in \mathbb{N}^{n+1} \;\middle|\; \ell_{i-1} \le j \le \ell_i,\ \bar{a} \in \hat{\gamma}^{\hat{D}_{i+j}},\ \text{and } \tau_{i+j}-\tau_i \ge c\right\}$$

$$U_r := \begin{cases} \left\{(\bar{a},j) \in \mathbb{N}^{n+1} \;\middle|\; (\bar{a},j+1) \in r_\alpha^{\hat{D}_{i-1}} \text{ and } \tau_{i+j}-\tau_i \ge c\right\} & \text{if } i>0,\\ \emptyset & \text{otherwise.}\end{cases}$$

Intuitively, $r_\alpha^{\hat{D}_i}$ stores the tuples satisfying the formula $\Diamond_I \gamma$ at the time point i, where each tuple in $r_\alpha^{\hat{D}_i}$ is augmented by the index relative to i where the tuple satisfies γ.

Similarly to $r_\alpha^{\hat{D}_i}$, the relation $s_\alpha^{\hat{D}_i}$ is the union of a set N_s for the new elements and a set U_s for the updates. These two sets are defined as

$$N_s := \left\{(\bar{a},j,j') \in \mathbb{N}^{n+2} \;\middle|\; \ell_{i-1} \le j \le j' \le \ell_i \text{ and } \bar{a} \in \hat{\beta}^{\hat{D}_{i+k}},\ \text{for all } k \in [j,j'+1)\right\}$$

and $U_s := \emptyset$ if $i=0$, and

$$\begin{aligned} U_s := &\left\{(\bar{a},j,j') \in \mathbb{N}^{n+2} \;\middle|\; (\bar{a},j+1,j'+1) \in s_\alpha^{\hat{D}_{i-1}}\right\} \cup \\ &\left\{(\bar{a},j,j') \in \mathbb{N}^{n+2} \;\middle|\; (\bar{a},j+1,\ell_{i-1}) \in s_\alpha^{\hat{D}_{i-1}} \text{ and } (\bar{a},\ell_{i-1},j') \in N_s\right\}\end{aligned}$$

otherwise. Intuitively, $s_\alpha^{\hat{D}_i}$ stores the tuples and the bounds of the interval (relative to i) in which β is satisfied.

With the relations $r_\alpha^{\hat{D}_i}$ and $s_\alpha^{\hat{D}_i}$ at hand, we define

$$p_\alpha^{\hat{D}_i} := \left\{\bar{a} \in \mathbb{N}^n \;\middle|\; (\bar{a},j) \in r_\alpha^{\hat{D}_i} \text{ and } (\bar{a},0,j') \in s_\alpha^{\hat{D}_i}, \text{for some } j \le j'+1\right\}.$$

LEMMA 10. *Let $\alpha = \beta\, \mathcal{U}_I\, \gamma$. Assume that $p_\delta^{\hat{D}_k}$ is regular and $p_\delta^{\hat{D}_k} = \delta^{(D,\tau,k)}$, for all $k \le i+\ell_i$ and $\delta \in tsub(\beta) \cup tsub(\gamma)$. Then the following properties hold:*

(i) The relation $r_\alpha^{\hat{D}_i}$ is regular and for all $\bar{a} \in \mathbb{N}$ and $j \in \mathbb{N}$,

$$(\bar{a},j) \in r_\alpha^{\hat{D}_i} \quad \textit{iff} \quad \bar{a} \in \gamma^{(D,\tau,i+j)} \text{ and } \tau_{i+j}-\tau_i \in I\,.$$

(ii) The relation $s_\alpha^{\hat{D}_i}$ is regular and for all $\bar{a} \in \mathbb{N}^n$ and $j,j' \in \mathbb{N}$,

$$(\bar{a},j,j') \in s_\alpha^{\hat{D}_i} \quad \textit{iff} \quad j \le j',\ \tau_{i+j'}-\tau_i < d,\ \text{and } \bar{a} \in \beta^{(D,\tau,i+k)},\ \text{for all } k \in [j,j'+1)\,.$$

(iii) The relation $p_\alpha^{\hat{D}_i}$ is regular and $p_\alpha^{\hat{D}_i} = \alpha^{(D,\tau,i)}$.

3.5 Monitor and Correctness

Figure 1 presents the monitor $\mathcal{M}(\phi)$. Without loss of generality, it assumes that each temporal subformula occurs only once in ϕ. In the following, we outline its operation.

The monitor uses two counters i and q. The counter i is the index of the current element (D_i,τ_i) in the input sequence $(D_0,\tau_0),(D_1,\tau_1),\dots$, which is processed sequentially. Initially, i is 0 and it is incremented at the end of each loop iteration (lines 4–16). The counter $q \le i$ is the index of the next time point q (possibly in the past, from the point of view of i) for which we evaluate $\neg\hat{\phi}$ over the structure $\hat{D}_q$. The evaluation is delayed until the relations $p_\alpha^{\hat{D}_q}$ for $\alpha \in tsub(\phi)$ are all instantiated (lines 10–13). Furthermore, the monitor uses the list[2]

[2]We abuse notation by using set notation for lists. Moreover, we assume that Q is ordered in that (α,j,S) occurs before (α',j',S'), whenever α is a proper subformula of α', or $\alpha=\alpha'$ and $j<j'$.

1: $i \leftarrow 0$ *% current index in input sequence* $(D_0, \tau_0), (D_1, \tau_1), \ldots$
2: $q \leftarrow 0$ *% index of next query evaluation in sequence* $(D_0, \tau_0), (D_1, \tau_1), \ldots$
3: $Q \leftarrow \{((\alpha, 0, \mathit{waitfor}(\alpha)) \mid \alpha \text{ temporal subformula of } \phi\}$
4: **loop**
5: Carry over constants and relations of D_i to $\hat{D}_i$.
6: **for all** $(\alpha, j, \emptyset) \in Q$ **do** *% respect ordering of subformulae*
7: Build relations for α in $\hat{D}_j$ (e.g., build $r_\alpha^{\hat{D}_j}$ and $p_\alpha^{\hat{D}_j}$ if $\alpha = \beta \,\mathcal{S}_I\, \gamma$).
8: Discard auxiliary relations for α in $\hat{D}_{j-1}$ if $j - 1 \geq 0$ (e.g., discard $r_\alpha^{\hat{D}_{j-1}}$ if $\alpha = \beta \,\mathcal{S}_I\, \gamma$).
9: Discard relations $p_\delta^{\hat{D}_j}$, where δ is a temporal subformula of α.
10: **while** all relations $p_\alpha^{\hat{D}_q}$ are built for $\alpha \in \mathit{tsub}(\phi)$ **do**
11: Output valuations violating ϕ at time point q, i.e., output $(\neg\hat{\phi})^{\hat{D}_q}$ and q.
12: Discard structure $\hat{D}_{q-1}$ if $q - 1 \geq 0$.
13: $q \leftarrow q + 1$
14: $Q \leftarrow \{(\alpha, i+1, \mathit{waitfor}(\alpha)) \mid \alpha \text{ temporal subformula of } \phi\} \cup$
$\{(\alpha, j, \bigcup_{\theta \in \mathit{update}(S, \tau_{i+1} - \tau_i)} \mathit{waitfor}(\theta)) \mid (\alpha, j, S) \in Q \text{ and } S \neq \emptyset\}$
15: $i \leftarrow i + 1$ *% process next element in input sequence* (D_{i+1}, τ_{i+1})
16: **end loop**

Figure 1: Monitor $\mathcal{M}(\phi)$

Q to ensure that the auxiliary relations of $\hat{D}_0, \hat{D}_1, \ldots$ are built at the right time: if $(\alpha, j, \emptyset)$ is an element of Q at the beginning of a loop iteration, enough time has elapsed to build the relations for the temporal subformula α of the structure $\hat{D}_j$. The monitor initializes Q in line 3. The function *waitfor* extracts the subformulae that cause a delay of the formula evaluation. We define $\mathit{waitfor}(\theta)$ to be: (i) $\mathit{waitfor}(\beta)$ if $\theta = \neg\beta$, $\theta = \exists x.\, \beta$, or $\theta = \bullet_I\, \beta$; (ii) $\mathit{waitfor}(\beta) \cup \mathit{waitfor}(\gamma)$ if $\theta = \beta \wedge \gamma$ or $\theta = \beta \,\mathcal{S}_I\, \gamma$, (iii) $\{\theta\}$ if $\theta = \bigcirc_I\, \beta$ or $\theta = \beta \,\mathcal{U}_I\, \gamma$, and (iv) $\emptyset$ otherwise. The list Q is updated in line 14 before we increment i and start a new loop iteration. For the update we use the function *update* that is defined as

$$\begin{aligned}\mathit{update}(U, \Delta) := {} & \{\beta \mid \bigcirc_I\, \beta \in U\} \cup \{\beta \,\mathcal{U}_{[\max\{0, c-\Delta\}, d-\Delta)}\, \gamma \mid \beta \,\mathcal{U}_{[c,d)}\, \gamma \in U, \text{ with } d - \Delta > 0\} \cup \\ & \{\beta \mid \beta \,\mathcal{U}_{[c,d)}\, \gamma \in U \text{ or } \gamma \,\mathcal{U}_{[c,d)}\, \beta \in U, \text{ with } d - \Delta \leq 0\}\,,\end{aligned}$$

for a formula set U and $\Delta \in \mathbb{N}$. The update adds a new tuple $(\alpha, i+1, \mathit{waitfor}(\alpha))$ to Q, for each temporal subformula α of ϕ, and it removes the tuples of the form $(\alpha, j, \emptyset)$ from Q. Moreover, for tuples (α, j, S) with $S \neq \emptyset$, the set S is updated using the functions *waitfor* and *update* by taking into account the elapsed time to the next time point, i.e. $\tau_{i+1} - \tau_i$.

In lines 6–9, we build the relations for which enough time has elapsed, i.e., the auxiliary relations for α in $\hat{D}_j$ with $(\alpha, j, \emptyset) \in Q$. Since a tuple $(\alpha', j, \emptyset)$ does not occur before a tuple $(\alpha, j, \emptyset)$ in Q, where α is a subformula of α', the relations in $\hat{D}_j$ for α are built before those for α'. To build the relations, we use the incremental constructions described earlier in this section. We thus discard certain relations after we have built the relations for α in $\hat{D}_j$ to reduce space consumption. For instance, if $j > 0$ and $\alpha = \beta \,\mathcal{S}_I\, \gamma$, we discard the relation $r_\alpha^{\hat{D}_{j-1}}$, and we discard $r_\alpha^{\hat{D}_{j-1}}$ and $s_\alpha^{\hat{D}_{j-1}}$ when $\alpha = \beta \,\mathcal{U}_I\, \gamma$.

In lines 10–13, the valuations violating ϕ at time point q are output together with q, for all q where the relations $p_\alpha^{\hat{D}_q}$ of all immediate temporal subformulae α of ϕ have been built. After an output, the remainder of the extended structure $\hat{D}_{q-1}$ is discarded and q is incremented by 1.

THEOREM 11. *The monitor $\mathcal{M}(\phi)$ from Figure 1 has the following properties:*

(i) Whenever $\mathcal{M}(\phi)$ outputs $(\neg\hat{\phi})^{\hat{D}_q}$, then $(\neg\hat{\phi})^{\hat{D}_q} = (\neg\phi)^{(D,\tau,q)}$. Furthermore, the set $(\neg\hat{\phi})^{\hat{D}_q}$ is effectively constructable and finitely representable.

(ii) For every $n \in \mathbb{N}$, $\mathcal{M}(\phi)$ eventually sets the counter q to n in some loop iteration.

4 MFOTL Monitoring with Finite Relations

In this section, we sketch how to use relational databases as an alternative to automata for implementing our monitor and analyze its space complexity. Details are provided in [5].

In the following, we assume that all relations are finite and thus can be stored in a relational database. When replacing "regular" by "finite", however, our constructions from §3.4, in particular Lemmas 7–10, become invalid. The problem is that the auxiliary relations constructed for the temporal subformulae are possibly infinite. We overcome this problem by extending work from database theory on domain independence [14]. In particular, we generalize the solutions for first-order queries [2] and non-metric first-order temporal logic [8, 10, 11] to MFOTL formulae by trying to rewrite the given MFOTL formula ϕ so that all temporal subformulae and their direct subformulae have only finitely many satisfying valuations. After rewriting the formula ϕ, we check, based on the syntax of the result ψ, whether each $\theta \in \{\alpha \mid \alpha = \psi, \alpha \text{ is a temporal subformula of } \psi, \text{ or } \alpha \text{ is a direct subformula of a temporal subformula of } \psi\}$ is *temporal domain independent*. If ψ passes this check, we know that it can be handled by our monitor for finite relations. Otherwise, no conclusions can be drawn. For the rest of this section, we assume that ϕ, all temporal subformulae of ϕ, and all direct subformulae of temporal subformulae of ϕ are temporal domain independent.

We now analyze the memory consumption of our monitor for finite relations. To obtain a polynomial bound on the memory consumption, we modify $\mathcal{M}(\phi)$ as follows: (i) the counters i and q are replaced by the relative counter $i - q$ and (ii) the update constructions for subformulae of the form $\alpha = \beta\, \mathcal{S}_{[c,\infty)}\, \gamma$ are modified to prevent the "age" y of a tuple $(\bar{a}, y) \in r_\alpha^{\hat{D}_{i-1}}$ from increasing forever. The analyze the resources consumed by monitors in general, we introduce the following abstract notion. Let C be a class of temporal structures over the signature $S = (\mathsf{C}, \mathsf{R}, a)$ and let $pre(C)$ denote the set of nonempty finite prefixes of the temporal structures in C.

DEFINITION 12. *Let $f, g : pre(C) \to \mathbb{N}$ and $s : \mathbb{N} \to \mathbb{N}$ be functions. We write $f \lhd^s g$ if $f(\bar{D}, \bar{\tau}) < s(g(\bar{D}, \bar{\tau}))$, for all $(\bar{D}, \bar{\tau}) \in pre(C)$.*

In our context, the function $f : pre(C) \to \mathbb{N}$ measures the consumption of a particular resource (e.g., storage) of a monitor after it has processed the finite prefix $(\bar{D}, \bar{\tau})$. The function $g : pre(C) \to \mathbb{N}$ measures the size of the prefix $(\bar{D}, \bar{\tau})$. Intuitively, $f \lhd^s g$ means that, at any time point, the resource consumption (measured by f) of the monitor is bounded by the function $s : \mathbb{N} \to \mathbb{N}$ with respect to the size of the processed prefix (measured by g) of an input from C. We use the following concrete functions f and g. Let $(\bar{D}, \bar{\tau}) \in pre(C)$ with $\bar{D} = (D_0, \ldots, D_i)$ and $\bar{\tau} = (\tau_0, \ldots, \tau_i)$.

– We define $g(\bar{D}, \bar{\tau}) := |adom(\bar{D})|$, where $adom(\bar{D})$ is the active domain of $(\bar{D}, \bar{\tau})$, i.e., $adom(\bar{D}) := \{c^{D_0} \mid c \in \mathsf{C}\} \cup \bigcup_{0 \le k \le i} \bigcup_{r \in \mathsf{R}} \{d_j \mid (d_1, \ldots, d_{a(r)}) \in r^{D_k} \text{ and } 1 \le j \le a(r)\}$.

Note that g only counts the number of elements of $\bar{D}$ that are constants or that occur in some of $\bar{D}$'s relations. It ignores the sizes of these elements as well as the number of times and where an element appears in $\bar{D}$. It also ignores the time stamps in $\bar{\tau}$.

– We define $f(\bar{D}, \bar{\tau})$ to be the sum of the cardinalities of the relations for $r \in \hat{\mathsf{R}}$ stored by $\mathcal{M}(\phi)$ after the $(i+1)$st loop iteration, having processed the input $(D_0, \tau_0), \ldots, (D_i, \tau_i)$.

Note that $f \lhd^s g$ is a desirable property of a monitor. It says that the amount of data stored does not depend on how long the monitor has been running but only on the number of domain elements that appeared so far, and that the stored data is bounded by the function s. We remark that the property of a (polynomially) bounded history encoding [8] can be formalized as $f \lhd^s g$, for some (polynomial) $s : \mathbb{N} \to \mathbb{N}$.

THEOREM 13. *Let C be a class of temporal databases. Assume that there is some $\ell \in \mathbb{N}$ such that $\max\{j \mid \tau_i = \tau_{i+1} = \ldots = \tau_{i+j}\} < \ell$, for all $(D, \tau) \in C$ and all $i \in \mathbb{N}$. Then, we have that $f \lhd^s g$, where $s : \mathbb{N} \to \mathbb{N}$ is a polynomial of degree $\max\{a(r) \mid r \in \hat{\mathsf{R}}\}$.*

Note that if such a bound ℓ on the sequence τ of time stamps does not exist, we cannot guarantee any upper bound on f. It is open whether Theorem 13 can be carried over to temporal structures with possibly infinite relations and automatic representations.

5 Conclusion and Future Work

We have presented an automata-based monitoring approach for an expressive fragment of a metric first-order temporal logic. The use of automata substantially generalizes both the kinds of structures and the class of formulae that can be monitored. Moreover, it eliminates the limitations that arise in databases, where relations must be finite. An interesting question here is to what extent the use of automatic structures can be carried over to other monitoring approaches, thereby solving the problems they have with infinite relations.

One direction for future work is to explore whether our approach can be used to monitor temporal first-order logics that have an interval-based semantics instead of a point-based semantics, or a combined interval and point-based semantics, which is useful for modeling state and event predicates. Another direction is to conduct a refined complexity analysis for our algorithm with automatic structures and to validate our results by implementation and testing. In particular, we plan to design and evaluate data structures and algorithms for efficiently incrementally updating relations, which is at the heart of our monitoring algorithm.

References

[1] *Proceedings of the 1st to 8th Workshop on Runtime Verification (RV)*, 2001–2008.
[2] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
[3] B. Alpern and F. Schneider. Defining liveness. *Inf. Process. Lett.*, 21(4):181–185, 1985.
[4] H. Barringer, A. Goldberg, K. Havelund, and K. Sen. Rule-based runtime verification. In *Verification, Model Checking, and Abstract Interpretation (VMCAI'04)*, vol. 2937 of *LNCS*, pp. 44–57.
[5] D. Basin, F. Klaedtke, S. Müller, and B. Pfitzmann. Runtime monitoring of metric first-order temporal properties. Technical Report RZ 3702, IBM Research and ETH Zurich, 2008.
[6] A. Bauer, M. Leucker, and C. Schallhart. Monitoring of real-time properties. In *Foundations of Software Technology and Theoretical Computer Science (FSTTCS'06)*, vol. 4337 of *LNCS*, pp. 260–272.

[7] A. Blumensath and E. Grädel. Finite presentations of infinite structures: Automata and interpretations. *Theory Comput. Syst.*, 37(6):641–674, 2004.
[8] J. Chomicki. Efficient checking of temporal integrity constraints using bounded history encoding. *ACM Trans. Database Syst.*, 20(2):149–186, 1995.
[9] J. Chomicki and D. Niwiński. On the feasibility of checking temporal integrity constraints. *J. Comput. Syst. Sci.*, 51(3):523–535, 1995.
[10] J. Chomicki and D. Toman. Implementing temporal integrity constraints using an active DBMS. *IEEE Trans. on Knowl. and Data Eng.*, 7(4):566–582, 1995.
[11] J. Chomicki, D. Toman, and M. Böhlen. Querying ATSQL databases with temporal logic. *ACM Trans. Database Syst.*, 26(2):145–178, 2001.
[12] B. D'Angelo, S. Sankaranarayanan, C. Sánchez, W. Robinson, B. Finkbeiner, H. Sipma, S. Mehrotra, and Z. Manna. LOLA: Runtime monitoring of synchronous systems. In *Temporal Representation and Reasoning (TIME'05)*, pp. 166–174.
[13] N. Dinesh, A. Joshi, I. Lee, and O. Sokolsky. Checking traces for regulatory conformance. In *Runtime Verification (RV'08)*.
[14] R. Fagin. Horn clauses and database dependencies. *J. ACM*, 29(4):952–985, 1982.
[15] C. Giblin, A. Liu, S. Müller, B. Pfitzmann, and X. Zhou. Regulations expressed as logical models (REALM). In *Legal Knowledge and Information Systems (JURIX'05)*, vol. 134 of *Frontiers in Artificial Intelligence and Applications*, pp. 37–48.
[16] K. Havelund and G. Rosu. Efficient monitoring of safety properties. *Int. J. Softw. Tools Technol. Transf.*, 6(2):158–173, 2004.
[17] J. Håkansson, B. Jonsson, and O. Lundqvist. Generating online test oracles from temporal logic specifications. *Int. J. Softw. Tools Technol. Transf.*, 4(4):456–471, 2003.
[18] B. Khoussainov and A. Nerode. Automatic presentations of structures. In *Logical and Computational Complexity*, vol. 960 of *LNCS*, pp. 367–392, 1995.
[19] R. Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Systems*, 2(4):255–299, 1990.
[20] K. Kristoffersen, C. Pedersen, and H. Andersen. Runtime verification of timed LTL using disjunctive normalized equation systems. *Electr. Notes Theor. Comput. Sci.*, 89(2):210–225, 2003.
[21] O. Lichtenstein, A. Pnueli, and L. Zuck. The glory of the past. In *Logic of Programs*, vol. 193 of *LNCS*, pp. 196–218, 1985.
[22] U. Lipeck and G. Saake. Monitoring dynamic integrity constraints based on temporal logic. *Inf. Syst.*, 12(3):255–269, 1987.
[23] O. Maler, D. Nickovic, and A. Pnueli. From MITL to timed automata. In *Formal Modeling and Analysis of Timed Systems (FORMATS'06)*, vol. 4202 of *LNCS*, pp. 274–289.
[24] D. Nickovic and O. Maler. AMT: A property-based monitoring tool for analog systems. In *Formal Modeling and Analysis of Timed Systems (FORMATS'07)*, vol. 4763 of *LNCS*, pp. 304–319.
[25] M. Roger and J. Goubault-Larrecq. Log auditing through model-checking. In *Computer Security Foundations Workshop (CSFW'01)*, pp. 220–234.
[26] G. Rosu and K. Havelund. Rewriting-based techniques for runtime verification. *Autom. Softw. Eng.*, 12(2):151–197, 2005.
[27] A. Sistla and O. Wolfson. Temporal triggers in active databases. *IEEE Trans. Knowl. Data Eng.*, 7(3):471–486, 1995.
[28] O. Sokolsky, U. Sammapun, I. Lee, and J. Kim. Run-time checking of dynamic properties. *Electr. Notes Theor. Comput. Sci.*, 144(4):91–108, 2006.
[29] P. Thati and G. Rosu. Monitoring algorithms for metric temporal logic specifications. *Electr. Notes Theor. Comput. Sci.*, 113:145–162, 2005.
[30] D. Toman. Logical data expiration. In *Logics for Emerging Applications of Databases*, pp. 203–238, 2003.

Foundations of Software Technology and Theoretical Computer Science (Bangalore) 2008.
Editors: R. Hariharan, M. Mukund, V. Vinay; pp 61-72

Solvency Games

N. Berger[1], N. Kapur[2], L. J. Schulman[2], V. V. Vazirani[3]

[1]Department of Mathematics
Hebrew University of Jerusalem, Israel
berger@math.ucla.edu

[2]Department of Computer Science
California Institute of Technology
{kapur, schulman}@caltech.edu

[3]College of Computing
Georgia Institute of Technology
vazirani@cc.gatech.edu

ABSTRACT. We study the decision theory of a maximally risk-averse investor — one whose objective, in the face of stochastic uncertainties, is to minimize the probability of ever going broke. With a view to developing the mathematical basics of such a theory, we start with a very simple model and obtain the following results: a characterization of best play by investors; an explanation of why poor and rich players may have different best strategies; an explanation of why expectation-maximization is not necessarily the best strategy even for rich players. For computation of optimal play, we show how to apply the Value Iteration method, and prove a bound on its convergence rate.

1 Introduction

A key concern in computer science and operations research is *decision-making under uncertainty*. We define a very simple game that helps us study the issue of solvency, or indefinite survival, in the presence of stochastic uncertainties. In Section 1.1 below we provide some motivating reasons for studying this issue.

We start by defining the model. A state of the game is an integer, which we call the wealth of the player. An *action* (representing, say, an investment choice) is a finitely supported probability distribution on the integers; this distribution specifies the probabilities with which various payoffs are received, if this action is chosen. Let w be the wealth of the player at time t. Let $\mathcal{A}$ be a set of actions. Suppose that after choosing a particular action from $\mathcal{A}$, the random variable sampled from that action is a. Then at time $t+1$ the wealth of the player is $w+a$. The game terminates if the player goes broke (wealth becomes ≤ 0). A *strategy* π for the set $\mathcal{A}$ of actions is a function $\pi : \mathbb{Z}_+ \to \mathcal{A}$ specifying the action that is chosen at each possible value of wealth. Corresponding to strategy π, define

$$p^\pi(w) = \mathbf{Pr}[\text{ever going broke, starting from wealth } w],$$

for each $w \in \mathbb{Z}_+$. The object of interest is a strategy that minimizes $p^\pi(w)$ for each value of $w \in \mathbb{Z}_+$. In this notation there are two implicit assumptions regarding an optimal strategy:

FSTTCS 2008
IARCS Annual Conference on
Foundations of Software Technology and Theoretical Computer Science
http://drops.dagstuhl.de/opus/volltexte/2008/1741

that the action depends only on current wealth (not past history), and that the action is deterministic. Both assumptions can be made without loss of generality.

This model, which is a certain kind of infinite-state Markov Decision Process (MDP), is a natural and elementary one to consider both from the point of view of probability theory, and that of mathematical finance. As far as we have been able to determine it has not previously been studied.

Before going into detail we pause for a simple illustration. Suppose two actions are available, called A and B; let q_i^{A} denote the probability of winning i dollars with action A:

$$\text{Action A:}\quad q^{\mathrm{A}}_{-1} = 0.5,\ \ q^{\mathrm{A}}_{15} = 0.5 \qquad\qquad \text{Action B:}\quad q^{\mathrm{B}}_{-10} = 0.5,\ \ q^{\mathrm{B}}_{150} = 0.5$$

Expected profit is ten times greater in action B, but it is easy to see that an investor with, say \$10, has probability of survival less than $1/2$ if he plays B, and close to 1 if he chooses and sticks to A. This illustrates how maximizing the likelihood of solvency can be quite different from maximizing expected profit. The problem, of course, is to determine proper strategy in less obvious situations.

1.1 Motivation

There are a couple of reasons to focus on maximization of the likelihood of indefinite survival. The first concerns investment strategies of individual, "middle class" investors. Economic decision theory concerns itself largely (though not solely) with maximization of utility as expressed by expected profit (or log profit). This framework may be appropriate to the decision theory of a shareholder-owned firm, whose bankruptcy creates an unpleasant but bounded effect on a balanced portfolio. But it is ill suited to the decision theory of an individual investor, whose goal is often not maximization of wealth for its own sake, but financial stability. For such a typical investor, bankruptcy, and its consequences for self and family, are dearly to be avoided.

The second reason concerns investment (loan) strategies of banks, which are unlike other corporations in that they are supposed to provide their depositors with a strong assurance of preservation of capital. The incompatibility between doing so and acting competitively in the loan marketplace has led to banking crises which have been addressed in part through government intervention including, in the USA, both federal deposit insurance and mandatory holding requirements. These restrict the extent to which banks can pursue purely profit-maximizing strategies (although we do not suggest that banks conversely act to maximize probability of indefinite solvency).

We return to the clash between optimizing for profit or survival. Naturally, a good way to avoid bankruptcy is to make a lot of money! But investment decisions entail a trade-off between risk and reward. The most secure investments typically provide returns below or only marginally above the inflation rate. So even a decision-maker whose sole purpose is to avoid bankruptcy cannot escape risk entirely, and must weigh the alternatives. The purpose of this paper is to develop some basic ingredients relevant to these decisions. In defining our model, simplicity is a key criterion. As a result, the model does not capture complications that always accompany realistic situations. On the other hand, this simplicity leads to clean mathematics and a basis from which more elaborate models can be considered.

1.2 Related work

As noted, the simple model defined above has apparently not been studied before. However, our motivation is very similar to that of previous authors, especially Ferguson [4], Truelove [11] and Browne [2]. The models are different enough to make the conclusions incomparable; some main differences are that in the previous work (a) The player has only one investment choice at any one time, and is simply deciding how much to invest, (b) That amount is unbounded except by the player's wealth. In some of the results, even the last restriction is dropped, and the player is permitted to borrow unlimited funds, sometimes with and sometimes without interest. Put simply, these authors' models are more general in allowing for investment scaling, and more special in not posing choices between dissimilar types of investments. The latter issue is the grist of our work.

An early book in the area, more relevant to Ferguson, Truelove and Browne's work than ours, is Dubins and Savage [3]. Slightly less related, but still relevant in terms of the motivation, is work in mathematical finance, in which risk (volatility) vs. reward is often measured with the Sharpe or Sterling ratios: see, e.g., [8]. Optimal investing by these criteria is less risk-averse than by ours. Shifting attention from the finance aspect to the decision theory, our work is more closely related to the large literature on the MDP model [10], a broad formalization of the study of decision-making under stochastic uncertainty. Specifically, the "multi-arm bandit" problem concerns maximizing profit from a collection of actions, where optimal play is characterized by the well-known Gittins index [6, 12]. Our problem does not seem to fit into this model.

Perhaps closest to our work is an interesting paper that first appeared in June 2007 [5]*. In this paper, Foster and Hart consider the question of measuring the *riskiness* of a gamble (their "gamble" has the same definition as our "action"). Surprisingly enough, they show that this can be boiled down to a single number – the *critical wealth*. If the wealth is strictly smaller than this number, it is risky to play the action, i.e., it will lead to bankruptcy with probability 1. Else, playing this action is guaranteed to not lead to bankruptcy, again with probability 1.

1.3 Results

The specific questions we address include:

1. In a set $\mathcal{A}$ of actions, is there a *rich man's strategy* — an investment that is always the best choice once one's wealth is above some threshold? Put another way, does the optimal strategy have a "pure tail"?

Besides its obvious role in the decision theory of our model, this question gets at a real phenomenon which we feel should be reflected in any good model of risk-averse investing: that the poor do disproportionately worse than the rich because they can not afford to make certain investments that are by-and-large profitable, yet risky.

2. If there is a "rich man's strategy," what characterizes it, and is there a bound on the threshold where it takes over? If there isn't one, then what *does* the tail of the optimal

*Most of our results date to 2004 and were presented at the 2004 AGATE Workshop on Algorithmic Game Theory in Bertinoro, Italy, and at a Plenary talk in RANDOM 2004.

strategy look like?

3. Can the optimal strategy be computed "efficiently"?

In Sections 3 and 4 we provide answers to these questions. We show that under certain technical conditions there does exist a rich man's strategy, and we provide a bound on where the pure tail begins. We also show that in general there is no such strategy — an interesting phenomenon, since it says that optimal play in a small-stakes game can depend, say, on the low-order bit of your bank balance. The MDP literature suggests three possible algorithms for computing the optimal strategy in the pure tail case (where this strategy has a finite description). For one of these algorithms, Value Iteration, we prove "linear convergence" (i.e., exponentially decreasing relative error) to the failure function of the optimal strategy.

1.4 Notation, terminology and structure of the paper

An *action* is represented by a probability mass function on a finite set of integers. For an action A, let q_j^{A} be the probability that the payoff is j. For an action A, define $l_{\mathrm{A}} := -\min\{j < 0 : \ q_j^{\mathrm{A}} > 0\}$ and $r_{\mathrm{A}} := \max\{j > 0 : \ q_j^{\mathrm{A}} > 0\}$. The action is said to have *positive drift* if $\sum_{j=-l_{\mathrm{A}}}^{r_{\mathrm{A}}} j q_j^{\mathrm{A}} > 0$. The action is said to be *irreducible* if $\gcd\left(\{j : q_j^{\mathrm{A}} > 0\}\right) = 1$. In this paper all actions will be assumed to be irreducible and positive drift, though some of our statements hold more generally.

A *strategy* (sometimes also referred to as *policy* or *decision rule*) is a function $\pi : \mathbb{Z}_+ \to \mathcal{A}$, where $\mathcal{A}$ is a set of actions. For a strategy π, we define the following Markov chain. $X_{t+1} = X_t + Y_t$ where Y_t is defined as follows: If $X_t \leq 0$ then $Y_t = 0$, whereas if $X_t > 0$ then Y_t is sampled according to $\pi(X_t)$, but otherwise independently of $X_0, \ldots, X_t$. The *failure probability* at a positive integer w (i.e, the probability of ever going broke) corresponding to π is defined as $p^{\pi}(w) := \mathbf{Pr}[\exists_{m>0} : X_m \leq 0 \mid X_1 = w]$. A strategy is said to be *pure* if $\pi(w) = \pi(1)$ for all $w \geq 1$. It is said to have a *pure tail* if there is a $w' \geq 1$ such that $\pi(w) = \pi(w')$ for all $w \geq w'$.

In Section 2 we develop the fairly simple theory of the behavior of the game under a pure strategy. Being basically a random walk, our results are mostly known. However, those results serve as necessary tools for the study of optimal strategy. In Section 3 we prove the main results of the paper - namely conditions for the existence of a rich man's strategy. In Section 4 we discuss algorithms for determining the optimal strategy. Missing proofs and the Appendix can be found in the full paper, available online at *ECCC* [1].

2 Pure strategies

Consider a pure strategy π^{A} consisting only of the action A with $l \equiv l_{\mathrm{A}}$ and $r \equiv r_{\mathrm{A}}$. Then, the failure probability $p(w) \equiv p^{\pi^{\mathrm{A}}}(w)$ satisfies the linear recurrence

$$p(w) = \sum_{j=-l}^{r} q_j^{\mathrm{A}} p(w+j), \quad w \geq 1, \tag{1}$$

where $q_j \equiv q_j^{\mathrm{A}}$ and with $p(w) = 1$ for all $w \leq 0$. The *characteristic rational function* of A is defined as

$$q(z) \equiv q^{\mathrm{A}}(z) := -1 + \sum_{j=-l}^{r} q_j z^j \tag{2}$$

LEMMA 1. *If $q'(1) > 0$ then q has exactly l roots in the open unit disk. Furthermore q has a unique positive root in the open unit disk.*

Note that the condition is equivalent to positive drift of the action.

PROOF. It suffices to consider instead the roots of the polynomial

$$A(z) := z^l q(z) := \sum_{\substack{j=0 \\ j \neq l}}^{l+r} q_{j-l} z^j - [1 - q_0] z^l.$$

Note that $A'(1) = q'(1) > 0$ and $A(1) = 0$ so that $A(1-) < 0$. Furthermore $A(0) = q_{-l} > 0$ so by continuity, A has a root in $(0, 1)$, call it ζ. For $\epsilon > 0$ define $A_\epsilon(z) := f_\epsilon(z) + h(z)$, where

$$f_\epsilon(z) := -(1+\epsilon)(1-q_0)z^l, \qquad h(z) := \sum_{\substack{j=0 \\ j \neq l}}^{l+r} q_{j-l} z^j.$$

Consider the circle $|z| = \zeta$. There,

$$|f_\epsilon(z)| = (1+\epsilon)(1-q_0)\zeta^l > (1-q_0)\zeta^l \quad \text{and} \quad |h(z)| \leq \sum_{\substack{j=0 \\ j \neq l}}^{l+r} q_{j-l}\zeta^j.$$

Since ζ is a zero of p, we have $|f_\epsilon(z)| > |h(z)|$ for all z with $|z| = \zeta$. Hence by Rouché's theorem (see, e.g., [9]), f_ϵ and A_ϵ have the same number of zeros inside $|z| = \zeta$. But f_ϵ has exactly l zeros inside $|z| = \zeta$, and hence so does A_ϵ. Similarly A_ϵ has exactly l zeros $|z| = 1$, so that there are no zeros of A_ϵ in $\zeta < |z| < 1$. Now letting $\epsilon \downarrow 0$ yields that A has exactly l zeros in the closed disk $|z| \leq \zeta$ and none in the annulus $\zeta < |z| < 1$ so that the first claim of the lemma follows.

For the second claim note that if ζ_1 and ζ_2 are distinct positive zeros of p, an argument similar to the one above yields that there are no zeros of p in the interval (ζ_1, ζ_2). The claim then follows by letting $\zeta_1 = \zeta$ and $\zeta_2 = 1$.

Remarks: The positive root of q in the open unit disk will be called the *Perron root*, for reasons explained in Appendix A in [1]. Since $q(1) = 0$, if $q'(1) < 0$ then there exits $z > 1$ such $q(z) < 0$. Also $q(z) > 0$ for large enough z. It follows then that q has a positive zero outside the closed unit disk and the proof of Lemma 1 reveals that this zero is unique.

Corollary 2 *If a pure strategy π^{A} has positive drift then its failure probabilities are*

$$p(w) \equiv p^{\pi^{\mathrm{A}}}(w) = \sum_{j=1}^{d} \lambda_j^w \sum_{k=0}^{m_j - 1} c_{j,k} w^k, \tag{3}$$

where $\lambda_1, \ldots, \lambda_d$ are the distinct zeros of q in the interior of the unit disk in decreasing order of norm, with multiplicities $m_1, \ldots, m_d$ such that $m_1 + \cdots + m_d = l$, and $(c_{j,k})$ are constants.

PROOF. Let λ be a zero of the characteristic rational function (2) with multiplicity m. Such a zero contributes a linear combination of $(w^j\lambda^w)_{j=0}^{m-1}$ to $p(w)$. Furthermore since we know a priori (see Fact 3) that $p(w) \to 0$ as $w \to \infty$, there cannot be any contribution from zeros with modulus at least 1. Since the pure strategy has positive drift, we have $(q^{\mathrm{A}})'(1) > 0$, so by Lemma 1, q^{A} has exactly l zeros in the unit disc and the result follows.

Remarks: Observe that the recurrence (1) defines a linear transformation mapping the initial conditions $p(w)_{w\le 0}$ monotonically to $p(w)_{w\ge 1}$. In particular, if λ is a zero of q, then $(\lambda^w)_{w\le 0}$ is mapped to $(\lambda^w)_{w\ge 1}$.

3 Optimal strategies

Let $\mathcal{A} = \{\mathrm{A}_1, \ldots, \mathrm{A}_k\}$ be a finite set of actions with positive drifts. We consider strategies $\pi : \mathbb{Z}_+ \to \mathcal{A}$. We start with a simple fact.

Fact 3 *For every strategy π, $p^\pi(w) \to 0$ as $w \to \infty$.*

PROOF. For $j = 1, \ldots, k$, let $\{Y_n^{(j)}\}_{n=1}^{\infty}$ be i.i.d. samples of A_j, and assume that for different values of j, the sequences $\{Y_n^{(j)}\}$ are independent. The displacement at any time n is of the form $\sum_{j=1}^{k}\sum_{i=1}^{n_j} Y_i^{(j)}$, where the $\{n_j\}$ sum to n and are (arbitrarily dependent) random variables. Fix ϵ. Due to the positive drifts, for all N large enough,

$$\mathbf{Pr}\left[\forall_{n,j} \sum_{i=1}^{n} Y_i^{(j)} > -N/k\right] > 1 - \epsilon.$$

But this shows that for all N large enough, $p^\pi(w) < \epsilon$.

For $w \ge 1$, an action A and a sequence p, we define

$$E_w^{\mathrm{A}}(p) := \sum_{j=-l_{\mathrm{A}}}^{r_{\mathrm{A}}} q_j^{\mathrm{A}} p(w+j). \tag{4}$$

For this to make sense, we need to have values for $p(w)$ for $k \le 0$. Unless otherwise mentioned, we take $p(w)$ to be 1 for all $w \le 0$. Similarly for a strategy π we define

$$E_w^{\pi}(p) := E_w^{\pi(w)}(p) \tag{5}$$

Clearly if p is the failure probability sequence of π, then

$$p(w) = E_w^{\pi}(p) \tag{6}$$

for every $w \ge 1$. Equation (6) determines p in the following sense:

LEMMA 4. *Fix a strategy π and initial conditions $b(w)$, $w \le 0$. There exists a unique solution to (6) satisfying $p(w) = b(w)$ for all $w \le 0$ and $\lim_{w\to\infty} p(w) = 0$.*

PROOF. This proof follows a conventional outline. Existence follows from Fact 3, Remark 2 and the fact that the probabilities satisfy (6). To see uniqueness, assume that p and q both satisfy the conditions. Then, $h = p - q$ also satisfies (6), $h(w) = 0$ for all $w \le 0$

and $lim_{w\to\infty}h(w) = 0$. Assume that there exists w' such that $h(w') \neq 0$. Without loss of generality, $h(w') > 0$. Since $h(w) \to 0$, there exists w_0 so that $h(w) < h(w')$ for all $w > w_0$. Therefore, $\max_w h(w) = \max\{h(w) : w \leq w_0\}$ and the maximum exists since it is taken over a finite set. Let H be this maximum, and let $\tilde{w} = \max\{w : h(w) = H\}$. By (6), $h(\tilde{w})$ is the average of numbers, all of which are no larger than H and some of which are strictly smaller than H. Therefore $h(\tilde{w}) < H$, in contradiction to its definition. Therefore, $h(w) \equiv 0$.

Definition 5 *We say that p is* harmonic *with respect to π if (6) holds for every $w \geq 1$. We say that p is* subharmonic *with respect to π if*

$$p(w) \leq E_w^\pi(p) \tag{7}$$

for every $w \geq 1$, and we say that p is superharmonic *with respect to π if*

$$p(w) \geq E_w^\pi(p) \quad \textit{for every} \quad w \geq 1. \tag{8}$$

The usefulness of Definition 5 is expressed in the following lemma:

LEMMA 6. *Let π be a strategy and p the unique solution to* (6) *with given initial conditions $b(w)$, $w \leq 0$. Let v be a sequence that satisfies the following conditions:*

1. *$v(w) = b(w)$ for all $w \leq 0$.*
2. *$\lim_{w\to\infty} v(w) = 0$.*
3. *v is subharmonic with respect to π.*

Then $v(w) \leq p(w)$ for every w. If instead v is superharmonic, then $v(w) \geq p(w)$ for every w.

3.1 Structure of optimal strategies

We can define a natural partial order between strategies: $\pi_1 \preceq \pi_2$ if for every w, $p^{\pi_1}(w) \leq p^{\pi_2}(w)$. We say that π^* is *optimal* if $\pi^* \preceq \pi$ for every strategy π. We say that σ is *locally optimal* if $\sigma \preceq \pi$ for every π satisfying $|\{w : \sigma(w) \neq \pi(w)\}| \leq 1$.

Proposition 7 *For every finite collection $\mathcal{A}$ of actions, there exists an optimal strategy. Furthermore, σ is optimal if and only if it is locally optimal.*

PROOF. We will start with the "furthermore" part: Let σ be locally optimal, and let π be another strategy. Let s be the failure probability sequence for σ, and let p be the failure probability sequence for π. By local optimality of σ, for every w, $E_w^\pi(s) \geq s(w)$. Therefore, s is subharmonic with respect to π, and by Lemma 6, $p(w) \geq s(w)$ for every w, i.e., $\sigma \preceq \pi$ and σ is optimal.

In order to prove the proposition, all we need is to find a locally optimal strategy. By compactness of the space of strategies (the product space of actions over all wealths), and continuity of

$$\sum_{w=1}^{\infty} p^\pi(w). \tag{9}$$

in this topology (using the positive-drifts assumption), there exits a strategy σ minimizes expression 9. We claim that σ is locally optimal. Indeed, let π be so that $|\{w : \sigma(w) \neq \pi(w)\}| = 1$, and let w be the unique index such that $\sigma(w) \neq \pi(w)$. Since σ and π disagree

at exactly one point, p^σ is either subharmonic or superharmonic with respect to π. It has to be subharmonic since p^σ minimizes (9), and therefore $\sigma \preceq \pi$ and σ is locally optimal.

For an action A, let $\lambda_{\mathrm{A}}^{(1)} > 0, \lambda_{\mathrm{A}}^{(2)}, \ldots, \lambda_{\mathrm{A}}^{(l_{\mathrm{A}})}$ be the roots of its characteristic rational function [recall (2)] in the open unit disk arranged in decreasing order of modulus.

We now present a characterization of optimal strategies. The next theorem exhibits the existence of a "rich man's strategy," as indicated in the introductory section.

THEOREM 8. *Let $\mathcal{A}$ be a finite set of actions and let $\mathrm{A} \in \mathcal{A}$ be an action so that $\lambda_{\mathrm{A}}^{(1)} < \lambda_{\mathrm{B}}^{(1)}$ for every $\mathrm{B} \neq \mathrm{A}$ in $\mathcal{A}$. Let π^* be optimal for $\mathcal{A}$. Then there exists M such that $\pi^*(w) = \mathrm{A}$ for every $w > M$.*

The existence of a "rich man's strategy" may seem natural, and if so, the imposition of technical hypotheses in Theorem 8 may seem disappointing. But this is not the case: strikingly, such conditions are necessary, as demonstrated in:

THEOREM 9. *Let $\mathcal{A} = \{\mathrm{A}, \mathrm{B}\}$ with $l_{\mathrm{A}} = l_{\mathrm{B}} = 2$, $\lambda_{\mathrm{A}}^{(1)} = \lambda_{\mathrm{B}}^{(1)}$, and $\lambda_{\mathrm{A}}^{(2)} \neq \lambda_{\mathrm{B}}^{(2)}$. If π^* is optimal for $\mathcal{A}$, then for every W there exist $w', w'' > W$ such that $\pi^*(w') = \mathrm{A}$ and $\pi^*(w'') = \mathrm{B}$.*

Remarks: Theorem 9 can be generalized to the case where l_g or l_f is greater than 2 under the assumption that the characteristic rational function of A has a root in the interior of the unit disk that is not shared by B and vice versa. The proof is omitted.

Proof of Theorem 8: For convenience of notation, let $\lambda := \lambda_{\mathrm{A}}^{(1)}$. Let π be a (fixed) strategy such that for every M there exists $w > M$ with $\pi(w) \neq \mathrm{A}$. We will show that π is not optimal. Let π^{A} be the pure-A strategy. Let $a(-w) = \lambda^{-w}$ and $p(-w) = 1$ for $w \geq 0$. Let a^π be the unique solution of $a(w) = E_w^\pi(a)$ with $a(w) \to 0$, and let $a^{\pi^{\mathrm{A}}}$ be the unique solution of $a(w) = E_w^{\pi^{\mathrm{A}}}(a)$ with $a(w) \to 0$. Let p^π and $p^{\pi^{\mathrm{A}}}$ be the failure probabilities for π and π^{A}.

It is sufficient to show that there exists w so that $p^{\pi^{\mathrm{A}}}(w) < p^\pi(w)$. Let l be the absolute value of the minimal number on the support of any of the actions in $\mathcal{A}$, i.e., $l := \max_{\mathrm{B} \in \mathcal{A}} l_{\mathrm{B}}$. Then by monotonicity (recall Remark 2), for every w,

$$a^\pi(w) \geq p^\pi(w) \geq \lambda^l a^\pi(w)$$

$$a^{\pi^{\mathrm{A}}}(w) \geq p^{\pi^{\mathrm{A}}}(w) \geq \lambda^l a^{\pi^{\mathrm{A}}}$$

Therefore it will suffice if we prove that there exist w so that

$$a^{\pi^{\mathrm{A}}}(w) < \lambda^l a^\pi(w). \tag{10}$$

In fact, we prove

$$\lim_{w \to \infty} \frac{a^\pi(w)}{a^{\pi^{\mathrm{A}}}(w)} = \infty. \tag{11}$$

To see (11), first note that (see Remark 2)

$$a^{\pi^{\mathrm{A}}}(w) = \lambda^w. \tag{12}$$

LEMMA 10. *$a^{\pi^{\mathrm{A}}}$ is subharmonic with respect to π.*

4 Algorithms for determining optimal strategies

We now turn our attention to the problem of determining the optimal strategy. To that end it will be useful to cast our problem in terms of *Markov decision processes* (MDPs). For background on MDPs, we refer the reader to the excellent book by Puterman [10]. Throughout, $\mathcal{A}$ is a finite set of *actions* with $l := \max\{l_{\mathrm{B}} : \mathrm{B} \in \mathcal{A}\}$ and $r := \max\{r_{\mathrm{B}} : \mathrm{B} \in \mathcal{A}\}$.

4.1 MDP formulations

For our purposes, a *Markov decision process* is a collection of objects $\{S, A_s, p(\cdot \mid s,a), r(s,a)\}$. Here S is a set of possible *states* the system can occupy. For each $s \in S$, the set of possible *actions* is denoted by A_s. The function $p(\cdot \mid s,a)$, called the *transition probability function* is a distribution on the set of states S and the *reward function* $r(s,a)$ is a real-valued function.

Under the assumptions of Theorem 8, we can modify our problem into an equivalent finite Markov decision problem, which makes determining an optimal strategy tractable. Let M be such that for some optimal strategy π^*, $\pi^*(w) = \mathrm{A}$ for every $w > M$. Here A is the action with the smallest Perron root. In Appendix B [1] we show how to explicitly bound M, with a method that extends the arguments of Theorem 8. To find an optimal strategy we need only consider strategies that have a pure-A tail starting at M. Let $S = \{-l+1, \ldots, M+r, \infty\}$. (The state ∞ represents the possibility of never returning to $\{-l+1, \ldots, M+r\}$.) The actions for $s \in \{1, \ldots, M\}$ are the original actions of the system. For $s \in \{M+1, \ldots, M+r\}$, the only action available is the action A′ with the following transition probability function:

$$p(j \mid s, \mathrm{A}') = \alpha^{\mathrm{A}}_{s-j}; \quad j = s-1, \ldots, s-l_{\mathrm{A}}; \quad p(\infty \mid s, \mathrm{A}') = 1 - \sum_{j=1}^{l_{\mathrm{A}}} \alpha^{\mathrm{A}}_j =: \alpha^{\mathrm{A}}_{\infty}.$$

Here the values $\{\alpha^{\mathrm{A}}_j\}$ are the coefficients of the linear functional giving p_w as a function of $p_{w-1}, \ldots, p_{w-l}$ in the pure A strategy; see Appendix A [1] for further details. The action set for the state ∞ as well as for any state in $\{-l+1, \ldots, 0\}$, consists only of the trivial action that leaves the state unchanged. The reward function is given by (13):

$$r(s,a) := -\sum_{j \in S} \mathbf{1}\{s > 0 \text{ and } j < 0\} p(j \mid s,a), \quad s \in S;\ a \in A_s. \tag{13}$$

Clearly the expected total reward is the negative of the failure probability.

Next, we present an algorithm that can be used to determine optimal decision rules.

4.2 Value iteration

An iterative procedure known as value iteration produces a sequence that converges to the optimal expected total reward for each $s \in S$. The critical thing of course will be the runtime analysis.

1. Set $v^0(s) = 0$ for each $s \in S$.
2. For each $s \in S$, compute $v^{n+1}(s)$ using

$$v^{n+1}(s) = \max_{a \in A_s} \left\{ r(s,a) + \sum_{j \in S} p(j \mid s,a) v^n(j) \right\}$$

and increment n.

The sequences converge monotonically to the optimal expected total reward v^* [10]. We show next that the order of convergence is linear.

To that end, let d^* denote an optimal decision rule and consider the sequence defined iteratively by $u^0(s) = 0$ for each $s \in S$ and

$$u^{n+1}(s) = r(s, d^*(s)) + \sum_{j \in S} p(j \mid s, d^*(s)) u^n(j). \tag{14}$$

This is just the sequence produced by value iteration when the only action available at a state is the optimal action. Clearly $u^n(s) \to v^*(s)$ and a simple induction argument yields $v^n(s) \geq u^n(s)$ for each $s \in S$ and $n \geq 0$.

Writing (14) in matrix notation we have $u^{n+1} = Pu^n + \alpha$, where $u^n, \alpha \in \mathbb{R}^{M+r}$ and $P \equiv P_{ij}$ is the $M + r \times M + r$ matrix with $P_{ij} := p(j \mid i, d^*(i))$.

LEMMA 11. *Let $P \equiv P(d^*)$ denote the transition matrix for an optimal decision rule d^*. Then, $\rho(P)$, the spectral radius of P is strictly less than 1.*

PROOF. If $\|\!|P|\!\|_\infty < 1$, then the claim is true. Suppose $\|\!|P|\!\|_\infty = 1$ so that $\rho(P) \leq 1$. Suppose $\rho(P) = 1$. Since P is nonnegative an eigenvalue of maximum modulus must be 1. Let $Px = x$, $x = [x_i] \neq 0$ and suppose p is an index such that $|x_p| = \|x\|_\infty \neq 0$. Now 1 lies on the boundary of $G(P)$, the Geršgorin region for the rows of P so that [7, Lemma 6.2.3(a)]

$$1 - P_{pp} = |1 - P_{pp}| = \sum_{\substack{j=1 \\ j \neq p}}^{M+r} P_{pj},$$

i.e., $\sum_{j=1}^{M+r} P_{pj} = 1$ so that $p \in \{1, \ldots, M\}$. Since P is the transition matrix for an optimal strategy there must be positive probability of reaching a state in $\{M+1, \ldots, M+r\}$ starting from the state p. In other words, there exist a sequence of distinct integers $k_1 = p, k_2, \ldots, k_m = q$ with $q \in \{M+1, \ldots, M+r\}$ such that all of the matrix entries $P_{k_1 k_2}, \ldots, P_{k_{m-1} k_m}$ are nonzero. But then [7, Lemma 6.2.3(b)], $|x_{k_i}| = |x_p|$ for each $i = 1, \ldots, m$. In particular $|x_q| = |x_p|$, so that again [7, Lemma 6.2.3(a)],

$$1 = |1 - P_{qq}| = \sum_{\substack{j=1 \\ j \neq q}}^{M+r} P_{qj} = \sum_{j=1}^{l} \alpha_j < 1, \qquad \text{which is a contradiction.}$$

Remarks: Using the fact that all actions have positive drift, we can estimate the spectral radius as follows. Let D be the diagonal matrix with entries $(\lambda + \epsilon, (\lambda + \epsilon)^2, \ldots, (\lambda + \epsilon)^{M+r})$, where λ is the largest Perron root among all roots of the characteristic rational functions of the actions and $\epsilon > 0$ is arbitrarily small. We show that $\|\!|D^{-1}PD|\!\|_\infty \leq \delta < 1$. Indeed for $i \in \{1, \ldots, M\}$, the ith row sum of $D^{-1}PD$ is given by

$$\sum_{j=1}^{M+r} P_{ij} (\lambda + \epsilon)^{j-i} \leq q^i (\lambda + \epsilon) + 1 := \delta_i, \tag{15}$$

where $q^i(\cdot)$ is the characteristic function of the action employed at state i. If λ_i is the *unique* positive root of q^i inside the unit disk, then $q^i(\lambda_i) = q^i(1) = 0$ and q^i has no zero crossing in $(\lambda_i, 1)$. Since i has positive drift we have $(q^i)'(1) > 0$ so that $q^i(z) < 0$ for $z \in (\lambda_i, 1)$. Hence the row-sum in (15) is bounded by $\delta_i < 1$.

On the other hand for $i \in \{M+1, \ldots, M+r\}$, the ith row-sum of $D^{-1}PD$ is given by $\sum_{j=1}^{l} \alpha_j^{\mathrm{A}}(\lambda+\epsilon)^{-j} := \delta_i < 1$, the last strict inequality following from the fact that if $\lambda_{\mathrm{A}} < \lambda + \epsilon$ is the Perron root of the pure-A tail, then

$$\sum_{j=1}^{l} \alpha_j^{\mathrm{A}} \lambda_{\mathrm{A}}^{-j} = 1$$

Taking $\delta := \max_{1 \le i \le M+r} \delta_i$ gives us $\rho(P) = \rho(D^{-1}PD) \le \|\!|D^{-1}PD|\!\|_\infty \le \delta < 1$.

The preceding lemma and remark lead directly to the following result.

THEOREM 12. *Let v^* denote the optimal total expected value and v^n the nth iterate of value iteration. Then $v^n \ge u^n$, where for some vector norm $\|\cdot\|$ and $n \ge 1$,*

$$\|v^* - u^n\| \le c\|v^* - u^{n-1}\|$$

and $c < 1$ satisfies

$$c \le \max\{1 + \max_{\text{action B}} q^{\mathrm{B}}(\lambda+\epsilon), \sum_{j=1}^{l} \alpha_j(\lambda+\epsilon)^{-j}\},$$

where λ is the largest of the Perron roots of the actions and $\epsilon > 0$ is arbitrarily small.

5 Discussion

In the MDP formulation, two other algorithms can be applied to computing the failure probabilities of the optimal strategy: policy iteration and linear programming. Their adaptation to our problem is discussed in Appendix C in [1].

It is clear that our results are at best a sketch of some elements of a larger theory. To begin with an equally well-motivated (and more general) model is one in which players are prohibited from taking actions that have nonzero probability of driving them immediately to a negative balance. (The player loses if no actions are available.) Our basic results carry over to this model. Another natural variant allows for the payoffs to be arbitrary real numbers. We have not explored this case.

It is natural to ask what happens if each available action can be scaled, at the player's discretion, by a positive constant. Allowing scaling by large constants is an interesting variant to study. (Allowing scaling by arbitrarily small constants trivializes the model: for any positive-drift action the probabilities of failure can be made to tend to 0. More importantly, it fails to match the motivating real-world scenarios. A bank deciding whether to issue a particular $\$200,000$ mortgage cannot change the associated risks by renaming it as $200,000$ separate \$1 mortgages.) Ideally in this context one would like to address a common extension of our model and those treated by Ferguson [4], Truelove [11] and Browne [2].

References

[1] N. Berger, N. Kapur, L. J. Schulman, and V. V. Vazirani. Solvency games. Available at ECCC (Electronic Colloquium on Computational Complexity) Report TR08-089, 2008.

[2] S. Browne. Optimal investment policies for a firm with a random risk process: exponential utility and minimizing the probability of ruin. *Math. Oper. Res.*, 20(4):937–958, 1995.

[3] L. E. Dubins and L. J. Savage. *How to gamble if you must. Inequalities for stochastic processes*. McGraw-Hill Book Co., New York, 1965.

[4] T. S. Ferguson. Betting systems which minimize the probability of ruin. *J. Soc. Indust. Appl. Math.*, 13:795–818, 1965.

[5] D. P. Foster and S. Hart. An operational measure of riskiness. Unpublished manuscript 2008. Available at:
http://www.ma.huji.ac.il/ hart/papers/risk.pdf.

[6] J. C. Gittins and D. M. Jones. A dynamic allocation index for the sequential design of experiments. In *Progress in statistics (European Meeting Statisticians, Budapest, 1972)*, pages 241–266. Colloq. Math. Soc. János Bolyai, Vol. 9. North-Holland, Amsterdam, 1974.

[7] R. A. Horn and C. R. Johnson. *Matrix analysis*. Cambridge University Press, Cambridge, 1990. Corrected reprint of the 1985 original.

[8] M. Magdon-Ismail, A. F. Atiya, A. Pratap, and Y. S. Abu-Mostafa. On the maximum drawdown of a Brownian motion. *J. Appl. Probab.*, 41(1):147–161, 2004.

[9] A. I. Markushevich. *Theory of functions of a complex variable. Vol. I, II, III*. Chelsea Publishing Co., New York, english edition, 1977. Translated and edited by Richard A. Silverman.

[10] M. L. Puterman. *Markov decision processes: discrete stochastic dynamic programming*. Wiley Series in Probability and Mathematical Statistics: Applied Probability and Statistics. John Wiley & Sons Inc., New York, 1994. , A Wiley-Interscience Publication.

[11] A. J. Truelove. Betting systems in favorable games. *Ann. Math. Statist.*, 41:551–556, 1970.

[12] J. N. Tsitsiklis. A short proof of the Gittins index theorem. *Ann. Appl. Probab.*, 4(1):194–199, 1994.

Foundations of Software Technology and Theoretical Computer Science (Bangalore) 2008.
Editors: R. Hariharan, M. Mukund, V. Vinay; pp 73-82

On the Power of Imperfect Information*

Dietmar Berwanger[1] **and Laurent Doyen**[2]

[1] RWTH Aachen, Germany

[2]EPFL Lausanne, Switzerland

ABSTRACT. We present a polynomial-time reduction from parity games with imperfect information to safety games with imperfect information. Similar reductions for games with perfect information typically increase the game size exponentially. Our construction avoids such a blow-up by using imperfect information to realise succinct counters which cover a range exponentially larger than their size. In particular, the reduction shows that the problem of solving imperfect-information games with safety conditions is EXPTIME-complete.

1 Introduction

Nondeterminism is a notorious source of complexity in automata. The process of determinisation, which consists in monitoring the uncertainty about the flow of control in a nondeterministic device, typically involves a power-set construction and an exponential blow-up of the state space. Reversing the argument, a nondeterministic automaton may be considerably more succinct than any equivalent deterministic automaton.

When we shift from automata to games, a similar jump in complexity arises as an effect of imperfect information of players about the history of a play. Already in the basic setting of two-player zero-sum games, the construction of a perfect-information game monitoring the uncertainty of a player about the flow of information in an imperfect-information game requires a powerset construction [13, 5].

The shape of winning conditions constitutes a further source of complexity in games. In particular in parity games, the range of the priority function is perceived as a key factor. For games with perfect information, the current situation is as follows. While games with two priorities can be solved in quadratic time, the complexity of the best known deterministic algorithms is exponential in the number of priorities. Several procedures for reducing the priorities in a parity game to a fixed small number have been proposed, all leading to an exponential blow-up in the size of the game [3, 9, 15]. A polynomial-time reduction of this kind would prove that parity games can be solved in polynomial time, which is a major open problem.

The question of parity-range reduction has also been investigated in the context of the modal μ-calculus, an expressive logic that subsumes many important specification formalisms. The model-checking problem for this logic corresponds to the problem of solving a parity game with as many priorities as there are alternations of fixed-point quantifiers in

*Research supported in part by the ESF programmes AutoMathA and GAMES, by the Swiss National Science Foundation, and by the European COMBEST project.

FSTTCS 2008
IARCS Annual Conference on
Foundations of Software Technology and Theoretical Computer Science
http://drops.dagstuhl.de/opus/volltexte/2008/1742

the formula [7]. A uniform method for reducing the number of quantifier alternations in formulae would thus lead to tractable model-checking games. For a particular fragment of distributive formulae, a reduction to formulae with only one alternation is presented in [12]. However, the fact that the μ-calculus alternation hierarchy is strict [11, 4, 1], implies that a uniform reduction, which depends only on the formula, cannot exist for the general case. In [15], Seidl proposes a reduction that removes fixed-point alternations syntactically in a non-uniform way, depending on the model, yielding one of the best algorithms for μ-calculus model checking, or equivalently, for solving parity games with perfect information.

In this paper, we consider parity games with imperfect information. We present a polynomial-time reduction of parity games into safety games that preserves the existence of winning strategies. This shows that, in the setting of imperfect information, parity games with only two priorities are able to simulate parity games with arbitrarily many priorities in a succinct way. In other words, the complexity arising from imperfect information preempts the complexity inherent to the winning condition.

The reduction implements a variant of the progress-measure algorithm for solving parity games proposed by Jurdzinski in [8]. We use the power of imperfect information in two ways: firstly, to design counters that cover a range exponentially larger than their size and, secondly, to maintain the number of occurrences of all odd priorities simultaneously during the play. The parity condition is monitored by synchronising the game graph with a small counter gadget equipped with a safety condition.

Our construction illustrates a basic design pattern for applying imperfect information as a synchronisation mechanism. Furthermore, the counting gadgets provide examples of safety games in which winning strategies require memory of exponential size. Finally, our reduction shows that the problem of solving imperfect-information games with safety conditions is EXPTIME-complete.

2 Parity games with perfect information

We first describe the model of parity games with perfect information and introduce the key properties needed for our reduction. In view of a uniform treatment of both perfect and imperfect information models, our terminology sometimes deviates from the standard literature.

2.1 Games and strategies

Let Σ be a finite alphabet of *actions*. A *game structure* with *perfect information* is a tuple $G = (L, \ell_0, \Delta)$ consisting of a finite set L of *locations* (or *positions*), a designated *initial location* $\ell_0 \in L$, and a *transition* relation $\Delta \subseteq L \times \Sigma \times L$. We assume that the transition relation is total, i.e., for every location $\ell \in L$ and every action $a \in \Sigma$, there exists at least one *a-successor* ℓ' such that $(\ell, a, \ell') \in \Delta$, and that all locations of L are reachable from ℓ_0 via transitions in Δ. Games on G are played by two players, Player 1 and Player 2, taking turns to move a token along transitions of G. Initially, the token is located at ℓ_0. The game proceeds in rounds. In every round, Player 1 first chooses an action $a \in \Sigma$, then Player 2 moves the token to an a-successor of the current location. Thus, playing the game yields an infinite

sequence of locations $\pi = \ell_1\ell_2\ldots$, called a *play*, such that $\ell_1 = \ell_0$ and $(\ell_i, a, \ell_{i+1}) \in \Delta$ for all $i \geq 1$. A *history* is a finite prefix $\ell_1 \ldots \ell_i$ of a play. A *strategy* for Player 1 in G is a function $\sigma : L^+ \to \Sigma$ that maps histories to actions. A play $\ell_1\ell_2\ldots$ is *consistent* with σ if, for every position $i \geq 1$, there is a transition $(\ell_i, a, \ell_{i+1}) \in \Delta$ with $a = \sigma(\ell_1, \ldots, \ell_i)$. We denote the set of plays in G that are consistent with σ by $\mathsf{Outcome}(G, \sigma)$.

A *winning condition* for a game structure G is a set $\varphi \subseteq L^\omega$. A strategy σ for Player 1 is *winning* for the condition φ, if all plays consistent with σ are winning, i.e., $\mathsf{Outcome}(G, \sigma) \subseteq \varphi$. A *game* is a pair (G, φ) consisting of a game structure G and a matching winning condition φ. We say that Player 1 *wins* the game, if he has a winning strategy for the condition φ.

We shall consider two kinds of winning conditions. Given a set $T \subseteq L$ of target locations, the *safety* condition requires that the play stay within the set T: $\mathsf{Safe}(T) = \{\ell_1\ell_2\ldots \mid \ell_i \in T \text{ for all } i \geq 1\}$. We call the elements of $L \setminus T$ *bad* locations. Given a *priority function* $\Omega : L \to \mathbb{N}$ that maps each location to a priority, the *parity* condition requires that the least priority visited infinitely often in a play be even: $\mathsf{Parity}(\Omega) = \{\ell_1\ell_2\ldots \mid \liminf_{i\to\infty} \Omega(\ell_i) \text{ is even}\}$. Parity conditions can be viewed as nested combinations of safety and reachability conditions, where reachability is the dual of safety: $\mathsf{Reach}(T) = L^\omega \setminus \mathsf{Safe}(L \setminus T)$. They provide a canonical form to express all ω-regular winning conditions [16].

The algorithmic problem of *solving* a game is to decide, given a game structure G and a winning condition φ, whether Player 1 wins the game (G, φ). Safety conditions are specified by a target set, and parity conditions are specified by a priority function.

A conceptually simple way of solving parity games is to provide a winning strategy for Player 1. For this purpose, strategies that depend only on the last location of the history of the play are of particular interest. A strategy σ is *memoryless* if $\sigma(\rho \cdot \ell) = \sigma(\rho' \cdot \ell)$ for all $\rho, \rho' \in L^*$. It is easy to see that, if Player 1 wins a game with safety or reachability condition, then he also has a memoryless strategy to win the game. The following fundamental result establishes that memoryless strategies are sufficient even for perfect-information games with parity conditions.

THEOREM 1. *[[6]] Player* 1 *wins a parity game with perfect information if and only if he has a memoryless winning strategy.*

A memoryless strategy σ for a game structure $G = (L, \ell_0, \Delta)$ can be represented as a substructure G_σ obtained by removing from Δ all transitions (ℓ, a, ℓ') with $a \neq \sigma(\ell)$. The plays in G_σ are then precisely the plays in $\mathsf{Outcome}(G, \sigma)$. Accordingly, for a parity condition φ, the strategy σ is winning if and only if all the plays in G_σ are winning, which amounts to saying that on each cycle in G_σ reachable from ℓ_0, the least visited priority is even. This remark provides the key argument for the transformation of parity games into safety games.

DEFINITION 2. Let $n \in \mathbb{N}$. An infinite sequence $p_1p_2\ldots$ of natural numbers is *parity-n-fair* if, for every odd number r, each subsequence $p_ip_{i+1}\ldots p_j$ that contains the number r more than n times also contains a number strictly smaller than r.

For a fixed game $(G, \mathsf{Parity}(\Omega))$, we say that a play π is parity-n-fair if the sequence of priorities visited by π is parity-n-fair. Notice that every parity-n-fair play satisfies the parity condition. Conversely, if G has n locations, then all plays consistent with a memoryless

winning strategy σ of Player 1 in G are parity-n-fair. Indeed, every subsequence of a play consistent with σ that contains more than n occurrences of an odd priority r must follow a cycle in G_σ. As the least priority in every cycle of G_σ is even, every such subsequence also contains a priority smaller than r. According to Theorem 1, we can hence restrict our attention, without loss of generality, to strategies that enforce parity-n-fair plays.

PROPOSITION 3. *Player* 1 *wins a parity game with perfect information of size n if and only if he has a strategy σ such that every play consistent with σ is parity-n-fair.*

Let us now turn to the computational complexity of solving a parity game with perfect information G. A memoryless strategy σ for Player 1 can be guessed in linear time and we can verify in polynomial time whether σ is winning, i.e., whether the minimal priorities on all reachable cycles in G_σ are even. Thus, the problem of solving a game belongs to NP and, by the Determinacy Theorem of [6], it follows that it is in NP $\cap$ Co-NP. Hence the problem is close to polynomial time, in terms of general complexity (see also [8]). The question whether parity games can be solved in polynomial time is a major open problem. The best known deterministic algorithms have running times that are polynomial with respect to the size of the game structure, but exponential with respect to the number of different priorities (see [10, 14]).

2.2 Priority-range reduction

Due to the apparent impact of the number of priorities on the complexity of solving parity games, it would be very desirable to find efficient procedures for reducing the range of the priority function.

An explicit reduction from parity to safety games with perfect information is presented by Bernet, Janin, and Walukiewicz in [3]; it can be understood as an online-version of Jurdzinski's progress-measure algorithm for solving parity games [9]. The main ingredient of the reduction is an internal memory device consisting of a vector of counters, one for each odd priority which is maintained along the transitions of a play. Basically, the device works as follows: if the current state has priority r, all counters corresponding to priorities strictly higher than r are reset; additionally, if r is odd, the counter corresponding to r is incremented. The range of each counter is bounded by the number of locations in the game. To transform a parity game into a safety game, the memory device is synchronised with the game structure via a product operation. Finally, the safety condition requires that no counter overflow occur. Essentially, the internal memory monitors whether the current play is parity-n-fair and forces the play into a bad location when it detects that this is not the case. The correctness of this reduction is justified by arguments similar to Proposition 3. Notice, however, that to monitor a game with n states and d priorities, a memory device with $O(n^{d/2})$ many states is needed. Accordingly, this reduction from parity to safety games involves an exponential blow-up of the game structure.

3 Games with Imperfect Information

We consider a model of games with imperfect information that was originally introduced in [13]. The set of locations is partitioned into information sets indexed by *observations*.

3.1 Observation-based model

In addition to the alphabet Σ of actions, we fix a finite alphabet Γ of observations. A *game structure with imperfect information* over Σ and Γ is a tuple $G = (L, \ell_0, \Delta, \gamma)$, where L, ℓ_0, Δ are defined as in the perfect-information case, and $\gamma : \Gamma \to 2^L \setminus \emptyset$ is an *observability* function that maps each observation to a nonempty set of locations such that the sets $\gamma(o)$ for $o \in \Gamma$ form a partition of L. For each location $\ell \in L$, we write $\mathsf{obs}(\ell)$ to denote the unique observation o such that $\ell \in \gamma(o)$. For an action $a \in \Sigma$ and a set of locations $s \subseteq L$, we define $\mathsf{post}_a(s) = \{\ell' \in L \mid \exists \ell \in s : (\ell, a, \ell') \in \Delta\}$.

The game on G is played in the same way as in the perfect information case, by moving a token along the transitions of G and forming an infinite play. But now, only the observation of the current location is revealed to Player 1. The effect of the uncertainty about the history of the play is formally captured by the notion of strategy.

A *strategy* for Player 1 in G is a function $\sigma : \Gamma^+ \to \Sigma$ that maps finite sequences of observations to actions. Given a play $\pi = \ell_1 \ell_2 \ldots$, we set $\mathsf{obs}(\pi) = \mathsf{obs}(\ell_1)\mathsf{obs}(\ell_2)\ldots$. We say that π is *consistent* with the strategy σ, if for every position $i \geq 1$, there is a transition $(\ell_i, a, \ell_{i+1}) \in \Delta$ with $a = \sigma(\mathsf{obs}(\ell_1)\ldots\mathsf{obs}(\ell_i))$. As before, we denote the set of plays in G that are consistent with σ by $\mathsf{Outcome}(G, \sigma)$.

Following [5], we express winning conditions in terms of observations. A winning condition for a game structure $G = (L, \ell_0, \Delta, \gamma)$ is a set $\varphi \subseteq \Gamma^\omega$ of infinite sequences of observations. A strategy σ for Player 1 is *winning* for the condition φ if $\mathsf{obs}(\pi) \in \varphi$ for all $\pi \in \mathsf{Outcome}(G, \sigma)$. The safety condition for a set $T \subseteq \Gamma$ is $\mathsf{Safe}(T) = \{o_1 o_2 \ldots \mid o_i \in T \text{ for all } i \geq 1\}$, and the parity condition for a priority function $\Omega : \Gamma \to \mathbb{N}$ is defined by $\mathsf{Parity}(\Omega) = \{o_1 o_2 \ldots \mid \liminf_{i\to\infty} p(o_i) \text{ is even}\}$.

Notice that games of perfect information correspond to the special case where $\Gamma = L$ and $\gamma(\ell) = \{\ell\}$ for all $\ell \in L$.

3.2 Reduction to perfect-information games

To solve a game with imperfect information (G, φ) over a structure $G = (L, \ell_0, \Delta, \gamma)$, the basic algorithm proposed in [13] constructs a game of perfect information $(G^{\mathsf{K}}, \varphi')$ over a game structure $G^{\mathsf{K}} = (S, s_0, \Delta')$ with the action alphabet Σ of G, such that Player 1 has a winning strategy for φ in G if and only if he has a winning strategy for φ' in G^{K}. The structure G^{K} is obtained by a *subset construction* which, intuitively, monitors the knowledge that Player 1 has about the current location of the play. The set of locations $S \subseteq 2^L \setminus \{\emptyset\}$ consists of the subsets of L reachable from the initial location $s_0 = \{\ell_0\}$ via transitions in Δ' defined by $(s_1, a, s_2) \in \Delta'$ if and only if there exists an observation $o \in \Gamma$ such that $s_2 = \mathsf{post}_a(s_1) \cap \gamma(o) \neq \emptyset$. Notice that each location in G^{K} corresponds to a unique observation in G, in the sense that for all $s \in S$, there is a unique $o \in \Gamma$ such that $s \subseteq \gamma(o)$. A bijection μ between strategies σ in G and strategies σ^{K} in G^{K} that preserves winning strategies is defined as follows. For all strategies σ in G, set $\mu(\sigma) = \sigma^{\mathsf{K}}$ such that $\sigma^{\mathsf{K}}(s_1 \ldots s_n) = \sigma(o_1 \ldots o_n)$ for all sequences $s_1 \ldots s_n$ of locations in G^{K}, where $o_1 \ldots o_n$ is the unique sequence of observations $o_1 \ldots o_n$ corresponding to $s_1 \ldots s_n$. Conversely, given a strategy σ^{K} in G^{K}, the strategy $\sigma = \mu^{-1}(\sigma^{\mathsf{K}})$ is such that for all $o_1 \ldots o_n \in \Gamma^+$, we have $\sigma(o_1 \ldots o_n) = \sigma^{\mathsf{K}}(s_1 \ldots s_n)$ where $s_1 = s_0$ and $s_{i+1} = \mathsf{post}_{a_i}(s_i) \cap \gamma(o_{i+1})$ with $a_i = \sigma^{\mathsf{K}}(s_1 \ldots s_i)$ for all $1 \leq i < n$. Observe that

the plays consistent with σ in G visit the same sequences of priorities as the plays consistent with $\sigma^{\mathsf{K}} = \mu(\sigma)$ in G^{K}.

The construction transforms games with imperfect information into games of perfect information with the same type of winning condition [5]. For a parity condition φ defined by the priority function $\Omega : \Gamma \to \mathbb{N}$, the parity condition φ' is defined by the priority function $\Omega' : S \to \mathbb{N}$ such that $\Omega'(s) = \Omega(o)$ for all $s \in S$ and $o \in \Gamma$ such that $s \subseteq o$.

PROPOSITION 4. *[[5]] Player* 1 *wins a game with imperfect information* $(G, \mathsf{Parity}(\Omega))$ *if and only if he wins the game with perfect information* $(G^{\mathsf{K}}, \mathsf{Parity}(\Omega'))$.

4 Reduction of parity to safety games

To present our reduction from parity to safety games, let us fix a parity game with imperfect information $(G, \mathsf{Parity}(\Omega))$ with n locations and with priorities ranging from 1 to d; we set $[d] = \{1, 2, \ldots, d\}$. Without loss of generality, we assume that d is even.

The game structure G^{K} obtained by the subset construction of Section 3.2 has less than 2^n locations. According to Proposition 3, we can require that a winning strategy of Player 1 in $(G^{\mathsf{K}}, \mathsf{Parity}(\Omega))$ (and thus also in $(G, \mathsf{Parity}(\Omega))$) ensures that no odd priority is visited more than 2^n times between two consecutive occurrences of lower priorities. This is a safety condition that can be checked by counting the occurrences of each odd priority in the play. If the count exceeds 2^n while no lower priority is visited, a bad location is entered.

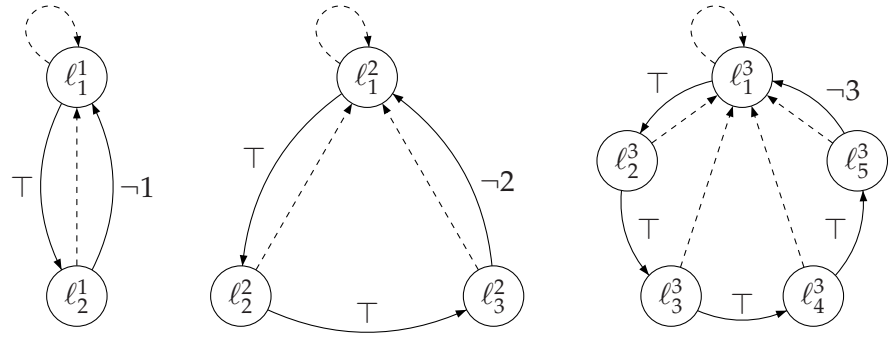
The challenge is to design counters with a bound of at least 2^n and to maintain simultaneously $d/2$ such counters, one for each odd priority, using only a polynomial number of locations.

We use a *counter gadget* to store the number of occurrences of an odd priority r. Whenever a priority smaller than r is visited, the counter is reset. For each visit to priority r, Player 1 has to increment the counter via a *click* action that he can choose from the set $[n] = \{1, 2, \ldots, n\}$. The gadgets are constructed in such a way that in each step at least one click can increment the counter, until the upper bound is reached. When this happens, *all* clicks would lead to the bad location.

To each odd priority, we associate a counter gadget. In the first round of the game, Player 2 can choose a counter associated to one particular odd priority r to be tracked. This choice is not observable to Player 1. Thus, Player 1 has to ensure that *every* odd priority occurs only a bounded number of times before a lower priority is visited. This translates the parity condition (that the minimal priority seen infinitely often is even) into a safety condition (that no counter ever overflows).

4.1 Succinct counters

For each odd priority r, the counter gadget C_r is a game structure consisting of n disjoint components (numbered $1, 2, \ldots, n$), one for each click. Figure 1 shows a counter gadget with 3 components. The i-th component has the shape of a loop over q_i locations, where q_i is the i-th prime number. The locations of C_r are all indistinguishable to Player 1. Therefore, we may think of a virtual game played simultaneously on all components, as if there was a token moving in each component of the gadget. The number of configurations of the tokens



Increment (solid edges)	on priority $p = r$, with any click except i on edges $\ell^i_{q_i} \to \ell^i_1$.
Reset (dashed edges)	on all priorities $p < r$.
Idle (not depicted)	on all priorities $p > r$ (self-loops).

Figure 1: Counter gadget for priority r with 3 components that counts modulo $2 \cdot 3 \cdot 5 = 30$.

in a counter is given by the primorial $q_n\# = \prod_{i=1}^{n} q_i$. Clearly, we have $q_n\# > 2^n$ whereas the number of locations in a counter is $\sum_{i=1}^{n} q_i = O(n^2 \log n)$ and thus polynomially bounded in n (cf. [2]).

The value of a counter is encoded by the position of the (virtual) tokens in each of its components. A counter can be incremented by taking, simultaneously in all components, a transition represented by a solid edge in Figure 1, it can be reset to 0 with the dashed edges, and it can idle with self-loops on each location (not drawn in the figure). The transitions of C_r are labeled by all actions $(a, p, k) \in \Sigma \times [d] \times [n]$ such that $p > r$ on all idle edges, $p < r$ on all reset edges, and $p = r$ on all increment edges, except the last edge of each component where the click k must be different from the number of the component (in Figure 1, the label $\top$ is interpreted as "for all clicks" and $\neg k$ is interpreted as "for all clicks except k", for $k \in \mathbb{N}$). Finally, we complete the transition relation, by sending all missing transitions to a sink location. Intuitively, whenever a counter is incremented, the value of the click k should be chosen (by Player 1) in such a way that every component has an enabled increment transition labeled with k, i.e., such that q_k does not divide the incremented counter value. This is always possible except when, in *all* components, the token is in the last location before completing the cycle. In the example of Figure 1, this happens after $2 \cdot 3 \cdot 5 - 1 = 29$ steps. From that moment on, Player 1 should avoid visiting priority r unless the counter is reset by a visit to a lower priority.

LEMMA 5. *Let $C_1, C_3, \ldots, C_{d-1}$ be counter gadgets, each with n components. A sequence $p_1 p_2 \ldots$ of priorities $p_i \in [d]$ is parity-$(q_n\#)$-fair if and only if there exist sequences $a_1 a_2 \ldots$ of actions and $k_1 k_2 \ldots$ of clicks such that $(a_1, p_1, k_1)(a_2, p_2, k_2) \ldots$ is a play in each of the components of $C_1, \ldots, C_{d-1}$.*

4.2 Reduction

For the parity game with imperfect information $(G, \mathsf{Parity}(\Omega))$ over alphabets Σ and Γ, we construct in polynomial time a safety game with imperfect information $(G, \mathsf{Safe}(T))$ over

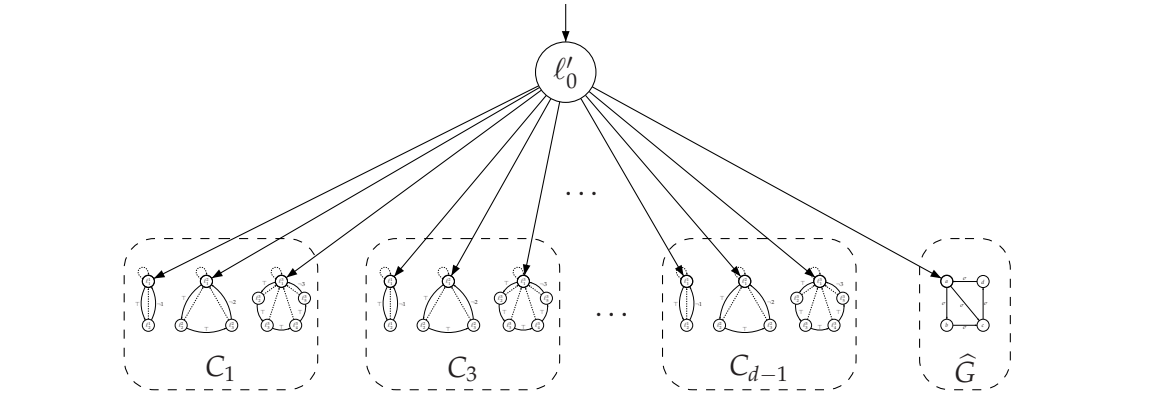


Figure 2: Reduction overview.

extended alphabets Σ' and Γ' such that Player 1 wins $(G, \mathsf{Parity}(\Omega))$ if and only if he wins $(G', \mathsf{Safe}(T))$. The set T contains all locations of G' except a designated sink location. The game structure G' consists of an initial location ℓ'_0 from which there is an outgoing transition to the initial location of each of the n components of each counter gadget $C_1, \ldots, C_{d-1}$ and to the initial location of a modified copy $\widehat{G}$ of G, as in Figure 2.

The game structure $\widehat{G}$ enriches the set Σ of actions to synchronise with the counter gadgets. The locations of $\widehat{G}$ are those of G and a fresh location with odd priority. For each transition (ℓ, a, ℓ') in G, there are transitions $(\ell, (a, p, k), \ell')$ for $p = \Omega(\mathsf{obs}(\ell))$ and for all $1 \leq k \leq n$. Hence, the set of actions of $\widehat{G}$ is $\Sigma' = \Sigma \times [d] \times [n]$. We complete the transition relation of $\widehat{G}$ by sending all missing transitions to the fresh location from which Player 1 cannot win. The game $\widehat{G}$ is equivalent to G, as the strategies of Player 1 in G have access to the observation of the current location (and therefore also to its priority) and can thus be translated into equivalent strategies for $\widehat{G}$ by simply choosing the priority $p = \Omega(\mathsf{obs}(\ell))$ of the current location ℓ for the second component of the indicated action (the third component is intended for synchronisation with the clicks and does not matter in $\widehat{G}$).

The observations in G' are the same as in G, that is, $\Gamma' = \Gamma$. However, the observability function γ' of G' is defined for all $o \in \Gamma$ by $\gamma'(o) = \gamma(o) \cup L_C$ where L_C is the set of all locations of the counter gadgets. This defines overlapping observations, but we can construct in polynomial time an equivalent safety game with partitioning observations (*cf.* [5, page 7]).

PROPOSITION 6. *The problem of solving a parity game with imperfect information can be reduced in polynomial time to the problem of solving a safety game with imperfect information.*

Proof. We show that Player 1 wins the game $(G, \mathsf{Parity}(\Omega))$ if and only if he wins the game $(G', \mathsf{Safe}(T))$.

First, let us assume that Player 1 wins $(G', \mathsf{Safe}(T))$ and let us fix a winning strategy σ' in G'. We construct a strategy σ in G such that for all $\rho \in \Gamma^+$, we have $\sigma(\rho) = a$ if $\sigma'(\rho) = (a, p, k)$ for a priority p and a click k. Now we claim that σ is winning in G. To show this, we argue that for all odd priorities r, if r occurs infinitely often in a path π of G_σ, then a smaller priority $p < r$ also occurs infinitely often in π. Towards a contradiction, assume that an odd priority r occurs infinitely often on a path π of G_σ, whereas all priorities lower than r

occur only finitely often. In particular, this implies that π is not a parity-q_n#-fair path. By Lemma 5 it follows that σ', which agrees with σ on the first component (on actions $a \in \Sigma$), cannot avoid an overflow of the counter C_r leading the play to the sink state. Hence, σ' is not a winning strategy in G'.

For the converse, assume that Player 1 wins $(G, \mathsf{Parity}(\Omega))$. Then, there exists a winning strategy σ for Player 1 in G ensuring that every path in G_σ is parity-2^n-fair, by Proposition 3 and via the bijection μ between strategies of G and G^{K} defined in Section 3.2. Therefore, each path of G_σ, can visit at most $2^n < q_n\#$ times an odd priority r without visiting a smaller priority. Hence, each counter C_r is reset before reaching the maximal value $q_n\#$. The winning strategy σ can therefore be extended to a winning strategy in G' by prescribing (a, r, k) whenever σ prescribes a, where r is the priority of the current observation, and k is a click allowed in the corresponding counter C_r (i.e., such that q_i is not a divisor of the number of visits to priority r since the last visit to a smaller priority). In this way, the sink location of the counters is never reached in G' and thus, the strategy σ is winning. ■

If we view the counter gadgets as individual games, we obtain a family of examples of safety games with an exponential lower bound for the memory size of a winning strategy.

COROLLARY 7. *There exists a family $(G_n, \mathsf{Safe}(T_n))_{n\in\mathbb{N}}$ of safety games with imperfect information where Player 1 wins, such that each game G_n is of size polynomial in n, whereas every winning strategy in G_n requires memory of size at least exponential in n.*

The problem of solving reachability and parity games of imperfect information is known to be EXPTIME-complete [13, 5]. However, the question about a matching lower bound for the complexity of safety games remained open. We can now settle this question as a direct consequence of Proposition 6.

COROLLARY 8. *The problem of solving safety games with imperfect information is* EXPTIME*-complete.*

References

[1] A. ARNOLD, *The μ-calculus alternation-depth is strict on binary trees*, Inf. Théorique et Applications, 33 (1999), pp. 329–339.

[2] E. BACH AND J. SHALLIT, *Algorithmic Number Theory, Vol. 1: Efficient Algorithms*, MIT Press, 1996.

[3] J. BERNET, D. JANIN, AND I. WALUKIEWICZ, *Permissive strategies: from parity games to safety games*, Inf. Théorique et Applications, 36 (2002), pp. 261–275.

[4] J. BRADFIELD, *The modal μ-calculus alternation hierarchy is strict*, Theoretical Computer Science, 195 (1998), pp. 133–153.

[5] K. CHATTERJEE, L. DOYEN, T. A. HENZINGER, AND J.-F. RASKIN, *Algorithms for omega-regular games of incomplete information*, Logical Methods in Computer Science, 3 (2007).

[6] E. A. EMERSON AND C. S. JUTLA, *Tree automata, mu-calculus and determinacy*, in Proc. of FoCS 1991, IEEE, 1991, pp. 368–377.

[7] E. GRÄDEL, W. THOMAS, AND T. WILKE, eds., *Automata, Logics, and Infinite Games*, LNCS 2500, Springer-Verlag, 2002.
[8] M. JURDZIŃSKI, *Deciding the winner in parity games is in UP ∩ co-UP*, Information Processing Letters, 68 (1998), pp. 119–124.
[9] M. JURDZIŃSKI, *Small progress measures for solving parity games*, in Proc. of STACS: Theor. Aspects of Comp. Sc., LNCS 1770, Springer, 2000, pp. 290–301.
[10] M. JURDZIŃSKI, M. PATERSON, AND U. ZWICK, *A deterministic subexponential algorithm for solving parity games*, in Proc. of SODA: Symp. on Discrete Algorithms, ACM Press, 2006, pp. 117–123.
[11] G. LENZI, *A hierarchy theorem for the mu-calculus*, in Proc. of ICALP: Automata, Languages and Programming, LNCS 1099, Springer, 1996, pp. 87–97.
[12] D. NIWINSKI AND H. SEIDL, *On distributive fixed-point expressions*, Inf. Théorique et Applications, 33 (1999), pp. 427–446.
[13] J. REIF, *The complexity of two-player games of incomplete information*, Journal of Computer and System Sciences, 29 (1984), pp. 274–301.
[14] S. SCHEWE, *Solving parity games in big steps*, in Proc. of FSTTCS: Foundations of Software Tech. and Theor. Comp. Sc., LNCS 4855, Springer, 2007, pp. 449–460.
[15] H. SEIDL, *Fast and simple nested fixpoints*, Information Processing Letters, 59 (1996), pp. 303–308.
[16] W. THOMAS, *Languages, automata, and logic*, Handbook of Formal Languages, 3 (1997), pp. 389–455.

Foundations of Software Technology and Theoretical Computer Science (Bangalore) 2008.
Editors: R. Hariharan, M. Mukund, V. Vinay; pp 83-94

Boolean algebras of unambiguous context-free languages

Didier Caucal
Institut Gaspard Monge, CNRS – Université Paris-Est
caucal@univ-mlv.fr

ABSTRACT. Several recent works have studied subfamilies of deterministic context-free languages with good closure properties, for instance the families of input-driven or visibly pushdown languages, or more generally families of languages accepted by pushdown automata whose stack height can be uniquely determined by the input word read so far. These ideas can be described as a notion of synchronization. In this paper we present an extension of synchronization to all context-free languages using graph grammars. This generalization allows one to define boolean algebras of non-deterministic but unambiguous context-free languages containing regular languages.

1 Introduction

Several restrictions of pushdown automata were recently studied in order to define classes of languages which generalize regular languages while retaining some of their closure properties, namely closure under boolean operations, concatenation and its iteration. All of these approaches consist in defining a notion of synchronization between pushdown automata [AM 04, Ca 06, NS 07] (see also [LMM 08] for complexity results). An approach which also avoids a special treatment of the ε-moves, is to define the synchronization at graph level [CH 08]. More precisely, the transition graph of any pushdown automaton A can be generated by a (deterministic graph) grammar R [MS 85, Ca 07] using infinite parallel rewritings. The stack height of a configuration of A is replaced by its weight, which is the minimal number of steps of parallel rewriting by R necessary to produce it.

The notion of synchronization can be defined for all graph grammars. A grammar G is synchronized by a grammar H if for any accepting path λ of (the graph generated by) G, there exists an accepting path μ of H with the same label u such that λ and μ are synchronized: for every prefix v of u, the prefixes of λ and μ labelled by v lead to vertices of the same weight. By extending usual constructions from finite automata to grammars generating deterministic graphs, we have shown that the languages recognized by all grammars synchronized with a given grammar form a boolean algebra lying between regular languages and deterministic context-free languages [CH 08].

In this paper, we apply the notion of synchronization to graph grammars recognizing unambiguous context-free languages, which are the languages generated by context-free grammars with at most one derivation tree for each word. Although these languages form a natural generalization of deterministic context-free languages, their equivalence problem remains a challenge in formal language theory [Gi 66]. Recent developments can be found in [Wi 04]. We present two classes of graph grammars, called unambiguous and level-unambiguous, recognizing all unambiguous context-free languages. A grammar is unambiguous if two accepting paths in the generated graph have distinct labels. More

FSTTCS 2008
IARCS Annual Conference on
Foundations of Software Technology and Theoretical Computer Science
http://drops.dagstuhl.de/opus/volltexte/2008/1743

generally, a grammar is level-unambiguous if two accepting paths with the same label are synchronized. We show that the languages recognized by grammars synchronized with a fixed level-unambiguous grammar form a boolean algebra containing the regular languages (where the complement operation is relative to the language of the synchronizing grammar). A direct consequence is the decidability of the inclusion problem between languages recognized by two level-unambiguous grammars synchronized by a third one.

The paper is structured as follows: after recalling some notations and definitions in Sections 2 and 3, we present the notion of synchronization of arbitrary grammars in Section 4. We then focus on the closure properties of level-unambiguous grammars in Section 5.

2 Notations

Let $\mathbb{N}$ be the set of natural numbers. For a set E, we write $|E|$ its cardinality, 2^E its powerset and for every $n \geq 0$, $E^n = \{(e_1, \ldots, e_n) \mid e_1, \ldots, e_n \in E\}$ is the set of n-tuples of elements of E. Thus $E^* = \bigcup_{n\geq 0} E^n$ is the free monoid generated by E for the *concatenation*: $(e_1, \ldots, e_m)\cdot(e'_1, \ldots, e'_n) = (e_1, \ldots, e_m, e'_1, \ldots, e'_n)$, whose neutral element is the 0-tuple $()$. A finite set E of symbols is an *alphabet* of *letters*, and E^* is the set of *words* over E. Any word $u \in E^n$ is of *length* $|u| = n$ and is also represented by a mapping from $[n] = \{1, \ldots, n\}$ into E, or by the juxtaposition of its letters: $u = u(1)\ldots u(|u|)$. The neutral element is the word of length 0 called the *empty word* and denoted by ε. We denote by $[0, n] = \{0, \ldots, n\}$ for any $n \in \mathbb{N}$. For any binary relation R, we also write xRy for $(x, y) \in R$; as usual $Dom(R) = \{x \mid \exists y, xRy\}$ and $Im(R) = \{y \mid \exists x, xRy\}$ are the *domain* and the *image* of R.

Let F be a set of symbols called *labels*, ranked by a mapping $\varrho : F \longrightarrow \mathbb{N}$ associating to each label f its *arity* $\varrho(f) \geq 0$, and such that $F_n := \{f \in F \mid \varrho(f) = n\}$ is countable for every $n \geq 0$. We consider simple, oriented and labelled hypergraphs: a *hypergraph* G is a subset of $\bigcup_{n\geq 0} F_n V^n$, where V is an arbitrary set, such that

- its *vertex* set $V_G := \{v \in V \mid FV^*vV^* \cap G \neq \emptyset\}$ is finite or countable,
- its *label* set $F_G := \{f \in F \mid fV^* \cap G \neq \emptyset\}$ is finite.

Any $fv_1\ldots v_{\varrho(f)} \in G$ is a *hyperarc* labelled by f and of successive vertices $v_1, \ldots, v_{\varrho(f)}$; it is depicted according to the arity of f as follows:

- for $\varrho(f) \geq 2$, as an arrow labelled f and successively linking $v_1, \ldots, v_{\varrho(f)}$;
- for $\varrho(f) = 1$, as a label f on vertex v_1 (f is called a *colour* of v_1);
- for $\varrho(f) = 0$, as an isolated label f called a *constant*.

This is illustrated in the next figures. Note that a vertex v is depicted by a dot named (v) where parentheses are used to differentiate a vertex name from a vertex label (a colour).

For a subset $E \subseteq F$ of labels, we write $V_{G,E} := \{v \in V \mid EV^*vV^* \cap G \neq \emptyset\} = V_{G\cap EV_G^*}$ the set of vertices of G linked by a hyperarc labelled in E. A *graph* G is a hypergraph whose labels are only of arity 1 or 2: $F_G \subset F_1 \cup F_2$. Hence a graph G is a set of *arcs* av_1v_2 identified with the labelled transition $v_1 \xrightarrow[G]{a} v_2$ or directly $v_1 \xrightarrow{a} v_2$ if G is understood, plus a set of coloured vertices fv.

A tuple $(v_0, a_1, v_1, \ldots, a_n, v_n)$ with $n \geq 0$ and $v_0 \xrightarrow[G]{a_1} v_1 \ldots v_{n-1} \xrightarrow[G]{a_n} v_n$ is called a *path* from v_0 to v_n labelled by $u = a_1 \ldots a_n$; we write $v_0 \xRightarrow[G]{u} v_n$ or directly $v_0 \xRightarrow{u} v_n$ if G is understood. For $P, Q \subseteq V_G$ and $u \in F_2^*$, we write $P \xRightarrow[G]{u} Q$ if $p \xRightarrow[G]{u} q$ for some $p \in P$ and $q \in Q$ and

$L(G,P,Q) := \{u \mid P \xRightarrow[G]{u} Q\}$ is the language recognized by G from P to Q. In these notations, we can replace P (and/or Q) by a colour $\circ$ to designate the subset $V_{G,\circ}$. In particular $\circ \xRightarrow[G]{u} Q$ means that there is a path labelled by u from a vertex coloured by $\circ$ to a vertex in Q, and $L(G,\circ,\bullet)$ is the label set of the paths from a vertex coloured by $\circ$ to a vertex coloured by $\bullet$.

In this paper, we use two colours $\circ, \bullet \in F_1$ to mark respectively the initial vertices and the final vertices. To depict an initial or final vertex, the dot is replaced by its colour, and $\circledcirc$ represents a vertex which is initial and final. For any graph G, we denote by $L(G) := L(G,\circ,\bullet)$ the *language recognized* by G. Recall that the *regular languages* over an alphabet $T \subset F_2$ form the set $Reg(T^*) := \{L(G) \mid G \text{ finite } \wedge\, F_G \subseteq T \cup \{\circ,\bullet\}\}$.

3 Graph grammars

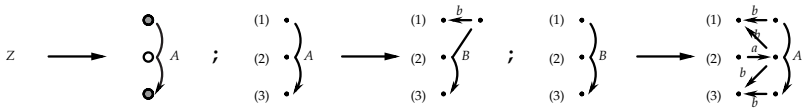
In this section, we recall the definition of deterministic graph grammars, together with the family of graphs they generate (called regular graphs). Using initial and final vertices, they can be viewed as infinite automata, generalizing finite automata. We also define two restricted classes of grammars recognizing all unambiguous context-free languages.

A graph *grammar* R is a finite set of rules of the form $fx_1 \ldots x_{\varrho(f)} \longrightarrow H$ where $fx_1 \ldots x_{\varrho(f)}$ is a hyperarc joining pairwise distinct vertices $x_1 \neq \ldots \neq x_{\varrho(f)}$ and H is a finite hypergraph; we denote by $N_R := \{f \in F \mid \exists x_1, \ldots, x_{\varrho(f)},\ fx_1 \ldots x_{\varrho(f)} \in Dom(R)\}$ the *non-terminals* of R (the labels of the left hand sides), by $T_R := \{f \in F - N_R \mid \exists H \in Im(R), V_{H,f} \neq \emptyset\}$ the *terminals* of R (the labels of R which are not non-terminals), and by $F_R := N_R \cup T_R$ the labels of R. We use grammars to generate graphs. Hence in the following, we assume that any terminal is of arity 1 or 2: $T_R \subset F_1 \cup F_2$.

Like a context-free grammar (on words), a graph grammar has an axiom, which is an initial finite hypergraph. To specify this axiom, we assume that any grammar R has a constant non-terminal $Z \in N_R \cap F_0$ which does not appear in any right hand side; the *axiom* of R is the right hand side H of the rule corresponding to Z: $Z \longrightarrow H \wedge Z \notin F_K$ for any $K \in Im(R)$.

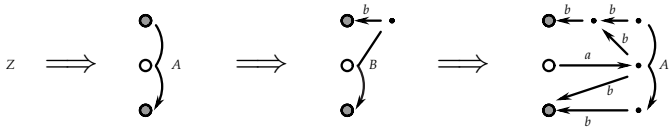
Starting from the axiom, we want R to generate a unique graph up to isomorphism. So we finally assume that any grammar R is *deterministic*, meaning that there is only one rule per non-terminal: $(X,H),(Y,K) \in R \wedge X(1) = Y(1) \Longrightarrow (X,H) = (Y,K)$. For any rule $X \longrightarrow H$, we say that $V_X \cap V_H$ are the *inputs* of H and $\bigcup\{V_Y \mid Y \in H \wedge Y(1) \in N_R\}$ are the *outputs* of H. For convenience and without loss of generality, it is simpler to assume that any grammar R is *terminal-outside* [Ca 07], meaning that there should be at least one non-input vertex in the support of any terminal arc or colour in a right hand side: $H \cap (T_R V_X V_X \cup T_R V_X) = \emptyset$ for any rule $(X,H) \in R$. We use upper-case letters $A, B, C, \ldots$ to denote non-terminals and lower-case letters $a, b, c \ldots$ for terminals.

The next figure shows an example of a (deterministic graph) grammar *Double* with non-terminals Z, A, B, terminals $a, b, \circ, \bullet$ and rule inputs $1, 2, 3$ (except for the axiom rule which has no input).

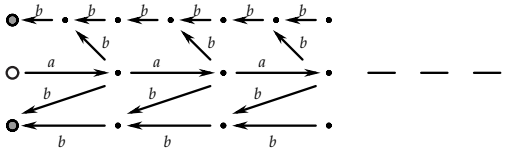


Given a grammar R, the *rewriting* $\underset{R}{\longrightarrow}$ is the binary relation between hypergraphs defined as follows: M rewrites into N, written $M \underset{R}{\longrightarrow} N$, if we can choose a non-terminal hyperarc $X = As_1 \dots s_p$ in M and a rule $Ax_1 \dots x_p \longrightarrow H$ in R such that N can be obtained by replacing X by H in M: $N = (M - X) \cup h(H)$ for some function h mapping each x_i to s_i, and the other vertices of H injectively to vertices outside of M; this rewriting is denoted by $M \underset{R,X}{\longrightarrow} N$. The rewriting $\underset{R,X}{\longrightarrow}$ of a hyperarc X is extended in an obvious way to the rewriting $\underset{R,E}{\longrightarrow}$ of any set E of non-terminal hyperarcs.

The *complete parallel rewriting* $\underset{R}{\Longrightarrow}$ is a simultaneous rewriting according to the set of all non-terminal hyperarcs: $M \underset{R}{\Longrightarrow} N$ if $M \underset{R,E}{\longrightarrow} N$ where E is the set of all non-terminal hyperarcs of M. We depict below the first three steps of the parallel derivation of the previous grammar *Double* from its constant non-terminal Z:



Given a deterministic grammar R and a hypergraph H, we denote by $[H] := H \cap T_R V_H^* = H \cap (T_R V_H V_H \cup T_R V_H)$ the set of terminal arcs and of terminal coloured vertices of H. A graph G is *generated* by R (from its axiom) if G belongs to the set of isomorphic graphs $R^\omega := \{\bigcup_{n \geq 0} [H_n] \mid Z \underset{R}{\longrightarrow} H_0 \underset{R}{\Longrightarrow} \dots H_n \underset{R}{\Longrightarrow} H_{n+1} \dots\}$. For instance by indefinitely iterating the previous derivation, we get the following infinite graph:



We call *regular* a graph generated by a (deterministic graph) grammar. Given a (regular) graph $G = \bigcup_{n \geq 0} [H_n]$ generated by a grammar R, with $Z \underset{R}{\longrightarrow} H_0 \underset{R}{\Longrightarrow} \dots H_n \underset{R}{\Longrightarrow} H_{n+1} \dots$, we define the *level* $\ell(s)$ of a vertex $s \in V_G$, denoted also $\ell_G^R(s)$ to specify G and R, as the minimal number of rewritings applied from the axiom to obtain s: $\ell(s) := \min\{n \mid s \in V_{H_n}\}$. The previous graph is represented by vertices of increasing level: vertices of the same level are vertically aligned for clarity. For any grammar R and for $G \in R^\omega$, we denote by $L(R) := L(G)$ the *language recognized* by R, which is well-defined since all graphs generated by a grammar are isomorphic. For instance, the grammar *Double* above recognizes the language $L(Double) = \{a^n b^n \mid n > 0\} \cup \{a^n b^{2n} \mid n > 0\}$.

A graph G is *deterministic* if $\circ$ colours a unique vertex, and two arcs with the same source have distinct labels: $r \underset{G}{\overset{a}{\longrightarrow}} s \wedge r \underset{G}{\overset{a}{\longrightarrow}} t \implies s = t$. Deterministic graph grammars recognize the family of context-free languages. The restriction to grammars generating a deterministic graph yields the family of deterministic context-free languages [Ca 07]. A grammar R is *unambiguous* if any pair of accepting paths have distinct labels: for $G \in R^\omega$,

$$s_0 \underset{G}{\overset{a_1}{\longrightarrow}} s_1 \dots \underset{G}{\overset{a_n}{\longrightarrow}} s_n \ \wedge \ t_0 \underset{G}{\overset{a_1}{\longrightarrow}} t_1 \dots \underset{G}{\overset{a_n}{\longrightarrow}} t_n \ \wedge \ \circ s_0, \circ t_0, \bullet s_n, \bullet t_n \in G \implies s_i = t_i \ \forall i \in [0, n].$$

Note that the previous grammar is unambiguous. Any grammar generating a deterministic graph is unambiguous. However, unambiguous grammars recognize strictly more languages than deterministic ones.

PROPOSITION 1. *Unambiguous grammars recognize the family of unambiguous context-free languages.*

Recall that there exist context-free languages which are not unambiguous *i.e.* which cannot be generated by an unambiguous context-free grammar; they are called inherently *ambiguous* context-free languages. An example of an ambiguous context-free language is $\{a^m b^m a^n b^n \mid m,n \geq 0\} \cup \{a^m b^n a^n b^m \mid m,n \geq 0\}$.

The synchronization relation we will soon define requires a slight generalization of unambiguous grammars. A grammar R is called *level-unambiguous* if for any pair of accepting paths λ, μ with the same label u and for every prefix v of u, the prefixes of λ and μ labelled by v lead to vertices of the same level. Formally, for (any) $G \in R^\omega$,

$$s_0 \xrightarrow[G]{a_1} s_1 \ldots \xrightarrow[G]{a_n} s_n \ \wedge\ t_0 \xrightarrow[G]{a_1} t_1 \ldots \xrightarrow[G]{a_n} t_n \ \wedge\ \circ s_0, \circ t_0, \bullet s_n, \bullet t_n \in G \implies \ell^R_G(s_i) = \ell^R_G(t_i)\ \ \forall i \in [0,n].$$

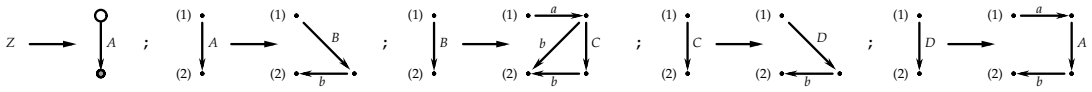
Note that any unambiguous grammar is also level-unambiguous. One can prove (Cf. Lemmas 12 and 13) that even though they are slightly more general, level-unambiguous grammars do not recognize more languages than unambiguous ones.

PROPOSITION 2. *Level-unambiguous grammars recognize the family of unambiguous context-free languages.*

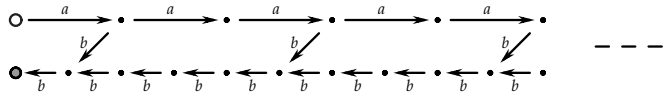
4 Synchronization of grammars

The notion of synchronization was defined in earlier work as a binary relation between grammars generating deterministic graphs [CH 08]. In this section, we extend it to all grammars. To each grammar R, we associate the family $\mathrm{Sync}(R)$ of languages recognized by grammars synchronized by R. We give closure properties of $\mathrm{Sync}(R)$ and show that this family is independent of the way to generate R^ω.

A grammar R *synchronizes* a grammar S, and we write $R \rhd S$ or $S \lhd R$ if for (any) $G \in R^\omega$ and (any) $H \in S^\omega$, whenever there exists a path $t_0 \xrightarrow[H]{a_1} t_1 \ldots \xrightarrow[H]{a_n} t_n$ with $\circ t_0, \bullet t_n \in H$, then there exists $s_0 \xrightarrow[G]{a_1} s_1 \ldots \xrightarrow[G]{a_n} s_n$ with $\circ s_0, \bullet s_n \in G$ and $\ell^R_G(s_i) = \ell^S_H(t_i)\ \forall i \in [0,n]$, meaning that for any accepting path μ labelled by u in the graph generated by S, there must be an accepting path λ label by u in the graph generated by R such that for every prefix v of u, the prefixes of λ and μ labelled by v lead to vertices of the same level.
For instance the grammar *Double* of the previous section synchronizes the following grammar S:



whose generated graph is represented by vertices of increasing level as follows:



and whose accepted language is $L(S) = \{a^{2n+1}b^{4n+2} \mid n \geq 0\}$.

Note in particular that $S \lhd R \implies L(S) \subseteq L(R)$. The relation $\rhd$ is reflexive and transitive but not antisymmetric. We denote by $\bowtie$ the *bi-synchronization* relation: $R \bowtie S$ if $R \rhd S$ and $S \rhd R$. The following lemma states that level-unambiguity is preserved for synchronized grammars.

LEMMA 3. *For any level-unambiguous grammar R:*
a) $S \lhd R \implies$ *S is level-unambiguous;*
b) $S \bowtie R \iff S \lhd R$ *and* $L(S) = L(R)$.

A useful transformation preserving bi-synchronization is to restrict to vertices accessible from $\circ$ and co-accessible from $\bullet$. The *restriction* $G_{|P}$ of a graph G to a subset $P \subseteq V_G$ of vertices is the subgraph of G induced by P:

$$G_{|P} := \{s \xrightarrow{a} t \mid s \xrightarrow[G]{a} t \wedge s, t \in P\} \cup \{cs \mid cs \in G \wedge s \in P\}.$$

We write $R^\omega_{\circ,\bullet} := \{G_{|\{s \mid \circ \underset{G}{\Longrightarrow} s \underset{G}{\Longrightarrow} \bullet\}} \mid G \in R^\omega\}$ the restriction of R^ω by accessibility from $\circ$ and co-accessibility from $\bullet$. We can restrict synchronization to grammars generating graphs accessible from their initial vertices and co-accessible from their final vertices.

LEMMA 4. *Any grammar R can be transformed into a grammar S such that $S \bowtie R$ and $S^\omega = R^\omega_{\circ,\bullet}$.*

Another basic transformation, given in Lemma 6.1 of [Ca 07] allows us to restrict ourselves to grammars with colours $\circ$ and $\bullet$ only in the axiom (i.e. whose generated graph only contains initial and final vertices at level 0). We say that a grammar R is *initial* when this is the case, *i.e.* when $(X, H) \in R \wedge X \neq Z \implies V_{H,\circ} = \emptyset = V_{H,\bullet}$.

This transformation works as follows. Let R be any grammar. We consider two arity 2 new symbols $i, f \in F_2$ such that $i, f \notin F_R$ and i, f are not vertices of R. To any non-terminal $A \in N_R - \{Z\}$, we associate a new symbol $A_{i,f}$ of arity $\varrho(A) + 2$. We consider the grammar:

$$\begin{aligned}[R, i, f] \;:=\; & \{(Z, H_{i,f} \cup \{\circ i, \bullet f\}) \mid (Z, H) \in R\} \cup \{(A_{i,f} X i f, H_{i,f}) \mid (AX, H) \in R \wedge A \neq Z\} \\ \text{where} \quad H_{i,f} \;:=\; & ([H] - \{\circ, \bullet\}V_H) \cup \{A_{i,f} X i f \mid AX \in H \wedge A \in N_R\} \\ & \cup \{i \xrightarrow{i} s \mid \circ s \in H\} \cup \{s \xrightarrow{f} f \mid \bullet s \in H\}.\end{aligned}$$

This grammar $[R, i, f]$ is an initial grammar such that, for any $G \in R^\omega$ with $i, f \notin V_G$, $G_{i,f} \cup \{\circ i, \bullet f\} \in [R, i, f]^\omega$. In particular $L([R, i, f]) = iL(R)f$. Moreover,

$S \lhd R \iff [S, i, f] \lhd [R, i, f]$ and $[R, i, f]$ is (level-)unambiguous if and only if R is.

Note that if R^ω has an infinite number of initial (resp. final) vertices then the initial (resp. final) vertex of $[R, i, f]^\omega$ is of infinite out-degree (resp. in-degree).

To any grammar R, we associate a family of *synchronized languages*

$$\mathrm{Sync}(R) \;:=\; \{L(S) \mid S \lhd R\}$$

which are thelanguages accepted by the grammars synchronized by R. Observe in particular that $R \bowtie S \implies \mathrm{Sync}(R) = \mathrm{Sync}(S)$, and $\mathrm{Sync}([R,i,f]) = \{iLf \mid L \in \mathrm{Sync}(R)\}$.

For any alphabet $T \subset F_2$, all the regular languages in T^* can be synchronized by the grammar Reg defined as the unique axiom rule $Z \longrightarrow \{0 \xrightarrow{a} 0 \mid a \in T\} \cup \{\circ 0, \bullet 0\}$ (in other words, $\mathrm{Sync}(\mathrm{Reg}) = Reg(T^*)$). Also note that any grammar R synchronizes any grammar without colour $\circ$ or $\bullet$, thus $\emptyset \in \mathrm{Sync}(R)$. Let us generalize this fact.

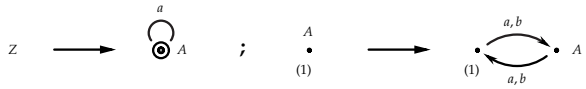
PROPOSITION 5. *For any grammar R, the family $\mathrm{Sync}(R)$ is closed under union, and contains $L(R) \cap M$ for any regular language M.*

PROOF. Closure under union will not be detailed here, but is straightforward. Containment of all regular languages inside $L(R)$ is done by synchronization product of R with a finite automaton K [CH 08]. Let $\{q_1, \ldots, q_n\} = V_K$ be the vertex set of K. To each $A \in N_R$, we associate a new symbol A' of arity $n \times \varrho(A)$, and to each hyperarc $Ar_1 \ldots r_m$ with $m = \varrho(A)$, we associate the hyperarc $(Ar_1 \ldots r_m)' := A'(r_1,q_1) \ldots (r_1,q_n) \ldots (r_m,q_1) \ldots (r_m,q_n)$. As an exception, we assimilate Z' to Z. We then define the grammar $R \times K$, which associates to each rule $(X,H) \in R$ the rule:

$$X' \longrightarrow \{(s,p) \xrightarrow{a} (t,q) \mid s \xrightarrow[H]{a} t \wedge p \xrightarrow[K]{a} q\} \cup \{(BU)' \mid BU \in H \wedge B \in N_R\}$$
$$\cup \{\circ(s,p) \mid \circ s \in H \wedge \circ p \in K\} \cup \{\bullet(s,p) \mid \bullet s \in H \wedge \bullet p \in K\}.$$

It is easily shown that $R \times K \lhd R$ and $L(R \times K) = L(R) \cap L(K)$. ∎

For any grammar R, the family $\mathrm{Sync}(R)$ is in general not closed under intersection, hence not closed under complement with respect to $L(R)$, since $L \cap M = L(R) - [(L(R) - L) \cup (L(R) - M)]$ for any $L, M \subseteq L(R)$. For instance the following grammar:

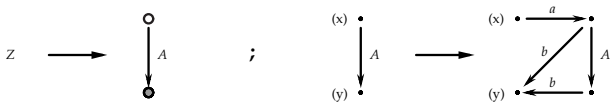


is not level-unambiguous, and for $L = \{a^m b^m a^n \mid m,n \geq 0\}$ and $M = \{a^m b^n a^n \mid m,n \geq 0\}$, we have $L, M \in \mathrm{Sync}(R)$ but $L \cap M = \{a^n b^n a^n \mid n \geq 0\} \notin \mathrm{Sync}(R)$.

For R^ω deterministic, $\mathrm{Sync}(R)$ coincides with the family of synchronized languages defined in [CH 08].

PROPOSITION 6. *For any grammar R such that R^ω is deterministic,*
$$\mathrm{Sync}(R) = \{L(S) \mid S \lhd R \wedge S^\omega \text{ deterministic}\}.$$

As a corollary of Proposition 6, $\mathrm{Sync}(R)$ is a boolean algebra when R^ω is deterministic [CH 08]. For instance, let *Single* be the following grammar:



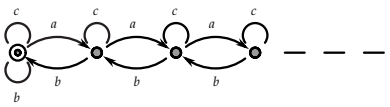
We have $L(Single) = \{a^n b^n \mid n > 0\}$ and $\mathrm{Sync}(Single) = \{L(G_{m,n}, I) \mid m \geq 0 \wedge n > 0\}$, where $L(G_{m,n}, I)$ is the language generated from I by the linear context-free grammar $G_{m,n}$:

$$I = P + a^m A b^m \quad \text{with} \quad P \subseteq \{ab, \ldots, a^m b^m\}$$
$$A = Q + a^n A b^n \quad \text{with} \quad Q \subseteq \{ab, \ldots, a^n b^n\}.$$

We conclude this section with a fundamental result concerning grammar synchronization, which states that $\mathrm{Sync}(R)$ is independent of the way the graph R^ω is generated.

THEOREM 7. *For any grammars R and S, $R^\omega = S^\omega \implies \mathrm{Sync}(R) = \mathrm{Sync}(S)$.*

This theorem allows to transfer the concept of grammar synchronization to the level of graphs: for any regular graph G, we can define $\mathrm{Sync}(G)$ as $\mathrm{Sync}(R)$ for any grammar R generating G. For instance, the following regular graph:



defines by synchronization the family of visibly pushdown languages (with a pushing, b popping and c internal) [AM 04].

5 Synchronization of level-unambiguous grammars

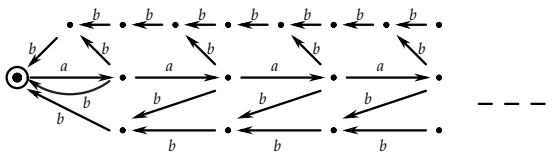
As previously stated, for any grammar R generating a deterministic graph, $\mathrm{Sync}(R)$ is an effective boolean algebra. In this section, we show that this remains true when R is level-unambiguous.

THEOREM 8. *For any level-unambiguous grammar R, the family $\mathrm{Sync}(R)$ is an effective boolean algebra with respect to $L(R)$, containing all the regular languages included in $L(R)$.*

For instance, let us consider the initial and unambiguous grammar *Double* of Section 3. We have $\mathrm{Sync}(Double) = \{L(G_{m,n}, I) \cup L(H_{m',n'}, I) \mid m, m' \geq 0 \wedge n, n' > 0\}$ where $G_{m,n}$ is defined above and $H_{m,n}$ is the following linear context-free grammar:

$$I = P + a^m A b^{2m} \quad \text{with} \quad P \subseteq \{abb, \ldots, a^m b^{2m}\}$$
$$A = Q + a^n A b^{2n} \quad \text{with} \quad Q \subseteq \{abb, \ldots, a^n b^{2n}\}.$$

This is indeed a boolean algebra with respect to $L(Double)$. Finally for the regular graph G



the family $\mathrm{Sync}(G)$ is the regular closure of $\mathrm{Sync}(Double)$.

A particular consequence of Theorem 8 is that we can decide the inclusion $L(S) \subseteq L(S')$ for two grammars S and S' synchronized by a common level-unambiguous grammar. Recall that the inclusion problem is undecidable for the so-called *simple* languages [Fr 77].

The constructions from [CH 08] cannot be trivially extended because level-unambiguity is a global property of accepted words and not a local property like graph determinism. However we can still work locally thanks to the notions of synchronization and level-unambiguity, which both only require to work level by level.

Closure under union was already stated in Proposition 5. We now proceed to prove the closures under intersection (Lemma 9) and complement (Lemma 14).

5.1 Closure under intersection

We will use other colours in addition to $\circ$ and $\bullet$. For any set of colours $C \subseteq F_1 - \{\circ\}$ and any grammar R, we denote R_C the grammar obtained from R by colouring every C-coloured vertex with $\bullet$ and removing $\bullet$ on all other vertices:

$$R_C := \{\big(X, (H - \{\bullet\}V_H) \cup \{\bullet p \mid \exists c \in C, cp \in H\}\big) \mid (X,H) \in R\}.$$

We define a level-preserving version of the grammar synchronization product. Let $\bullet_1, \bullet_2$ be new colours. Let R and S be two grammars, $G \in R^\omega$ and $H \in S^\omega$ two graphs they generate, and let $W := \{(s,p) \in V_G \times V_H \mid \ell^R_G(s) = \ell^S_H(p)\}$, the *level synchronization product* $G \times H$ is

$$\begin{aligned} G \times H := \{(s,p) \xrightarrow{a} (t,q) \mid s \xrightarrow[G]{a} t \wedge p \xrightarrow[H]{a} q \wedge (s,p),(t,q) \in W\} \\ \cup \{\circ(s,p) \mid \circ s \in G \wedge \circ p \in H \wedge (s,p) \in W\} \cup \{\bullet(s,p) \mid \bullet s \in G \wedge \bullet p \in H \wedge (s,p) \in W\} \\ \cup \{\bullet_1(s,p) \mid \bullet s \in G \wedge \bullet p \notin H \wedge (s,p) \in W\} \cup \{\bullet_2(s,p) \mid \bullet s \notin G \wedge \bullet p \in H \wedge (s,p) \in W\}. \end{aligned}$$

We then simply define $R^\omega \times S^\omega$ as $\{G \times H \mid G \in R^\omega \wedge H \in S^\omega\}$. The standard synchronization product of two regular graphs can be non regular, but the level synchronization product $R^\omega \times S^\omega$ can be generated by a grammar $R \times S$ that we define.

Let $(A,B) \in N_R \times N_S$ be any pair of non-terminals, we consider binary relations E over inputs such that $\forall i,j \in [\varrho(A)], E(i) \cap E(j) \neq \emptyset \implies E(i) = E(j)$, where $E(i) = \{j \mid (i,j) \in E\}$ denotes the *image* of $i \in [\varrho(A)]$. To any such A, B and E, we associate a new symbol $[A,B,E]$ of arity $|E|$ (where $[Z,Z,\emptyset]$ is assimilated to Z). To each non-terminal hyperarc $Ar_1 \ldots r_m$ of R ($A \in N_R$ and $m = \varrho(A)$) and each non-terminal hyperarc $Bs_1 \ldots s_n$ of S ($B \in N_S$ and $n = \varrho(B)$), we associate the hyperarc $[Ar_1 \ldots r_m, Bs_1 \ldots s_n, E] := [A,B,E](r_1,s_1)_E \ldots (r_1,s_n)_E \ldots (r_m,s_1)_E \ldots (r_m,s_n)_E$ with $(r_i,s_j)_E := (r_i,s_j)$ if $(i,j) \in E$, and ε otherwise. The grammar $R \times S$ is then defined as the set of rules

$$[AX, BY, E] \longrightarrow \big([P] \times [Q]\big)_{|\overline{E}} \cup \{[CU, DV, E'] \mid CU \in P \wedge C \in N_R \wedge DV \in Q \wedge D \in N_S\}$$

for each $(AX,P) \in R$, each $(BY,Q) \in S$, and each $E \subseteq [\varrho(A)] \times [\varrho(B)]$ with

$$\begin{aligned} \overline{E} &:= \{(X(i), Y(j)) \mid (i,j) \in E\} \cup \big(V_P - V_X\big) \times \big(V_Q - V_Y\big) \\ E' &:= \{(i,j) \in [\varrho(C)] \times [\varrho(D)] \mid (U(i), V(j)) \in \overline{E}\} \end{aligned}$$

and where the level synchronization product $[P] \times [Q]$ is defined according to

$$\ell(s) = \begin{cases} 0 & \text{if} \quad s \in V_X \\ 1 & \text{if} \quad s \in V_P - V_X \end{cases} \qquad \ell(t) = \begin{cases} 0 & \text{if} \quad t \in V_Y \\ 1 & \text{if} \quad t \in V_Q - V_Y. \end{cases}$$

Finally we restrict $R \times S$ to the non-terminals accessible from Z. This grammar indeed generates the level synchronization product $(R \times S)^\omega = R^\omega \times S^\omega$ of their generated graphs, and also satisfies the following properties:

$$(R \times S)_{\bullet,\bullet_1} \lhd R \quad ; \quad (R \times S)_{\bullet,\bullet_2} \lhd S \quad ; \quad S \lhd R \implies R \times S \bowtie S.$$

This implies that for any level-unambiguous R, $\mathrm{Sync}(R)$ is closed under intersection.

LEMMA 9. *For any $S, S' \lhd R$ with R level-unambiguous, $L(S \times S') = L(S) \cap L(S')$.*

5.2 Level-wise determinization

Before proving the closure under complement of $\mathrm{Sync}(R)$ in the next subsection, we need to define a suitable notion of level-wise determinism, and show that any level-unambiguous grammar is equivalent, in terms of synchronised languages, to one generating a level-wise deterministic graph. We say that a grammar R is *level-deterministic* if for any $G \in R^\omega$, there is at most one initial vertex per level, and the targets of any pair of arcs with the same source and label have distinct levels: $\circ s, \circ t \in G \vee (r \xrightarrow[G]{a} s \wedge r \xrightarrow[G]{a} t) \implies s = t \vee \ell_G(s) \neq \ell_G(t)$.

In other words, R is level-deterministic if and only if there exists no pair of level-synchronized initial paths in $R^\omega_\circ$. So any grammar generating a deterministic graph is level-deterministic. We state another property oflevel-deterministic grammars.

LEMMA 10. *Any level-deterministic and level-unambiguous grammar is unambiguous.*

Another advantage of level-deterministic grammars is that the synchronization relation is recursive when the synchronizer is level-deterministic (this is proved using the generalized grammar synchronization product defined in the next section).

LEMMA 11. *We can decide whether $R \rhd S$ for R level-deterministic.*

Similarly to way level synchronization is done, we perform the standard powerset construction only level by level.

For R a grammar generating G, let $\Pi := \{P \mid \emptyset \neq P \subseteq V_G \wedge \forall p, q \in P, \ell(p) = \ell(q)\}$ be the set of subsets of vertices with same level, and let $\mathrm{Succ}_a(P)$ be the set of successors of vertices in $P \in \Pi$ by $a \in F_G \cap F_2$: $\mathrm{Succ}_a(P) := \{q \mid \exists p \in P(p \xrightarrow[G]{a} q)\}$. The *level-determinization* of any grammar R is defined as $\mathrm{Det}(R^\omega) := \{\mathrm{Det}(G) \mid G \in R^\omega\}$, where $\mathrm{Det}(G)$ is:

$$\begin{aligned}\mathrm{Det}(G) :=& \{P \xrightarrow{a} Q \mid P, Q \in \Pi \wedge Q \subseteq \mathrm{Succ}_a(P) \wedge \forall q \in \mathrm{Succ}_a(P) - Q, Q \cup \{q\} \notin \Pi\} \\ &\cup \{\circ P \mid P \in \Pi \wedge \forall p \in P(\circ p \in G) \wedge \forall q(\circ q \in G \wedge q \notin P \implies P \cup \{q\} \notin \Pi)\} \\ &\cup \{cP \mid P \in \Pi \wedge c \in F_1 - \{\circ\} \wedge \exists p \in P(cp \in G)\}\end{aligned}$$

restricted to the vertices accessible from $\circ$.

Contrary to the level synchronization product, Det does not preserve regularity. However $\mathrm{Det}(R^\omega)$ can be generated by a grammar when R is in a certain normal form which preserves synchronised languages.

Let us define an *arc grammar* R as an initial grammar whose rules (except the axiom rule) are all of the form $A12 \longrightarrow H_A$ where H_A is a finite graph with no terminal arc of target 1, or of source 2, or of source 1 and target 2 : $s \xrightarrow[[H_A]]{a} t \implies s \neq 2 \wedge t \neq 1 \wedge (s,t) \neq (1,2)$. We tranform a grammar into an arc grammar by splitting non-terminal hyperarcs into non-terminal arcs of arity 2 (hence the name).

LEMMA 12. *Any initial grammar can be transformed into a bi-synchronized arc grammar, while preserving unambiguity.*

This lemma allows to prove Proposition 1 by translating any unambiguous arc grammar R into an unambiguous context-free grammar generating $L(R)$, and conversely.

For any arc grammar R, $\mathrm{Det}(R^\omega)$ can be generated by a grammar $\mathrm{Det}(R)$ that we define. Let R be an arc grammar generating a graph accessible from $\circ$. To any $A \in N_R - \{Z\}$, we

associate a new symbol $\overline{A}$ of arity 2 and we define the grammar $\overline{R}$ obtained from R by adding the rules $\overline{A}12 \longrightarrow H_A$ for all $A \in N_R - \{Z\}$, and then by replacing in the right hand sides any non-terminal arc $s \xrightarrow{B} 2$ by $s \xrightarrow{\overline{B}} 2$:

$$\begin{aligned}\overline{R} := \{(Z, H_Z)\} \cup \{\big(A12, (H_A - N_R V_{H_A} 2) \cup \{\overline{B}s2 \mid B \in N_R \wedge Bs2 \in H_A\}\big) \mid A \in N_R - \{Z\}\} \\ \cup \{\big(\overline{A}12, (H_A - N_R V_{H_A} 2) \cup \{\overline{B}s2 \mid B \in N_R \wedge Bs2 \in H_A\}\big) \mid A \in N_R - \{Z\}\}.\end{aligned}$$

Let $<$ be a linear order over $2^{N_{\overline{R}} - \{Z\}}$ of smallest element $\emptyset$. For each $P \subseteq N_{\overline{R}} - \{Z\}$, $P \neq \emptyset$, we take a new symbol P' of arity $2^{|P|}$ and a hyperarc $<P> = P' p_1 \dots p_m$ with $\{p_1, \dots, p_m\} = 2^P$ and $p_1 < \dots < p_m$, and we define a graph H_P such that $\{Z \xrightarrow{A} A \mid A \in P\} \cup \{\circ Z\} \underset{\overline{R}}{\Longrightarrow} H_P$. In the special case where $P = \emptyset$, we let $<\emptyset> = Z$ and $H_\emptyset = H_Z$.

For every $P \subseteq N_{\overline{R}} - \{Z\}$, we apply to H_P the level-determinization procedure described above to get the graph $H'_P := \mathrm{Det}(H_P)[\emptyset/\{Z\}] - \{\circ\emptyset\}$ whose vertex level mapping ℓ is defined by $\ell(A) = 0$ for all $A \in P - N_R$, $\ell(A) = 1$ for all $A \in P \cap N_R$ and $\ell(s) = 2$ for all $s \in V_{H_P} - (P \cup \{Z\})$. Note that the level $\ell(Z)$ of Z is not significant because there is no arc of target Z in H_P. We define grammar $Det(R)$ by associating to each $P \subseteq N_{\overline{R}} - \{Z\}$ the rule:

$$<P> \quad \longrightarrow \quad [H'_P] \cup \{<Q>[s/\emptyset][\cup_{e \in E} s_e / E]_{\emptyset \neq E \subseteq Q} \mid s \in V_{H'_P} \wedge Q \neq \emptyset\}$$

with $Q := \{A \in N_{\overline{R}} \mid s \xrightarrow[H'_P]{A}\}$ and $s \xrightarrow[H'_P]{A} s_A$ for any $A \in Q$. Note that when R is unambiguous, we can restrict $<P> = P' p_1 \dots p_m$ to $\{p_1, \dots, p_m\} = P$.

LEMMA 13. *For any arc grammar R, $(\mathrm{Det}(R))^\omega = \mathrm{Det}(R^\omega)$, $\mathrm{Det}(R) \bowtie R$ and $Det(R)$ is level-deterministic, hence $\mathrm{Det}(R)$ is unambiguous for R level-unambiguous.*

5.3 Closure under complement

We now consider the closure under complement of $\mathrm{Sync}(R)$ for R level-unambiguous.

First we have to extend the level synchronization product $R \times S$ of any grammars R and S in order to retain a path for all the words accepted by R. We take new colours $\bullet^1, \bullet^2$ and a fresh symbol $\perp$. For any grammars R and S, the *generalized level synchronization product* of their generated graphs is $R^\omega \otimes S^\omega := \{G \otimes H \mid G \in R^\omega \wedge H \in S^\omega\}$, where $G \otimes H$ is defined as:

$$\begin{aligned}G \otimes H := & G \times H \cup \{\bullet^1(s, \perp) \mid \bullet s \in G\} \cup \{\bullet^2(\perp, p) \mid \bullet p \in H\} \\ & \cup \{(s,p) \xrightarrow{a} (t, \perp) \mid s \xrightarrow[G]{a} t \wedge ((s,p) \in V_{G \times H} \vee p = \perp) \wedge \forall q (p \xrightarrow[H]{a} q \Longrightarrow \ell(q) \neq \ell(t))\} \\ & \cup \{(s,p) \xrightarrow{a} (\perp, q) \mid p \xrightarrow[H]{a} q \wedge ((s,p) \in V_{G \times H} \vee s = \perp) \wedge \forall t (s \xrightarrow[G]{a} t \Longrightarrow \ell(t) \neq \ell(q))\} \\ & \cup \{\circ(s, \perp) \mid \circ s \in G \wedge \forall p (\circ p \in H \Longrightarrow \ell(p) \neq \ell(s))\} \\ & \cup \{\circ(\perp, p) \mid \circ p \in H \wedge \forall s (\circ s \in G \Longrightarrow \ell(s) \neq \ell(p))\}.\end{aligned}$$

The definition of $R \times S$ from the previous section is extended to define a grammar $R \otimes S$. The symbol $[A, B, E]$ is now of arity $|E| + \varrho(A) + \varrho(B)$ with the definition

$$[Ar_1 \dots r_m, Bs_1 \dots s_n, E] := [A, B, E](r_1, s_1)_E \dots (r_m, s_n)_E (r_1, \perp) \dots (r_m, \perp)(\perp, s_1) \dots (\perp, s_n)$$

and we replace $\big([P]\times[Q]\big)_{|\overline{E}}$ by $\big([P]\otimes[Q]\big)_{|\overline{E}\,\cup\, V_P\times\{\bot\}\,\cup\,\{\bot\}\times V_Q}$ in the right hand side of the rule of $[AX, BY, E]$. The grammar $R\otimes S$ generates $R^\omega\otimes S^\omega$, and satisfies:

$$(R\otimes S)_{\bullet,\bullet_1,\bullet^1}\bowtie R \quad ; \quad (R\otimes S)_{\bullet,\bullet_2,\bullet^2}\bowtie S \quad ; \quad \forall f\in\{\bullet,\bullet_1,\bullet_2\}, (R\otimes S)_f\bowtie(R\times S)_f.$$

The language $L(R)-L(S)$ for $S\lhd R$ is the set of non accepting words labelling initial paths in $R\otimes S$ which end in a vertex coloured by $\bullet_1$ or $\bullet^1$:

$$L(R)-L(S)=L(R)-(L(R)\cap L(S))=L((R\otimes S)_{\bullet,\bullet_1,\bullet^1})-L(R\otimes S)=L((R\otimes S)_{\bullet_1,\bullet^1})-L(R\otimes S)$$

When $(R\otimes S)_{\bullet,\bullet_1,\bullet^1}$ is unambiguous, the language $L((R\otimes S)_{\bullet_1,\bullet^1})-L(R\otimes S)$ is the set of words which label paths ending in non final vertices coloured by $\bullet_1$ or $\bullet^1$. As $(R\otimes S)_{\bullet,\bullet_1,\bullet^1}$ is level-unambiguous when R is, we get the closure under complement of $\mathrm{Sync}(R)$ using Lemmas 12 and 13.

LEMMA 14. *For R level-unambiguous and $S\lhd R$, $L(R)-L(S)\ \in\mathrm{Sync}(R)$.*

6 Conclusion

For lack of space, we had to omit from this paper a condition on grammars ensuring that their synchronized languages are closed under concatenation and Kleene star. Many other examples of grammars and their families of synchronized languages also have to be studied.

Acknowledgements. Many thanks to Antoine Meyer for helping me prepare the final version of this paper.

References

[AM 04] R. ALUR and P. MADHUSUDAN *Visibly pushdown languages*, 36^{th} STOC, ACM Proceedings, L. Babai (Ed.), 202–211 (2004).

[Ca 07] D. CAUCAL *Deterministic graph grammars*, Texts in Logic and Games 2, Amsterdam University Press, J. Flum, E. Grädel, T. Wilke (Eds.), 169–250 (2007).

[Ca 06] D. CAUCAL *Synchronization of pushdown automata*, 10^{th} DLT, LNCS 4036, O. Ibarra, Z. Dang (Eds.), 120-132 (2006).

[CH 08] D. CAUCAL and S. HASSEN *Synchronization of grammars*, 3^{rd} CSR, LNCS 5010, E. Hirsch, A. Razborov, A. Semenov, A. Slissenko (Eds.), 110–121 (2008).

[Fr 77] E. FRIEDMAN *Equivalence problems for deterministic context-free languages and monadic recursion schemes*, JCSS 14, 344–359 (1977).

[Gi 66] S. GINSBURG *The mathematical theory of context free languages*, McGraw-Hill (1966).

[LMM 08] N. LIMAYE, M. MAHAJAN and A. MEYER *On the complexity of membership and counting in height-deterministic pushdown automata*, 3^{rd} CSR, LNCS 5010, E. Hirsch, A. Razborov, A. Semenov, A. Slissenko (Eds.), 240–251 (2008).

[MS 85] D. MULLER and P. SCHUPP *The theory of ends, pushdown automata, and second-order logic*, Theoretical Computer Science 37, 51–75 (1985).

[NS 07] D. NOWOTKA and J. SRBA *Height-deterministic pushdown automata*, 32^{nd} MFCS, LNCS 4708, L. Kucera, A. Kucera (Eds.), 125–134 (2007).

[Wi 04] K. WICH *Ambiguity functions of context-free grammars and languages*, PhD Thesis, Universität Stuttgart (2004).

Foundations of Software Technology and Theoretical Computer Science (Bangalore) 2008.
Editors: R. Hariharan, M. Mukund, V. Vinay; pp 95-106

Increasing the power of the verifier in Quantum Zero Knowledge

André Chailloux and Iordanis Kerenidis*
CNRS - LRI
Université Paris-Sud
{chaillou,jkeren}@lri.fr

ABSTRACT.
In quantum zero knowledge, the assumption was made that the verifier is only using unitary operations. Under this assumption, many nice properties have been shown about quantum zero knowledge, including the fact that Honest-Verifier Quantum Statistical Zero Knowledge ($HVQSZK$) is equal to Cheating-Verifier Quantum Statistical Zero Knowledge ($QSZK$) (see [17, 18]).
In this paper, we study what happens when we allow an honest verifier to flip some coins in addition to using unitary operations. Flipping a coin is a non-unitary operation but doesn't seem at first to enhance the cheating possibilities of the verifier since a classical honest verifier can flip coins. In this setting, we show an unexpected result: any classical Interactive Proof has an Honest-Verifier Quantum Statistical Zero Knowledge proof with coins. Note that in the classical case, honest verifier SZK is no more powerful than SZK and hence it is not believed to contain even NP. On the other hand, in the case of cheating verifiers, we show that Quantum Statistical Zero Knowledge where the verifier applies any non-unitary operation is equal to Quantum Zero-Knowledge where the verifier uses only unitaries.
One can think of our results in two complementary ways. If we would like to use the honest verifier model as a means to study the general model by taking advantage of their equivalence, then it is imperative to use the unitary definition without coins, since with the general one this equivalence is most probably not true. On the other hand, if we would like to use quantum zero knowledge protocols in a cryptographic scenario where the honest-but-curious model is sufficient, then adding the unitary constraint severely decreases the power of quantum zero knowledge protocols.

1 Introduction

Zero knowledge protocols propose an elegant way of doing formally secure identification. In these interactive protocols, a prover P knows a secret s and he wants to convince a verifier V that he knows s without revealing any information about s. The condition "without revealing any information" has been formalized in [3, 4] and this security condition has been defined in the computational (CZK) and the information-theoretic setting (SZK). Zero knowledge has been extensively studied and found numerous applications in theoretical computer science and cryptography (see [16] and references therein).

In addition, zero knowledge is defined for the case of honest or cheating verifiers. In the honest verifier model, we force the protocol to be zero knowledge only against a verifier who follows the protocol but tries to extract as much information as possible from the

*Supported in part by ACI Securité Informatique SI/03 511 and ANR AlgoQP grants of the French Ministry and in part by the European Commission under the Integrated Project Qubit Applications (QAP) funded by the IST directorate as Contract Number 015848.

interaction. An honest verifier is equivalent to the 'Honest-but-Curious' or 'Semi-Honest' adversary in cryptography. This model has been widely studied in cryptography and is important in certain realistic scenarios (for example online protocols), where the protocols are used in complex interactions with limited capacity of cheating ([5], ch. 7). Moreover, in the case of classical zero knowledge it is particularly interesting, due to the fact that it is equivalent to the general Zero-Knowledge model against cheating verifiers [6].

In 2002, Watrous proposed a quantum equivalent of zero knowledge proofs [17] for the case of honest verifiers. In this definition, the prover and the verifier are allowed to use only unitary operations and the zero knowledge property is defined in a seemingly weaker way than in the classical case. Watrous proved many interesting results for this class, such as complete problems, closure properties and a few years later, that honest verifier equals cheating verifier (i.e. $HVQSZK = QSZK$) [18]. These results provided strong *a posteriori* evidence that Watrous' definition is the right one for quantum Zero Knowledge.

In this paper, we revisit the definition of quantum zero knowledge and examine the importance of the unitarity constraint. First, we increase the power of the honest verifier by allowing him to flip classical coins in addition to performing unitary quantum operations. Note that flipping classical coins is not a unitary operation and that coin flips are also allowed in the classical case. In this new setting, we also strengthen the definition of simulation in order to still catch the essence of Zero-Knowledge protocols. In particular, the verifier does not "forget" or "erase" these coins, since he remains honest but curious. Even though this augmentation to the model seems minimal if not trivial, we prove that any classical interactive proof has a quantum honest-verifier statistical zero-knowledge proof (Section 3) with coins. Note that in the classical case, honest verifier SZK is no more powerful than SZK and hence it is not believed to contain even NP. If, on the other hand, we look at cheating verifiers, we show that the most general cheating strategies for quantum verifiers are the unitary ones. In Section 4, we transform any general Zero Knowledge protocol into a unitary protocol that retains completeness, soundness and the zero-knowledge property.

We like to see the consequences of our results from two different points of view. On one hand, if we want to use the honest verifier model as a means for the study of general zero knowledge, then the most important property that we would like is the equivalence of the two models. This way, one only needs to prove that a protocol is zero knowledge against honest verifiers and immediately conclude that it can also be made zero knowledge against cheating verifiers. Our results show that in this case, Watrous' definition with unitaries is indeed the right one, since we give strong evidence that this equivalence does not hold in the non-unitary case. Moreoover, we prove that the use of non-unitaries does not change the power of a cheating verifier.

On the other hand, the Honest-but-Curious model (that corresponds to the honest verifier) is not only a means for the study of the malicious model (that corresponds to the cheating verifier) but an important model in itself pertinent to many realistic cryptographic scenarios. For example, in certain settings, we can assume that the verifier is semi-honest when he interacts with the prover via a secure interface, eg. an ATM or a secure web interface. In this case, it might suffice to assume that the verifier does not open the ATM by force or hack the webpage, instead he can only provide well-chosen legal inputs to these machines and try to extract as much information as possible from the interaction.

2 Definitions of classical and quantum Statistical Zero Knowledge

An *interactive proof system* for a problem Π is an interactive protocol between a computationally unbounded prover P and a probabilistic polynomial-time verifier V that satisfies the following two properties:

- *Completeness:* if x is a YES instance of Π ($x \in \Pi_Y$), then V will accept with probability greater than $2/3$ after interacting with P on common input x.
- *Soundness:* if x is a NO instance of Π ($x \in \Pi_N$), then for every (even computationally unbounded) prover strategy P^*, V will accept with probability less than $1/3$ after interacting with P^* on common input x.

DEFINITION 1. *We say that a protocol $\langle P, V\rangle$ solves Π if and only if $\langle P, V\rangle$ is an interactive proof system for Π.*

In the classical Zero-Knowledge setting, we want the Verifier to learn nothing from the interaction with the Prover, other than the fact that the input is a Yes instance of the problem ($x \in \Pi_Y$) when it is the case. The way this is formalized is that for $x \in \Pi_Y$, one can simulate in probabilistic polynomial-time the Verifier's view of the protocol $view_{\langle P,V\rangle}(x)$, i.e. his private coins, the messages he received from the Prover and the messages he sent to the Prover. Note that the view is a distribution depending on the random coins of the Prover and the Verifier and contains all the information that the Verifier gains by interacting with the Prover. Specifically,

DEFINITION 2. *A protocol $\langle P, V\rangle$ has the zero-knowledge property for Π if there exists a probabilistic polynomial-time simulator S and a negligible function μ such that for $\forall x \in \Pi_Y$, the simulator outputs a distribution $S(x)$ such that $|view_{\langle P,V\rangle}(x) - S(x)|_1 \leq \mu(|x|)$.*

In our discussion so far, we have considered the case where the Verifier honestly follows the protocol but tries to extract as much information as possible from the interaction with the Prover. In order to do that, the Honest Verifier would keep a copy of all the messages and his coins throughout the protocol and would not erase or discard any of this information.

We can now define the class of Honest Verifier Statistical Zero Knowledge ($HVSZK$):

DEFINITION 3. *$\Pi \in HVSZK$ iff there exists an interactive protocol $\langle P, V\rangle$ that solves Π and that has the zero-knowledge property for Π.*

2.1 Honest Verifier Quantum Statistical Zero Knowledge

Quantum Statistical Zero Knowledge proofs are a special case of Quantum Interactive Proofs. They were defined for honest verifiers by Watrous in [17] and have been also studied in [8, 18, 9]. We can think of a quantum interactive protocol $\langle P, V\rangle(x)$ for a promise problem Π as a circuit $(V_1(x), P_1(x), \ldots, V_k(x), P_k(x))$ acting on $\mathcal{V} \otimes \mathcal{M} \otimes \mathcal{P}$. $\mathcal{V}$ are the Verifier's private qubits, $\mathcal{M}$ the message qubits and $\mathcal{P}$ the Prover's private qubits. $V_i(x)$ (resp. $P_i(x)$) represents the i^{th} action of the Verifier (resp. of the prover) during the protocol and is decribed by a super-operator acting on $\mathcal{V} \otimes \mathcal{M}$ (resp. on $\mathcal{M} \otimes \mathcal{P}$). β_i corresponds to the state

in $\mathcal{V} \otimes \mathcal{M} \otimes \mathcal{P}$ after the i^{th} action of the protocol. In other words, β_0 is the initial state, β_{2i} is the state after P_i and β_{2i-1} the one after V_i.

Defining the Zero-Knowledge property in the quantum setting is not straightforward, even for the Honest Verifier case. We would still like to say that a quantum protocol has the zero knowledge property if there exists an efficient way to simulate the Verifier's view of the protocol. The main difficulty, however, is the definition of the view of the Verifier, since in the quantum case there is no notion of transcript. Indeed, the Verifier and Prover send the same qubits back and forth during the protocol and hence an Honest-but-curious Verifier cannot follow the protocol and simultaneously keep a copy of all the quantum messages that have been previously sent.

Watrous ([17]) tried to resolve these problems by defining honest verifier quantum zero knowledge in the following way: the view of the Honest Verifier for every round j is the Verifier's part of the state β_j, i.e. $view_{\langle P,V\rangle}(j) = Tr_{\mathcal{P}}(\beta_j)$. We say that the Verifier's view can be simulated if there is a negligible function μ such that on any input x and for each step j we can create in quantum polynomial-time a state σ_j such that $\|\sigma_j - view_{\langle V,P\rangle}(j)\| \leq \mu(|x|)$.

We also distinguish the Verifier's view depending on whether the last action was made by the Verifier or the Prover. We note ρ_0 the input state, ρ_i the Verifier's view after P_i and ξ_i the Verifier's view after V_i. Note that for a state σ with $\|\sigma - \rho_i\| \leq \mu(|x|)$ it is easy to see that $\sigma' = V_{i+1}(\sigma)$ is close to $\xi_{i+1} = V_{i+1}(\rho_i)$ in the sense that $\|\sigma' - \xi_{i+1}\| \leq \mu(|x|)$. Hence, we just need to simulate the ρ_i's and hence

DEFINITION 4. *A protocol $\langle P, V\rangle$ has the zero-knowledge property for Π if there is a negligible function μ such that $\forall x \in \Pi_Y$ and $\forall j$ we can create σ_j with quantum polynomial computational power such that $\|\sigma_j - \rho_j\|_{tr} \leq \mu(|x|)$.*

Let us look more closely to the 'round-by-round' definition of the simulation. First, the fact that we simulate the verifier's view at every round and not just at the end of the protocol ensures that the zero knowledge property is retained even if the Honest Verifier follows the protocol up to some round and then decides to abort.

Second, in order for this definition to be pertinent in the honest but curious model, we need to ensure that the verifier will retain all the information that he acquires during the protocol and not forget any of it. One way to ensure this is by restricting the verifier to use only unitary operations. The intuition is that since unitary operations are reversible, they do not allow for 'forgetting' any information. This is precisely the way Watrous defined the class of Honest Verifier Quantum Statistical Zero Knowledge ($HVQSZK$):

DEFINITION 5. *$\Pi \in HVQSZK$ iff there exists a quantum protocol $\langle P, V\rangle$ with V using only unitaries that solves Π and that has the zero-knowledge property for Π.*

The above intuition was later confirmed by the fact that indeed Honest Verifier Quantum Statistical Zero Knowledge with unitaries is equivalent to general cheating verifiers ([18]).

2.2 The coin model for Honest Verifier Quantum Zero-Knowledge

As we said, we would like to investigate the importance of the unitarity constraint in the power of quantum zero knowledge. For this, we define and study a new model for quantum

zero-knowledge protocols, where we just allow the verifier to flip classical coins in addition to performing unitary operations. This is equivalent to saying that the verifier starts with a private random string r^* or in a quantum language that the verifier starts with some private qubits initialized to $|0\rangle$ − acting as the verifier usual workspace, and additionally some qubits in the totally mixed state $\mathbb{I}$ − acting as the verifier's initial coins. The verifier uses his coins (the state $\mathbb{I}$) only as control bits. More formally, if we suppose that the verifier starts with the state $\mathbb{I} \otimes |0\rangle\langle 0|$ in the space $\mathcal{A} \otimes \mathcal{B}$, then he can only use the space $\mathcal{A}$ by applying unitaries of the form:

$$U(|x\rangle, |y\rangle) = |x\rangle \otimes |y \oplus f(x)\rangle \quad \text{with } |x\rangle \in \mathcal{A} \text{ and } |y\rangle \in \mathcal{B}$$

Note that this constraint just implies that the verifier doesn't forget his coins. In particular, he does not discard these bits by sending them to the prover.

In this case, of course, one needs to be very careful with the definition of the simulation since now, the Verifier has the extra classical information of the coins. Since the interaction is quantum we still have to consider a 'round-by-round' simulation. However, in our definition of the 'round-by-round' simulation we need to insist that one must simulate the entire private random string of the verifier in addition to the quantum view of the Verifier.

Note that apart from these additional initial coins, the verifier is allowed to use only unitaries like in the original definition of $HVQSZK$. We can now define $HVQSZK^C$:

DEFINITION 6. *$\Pi \in HVQSZK^C$ iff, there exists a quantum protocol $\langle P, V\rangle$, where the verifier's initial state is $(|0\rangle\langle 0|)^{\otimes n} \otimes \mathbb{I}_n$, that solves Π and has the zero-knowledge property for Π. The verifier uses only unitaries and uses his coins (the state $\mathbb{I}_n$) only as control bits.*

This model is meant to be a very small augmentation of the original model proposed by Watrous. Note that the verifier is not able to create by himself the totally mixed state using only unitaries. It is important to notice that the requirement "the prover uses the state $\mathbb{I}_n$ as control bits" means that these coins are always part of his view of the protocol or in other words that he never forgets his coins.

2.3 The hidden-bits model for Statistical Zero-Knowledge

The hidden-bits model was first defined for Non-Interactive Zero-Knowledge [2], however, it naturally extends to the interactive case.

DEFINITION 7. *We say that the prover has a hidden-bit r with security parameter k iff:*

- *r is a truly random bit known to the prover.*
- *The verifier has no information about r.*
- *The prover can reveal the value of the bit r to the Verifier. If he tries to convince the Verifier that the value is $\bar{r}$ then he will be caught with probability $(1 - 2^{-k})$.*

DEFINITION 8. *$\Pi \in HVSZK^{HB}$ iff there exists a classical protocol $\langle P, V\rangle$ that solves Π and has the zero-knowledge property for Π where the prover starts with a polynomial number of hidden bits.*

We can also define the associated quantum class

DEFINITION 9. *$\Pi \in HVQSZK^{HB}$ iff there exists a quantum protocol $\langle P, V\rangle$ that solves Π and has the zero-knowledge property for Π where the prover starts with a polynomial number of hidden bits.*

Note that the existence of hidden-bits is a very strong assumption. In particular, we can remark that hidden-bits imply that the prover and verifier can perform bit commitment with perfect hiding and statistically binding conditions. Bit commitment is a primitive used in many cryptographic protocols. More formally:

DEFINITION 10. *A bit commitment scheme with perfect hiding condition and statistically binding condition with security parameter k is a scheme with a commit phase and a reveal phase such that:*

- *Commit phase: the prover chooses a bit c and commits to it by interacting with the verifier. At the end of the interaction, the verifier has no information about c (perfectly hiding).*
- *Reveal phase: the prover sends a message to the verifier and reveals the commited bit c. If the prover tries to cheat and reveal $\bar{c}$ then he will be caught by the verifier with probability greater than $(1 - 2^{-k})$ (statistically binding).*

Note that both classically and quantumly, bit commitment schemes with $k \geq 1$ do not exist unconditionally [11, 12]. However, there is an easy way to do bit commitment which is perfectly hiding and statistically binding with security parameter k from a hidden bit r with security parameter k. The prover commits to a bit c by sending $c \oplus r$ and later reveals r. After the commit phase the verifier has no information about r (and hence c) and during the reveal phase the prover cannot lie about r (and hence c) without being caught with probability at least $(1 - 2^{-k})$. Hence, this scheme is a commitment scheme which is perfectly hiding and statistically binding with security parameter k.

Classically, if we suppose the existence of such a bit commitment scheme, we can create zero-knowledge protocols for all interactive proofs [1] and since Shamir showed that $IP = PSPACE$ [15], we have

$$PSPACE = IP \subseteq HVSZK^{HB}$$

3 The role of coins in Quantum Statistical Zero-Knowledge

In this section we prove our main result, that $HVSZK^{HB} \subseteq HVQSZK^{C}$ which implies that $PSPACE \subseteq HVQSZK^{C}$. We will first present a general method to create hidden-bits out of shares. We will then show a way to achieve these shares with a quantum honest verifier that has coins.

3.1 A general method for creating hidden-bits

The method described here is the one used in [14] to create hidden bits from secret help, which in turn uses ideas from [7] in order to do Oblivious Transfer. For clarity of exposition, we show how to hide a single bit, but the construction naturally generalizes to n bits by repeating in parallel.

PROPOSITION 11. *Let three random bits (s^0, s^1, b) be such that: the prover knows s^0 and s^1 and has no information about b; the verifier knows b and the associated bit s^b but has no information about $s^{\overline{b}}$. Then we can create a hidden bit r with security parameter $k = 1$.*

PROOF. From these bits, the associated hidden bit will be $r = s^0 \oplus s^1$ and s^0, s^1 will be called the shares of r. The way the prover will reveal r is by sending these two shares to the verifier who checks that they correspond with the one share he has. We now show that r is a hidden bit with security parameter 1:

- Since s^0 and s^1 are random and known to the prover, then so is r.
- Since the verifier knows s^b but has no information about $s^{\overline{b}}$, he has no information about r.
- If the prover tries to lie about r then he has to flip exaclty one of the two shares. He will get caught if he flips s^b and will not get caught if he flips $s^{\overline{b}}$. Since he has no information about b, he will be caught cheating with probability $1/2$.

Note that if we have for each hidden bit r, k independent random couples of shares (s_k^0, s_k^1) such that $s_k^0 \oplus s_k^1 = r$ then similarly, we can suppose that r is a hidden-bit with security parameter k.

3.2 A quantum way of achieving Hidden Bits

From the coins of the verifier, we now show how to create the shares described in the previous part. As before, we describe the construction of one hidden-bit which easily generalizes to n bits. We use three qubits of the verifier's initial totally mixed state (three coins) as $\sum_{b,s^b,c\in\{0,1\}} |b, s^b, c\rangle\langle b, s^b, c|$.

As in the previous part, the bit b corresponds to which share the Verifier has and s^b corresponds to the value of that share. The bit c corresponds to the value of the other share in the Hadamard basis, i.e. we define $|c^\times\rangle = \frac{1}{\sqrt{2}}(|0\rangle + (-1)^c|1\rangle)$. The verifier performs the unitary $U_{b,s^b,c}$ that depends on (b, s^b, c) and sends the outcome to the Prover.

$$U_{0,s^b,c} : |0\rangle|0\rangle \to |s^b\rangle|c^\times\rangle \quad \text{and} \quad U_{1,s^b,c} : |0\rangle|0\rangle \to |c^\times\rangle|s^b\rangle$$

The prover has two qubits which he measures in the computational basis and the outcomes of this measurement will correspond to the two shares. One of this measurements will give s^b and the other one will give a random bit $s^{\overline{b}}$. The hidden bit r is equal to $s^b \oplus s^{\overline{b}}$.

LEMMA 12. *The above construction results in a hidden bit r with security parameter* 1.

PROOF.

- The bit $r = s^b \oplus s^{\overline{b}}$ is random since the verifier picks s^b at random and the outcome of the measurement of $|c^\times\rangle$ in the computational basis is also random (hence $s^{\overline{b}}$ is random). Since the prover knows the two shares he knows r.
- The verifier knows a share s^b which is random since b is random. He has no information about the share $s^{\overline{b}}$ since the outcome of the Prover's measurement of $|c^\times\rangle$ is independent of the Verifier's coins. Hence he has no information about r.

- b is unknown to the prover: to show this, let ρ_b be the state of the prover conditioned on the verifier's coin b

$$\rho_0 = \begin{cases} wp.\ 1/4 & |0,+\rangle \\ wp.\ 1/4 & |0,-\rangle \\ wp.\ 1/4 & |1,+\rangle \\ wp.\ 1/4 & |1,-\rangle \end{cases} \quad \text{and} \quad \rho_1 = \begin{cases} wp.\ 1/4 & |+,0\rangle \\ wp.\ 1/4 & |+,1\rangle \\ wp.\ 1/4 & |-,0\rangle \\ wp.\ 1/4 & |-,1\rangle \end{cases}$$

  We can easily see that $\rho_0 = \rho_1$ hence the prover has no information about b. Moreover, since $\rho_0 = \rho_1 = \mathbb{I}$, the prover's state is equivalent to a mixture of classical pairs of shares. Since he has no information about b, the prover cannot cheat for any of those classical pairs of shares with probability striclty greater than $1/2$.

We can easily extend the above construction to a hidden-bit with security parameter k for any polynomial k (by creating k independent pairs of shares for this hidden-bit) and also to n hidden-bits with security parameter k by just repeating this process n times.

Note also that the unitary used by the verifier uses his coins only as control bits. Therefore, we can use this construction to create hidden bits in a way which is consistent with our enhanced notion of simulation and show that $HVSZK^{HB} \subseteq HVQSZK^{C}$. Let us prove this fact formally:

PROPOSITION 13. $HVSZK^{HB} \subseteq HVQSZK^{C}$

PROOF.

Let Π a problem in $HVSZK^{HB}$ and $\langle P, V\rangle$ a classical zero-knowledge protocol with hidden-bits that solves Π. We create the following quantum protocol $\langle P', V'\rangle$ where the verifier starts with the state : $(|0\rangle\langle 0|)^{\otimes n} \otimes \mathbb{I}_n$ (acting as his workspace and coins).

- The verifier V' views his coins as the coins of the original verifier V and the coins needed in order to create hidden bits.
- In the beginning of the protocol the verifier uses our construction and creates hidden bits with security parameter k.
- Then, the prover and verifier both follow the original classical protocol $\langle P, V\rangle$. Note that this is possible since any classical circuit C can be transformed into a quantum unitary circuit U_C such that $U_C(|x,0\rangle) = |x, C(x)\rangle$.

Note that since V' uses his coins as the private randomness of V, he can perform the classical protocol $\langle P, V\rangle$ using unitaries.

We now prove that $\langle P', V'\rangle$ is a Zero-Knowledge protocol that solves Π. Completeness is straightforward from the completeness of the original protocol and the fact that in our construction the prover can always reveal the correct hidden bits. Concerning soundness:

1. If the prover reveals all the hidden-bits correctly, the soundness of $\langle P', V'\rangle$ is the same as the soundness of $\langle P, V\rangle$.
2. If the prover lies on at least one of the hidden-bits he reveals, then the soundness of $\langle P', V'\rangle$ will be smaller than 2^{-k} since the hidden-bits created have security parameter k.

To show the zero-knowledge property, we use the fact that we can already simulate the verifier's view in the protocol $\langle P, V\rangle$. This includes the private coins of V, the messages and in particular, all the hidden-bits r_i revealed by the prover.

In order to simulate the verifier's view in the new protocol $\langle P, V\rangle$ we have to additionally simulate the following:

- all the coins that the verifier V' used in order to create hidden bits.
- The k pairs of shares $(s_i^0, s_i^1)_j, j \in [k]$ for every revealed hidden bit r_i.

First, the simulator just flips some coins in order to simulate all the random bits the verifier uses to construct the hidden bits. In particular, for every revealed hidden bit r_i, the simulator has the corresponding bits $(b_i, s_i^b, c_i)_j, j \in [k]$. From these bits and the value of r_i (which we know from the original simulation), we can now create all the couples of shares $(s_i^0, s_i^1)_j$. This allows us to simulate the view of the verifier in the protocol $\langle P', V'\rangle$.

THEOREM 14. *$PSPACE \subseteq HVQSZK^C$*

PROOF. From Section 2.3 we know that: $PSPACE \subseteq HVSZK^{HB}$. We now use the fact that $HVSZK^{HB} \subseteq HVQSZK^C$ and conclude.

One might think that this surprising result comes from the fact that the round-by-round simulation is too weak in our setting and that a satisfactory zero-knowledge property is not achieved. In fact, if we assume that the verifier follows the protocol, then our notion of simulation is as strong as in the unitary case. The only extra information that the verifier has in our protocols is the initial random string which we always simulate at every round.

4 Non-unitaries and cheating verifiers

4.1 Definitions

The goal of this section is to describe Watrous' definition of Quantum Statistical Zero Knowledge ($QSZK$) for cheating verifiers. Consider a quantum zero-knowledge protocol between a prover P and a verifier V where the verifier starts with an auxiliary input w. Additionally, the prover and verifier have as common input the input of the promise problem which is a classical string. All the operations described hereafter will depend on this input and this dependence will be omitted.

We will use the following Hilbert spaces for our analysis.

- $\mathcal{P}$ the space of the prover.
- $\mathcal{M}$ the space where the prover and verifier store the messages they send.
- $\mathcal{V}$ the verifier's workspace initialized to $|0\rangle$.
- $\mathcal{W}$ the verifier's space where the auxiliary input is initially stored.

Let $\langle P, V\rangle = \langle P_1, V_1, \ldots, P_n, V_n\rangle$. Each P_i acts on $\mathcal{P} \otimes \mathcal{M}$ and each V_i acts on $\mathcal{M} \otimes \mathcal{V} \otimes \mathcal{W}$. We can tensor these operations with the identity and suppose that they all act on the space: $\mathcal{P} \otimes \mathcal{M} \otimes \mathcal{V} \otimes W$. We can therefore see the whole protocol as a big operation O acting on $\mathcal{P} \otimes \mathcal{M} \otimes \mathcal{V} \otimes W$. More formally:

DEFINITION 15. *For any protocol $\langle P_1, V_1, \ldots, P_n, V_n\rangle$ where each V_i and P_i acts on $\mathcal{P} \otimes \mathcal{M} \otimes \mathcal{V} \otimes W$ (in fact by tensoring the V_i's and P_i's with the identity) we denote by $O_{P,V}$ the following admissible mapping:*

$$\begin{aligned} O_{P,V} \quad &: \mathcal{L}(\mathcal{W}) \quad \to \mathcal{L}(\mathcal{P} \otimes \mathcal{M} \otimes \mathcal{V} \otimes W) \\ &: w \qquad\quad \to V_n(P_n(\ldots(V_1(P_1(\underbrace{|0\rangle}_{\in \mathcal{P}\otimes\mathcal{M}\otimes\mathcal{V}} \otimes \underbrace{w}_{\in \mathcal{W}})))))\end{aligned}$$

where $\mathcal{L}(\mathcal{X},\mathcal{Y})$ is the set of linear operators from $\mathcal{X}$ to $\mathcal{Y}$, and $\mathcal{L}(\mathcal{X}) = \mathcal{L}(\mathcal{X},\mathcal{X})$. In particular, any mixed state in $\mathcal{X}$ can be represented as an element of $\mathcal{L}(\mathcal{X})$.

The zero-knowledge property concerns only what the verifier has at the end of the protocol. Without loss of generality, we can suppose that $\mathcal{M}$ is empty since a cheating verifier can always move the information from $\mathcal{M}$ to $\mathcal{V}$ at the end of the protocol. Hence, we will be interested in:

$$\begin{aligned} O_V &: \mathcal{L}(\mathcal{W}) \rightarrow \mathcal{L}(\mathcal{V}\otimes W) \\ &: w \rightarrow Tr_{\mathcal{P}\otimes\mathcal{M}}\left(V_n(P_n(\ldots(V_1(P_1(\underbrace{|0\rangle}_{\in\mathcal{P}\otimes\mathcal{M}\otimes\mathcal{V}}\otimes\underbrace{w}_{\in\mathcal{W}})))))\right) \end{aligned}$$

which for short we will also denote as $O_V = Tr_{\mathcal{P}\otimes\mathcal{M}}O_{P,V}$. More generally, for any super-operator X that outputs in $\mathcal{A}\otimes\mathcal{B}$, we denote $Tr_{\mathcal{A}}X$ the super-operator such that $(Tr_{\mathcal{A}}X)(\rho) = Tr_{\mathcal{A}}(X(\rho))$. We say that O_V is the mapping that corresponds to the verifier's view of the protocol. We want to be able to simulate this mapping *i.e.* be able to create in quantum polynomial time a mapping Σ which will act like O_V and this for every auxiliary input w. We can now define $QSZK$:

DEFINITION 16. *We say that $\Pi \in QSZK$ if there is a protocol $\langle P,V\rangle = \langle P_1,V_1,\ldots,P_n,V_n\rangle$ such that:*

- *Completeness: $\forall x \in \Pi_Y$, the verifier accepts with probability greater than $2/3$.*
- *Soundness: $\forall x \in \Pi_N$, and for all prover's strategies P^*, the verifier accepts with probability smaller than $1/3$.*
- *Zero-knowledge: for any cheating verifier V^* (where O_{V^*} is the mapping associated to $\langle P,V^*\rangle$), there is a function μ and a mapping $\Sigma : \mathcal{L}(\mathcal{W}) \rightarrow \mathcal{L}(\mathcal{V}\otimes W)$ that can be computed in quantum polynomial time such that $\forall x \in \Pi_Y$, we have*

$$||O_{V^*} - \Sigma||_\diamond \leq \mu(|x|).$$

where for any super-operator Φ, $||\Phi||_\diamond = sup\{||\Phi\otimes I_{\mathcal{L}(\mathcal{Z})}||_{tr},\ \mathcal{Z}$ is a complex Euclidean space$\}$ (see [10] for more details on this diamond norm).

Note that if Σ uses V^* only as a black box, then we can change the order of quantifiers and have a single mapping Σ for all possible V^*.

In the definition of $QSZK$, the verifier and the prover can use any physically admissible operation. We will show that in fact, if the zero-knowledge property holds against cheating verifiers that only use unitaries then it also holds for cheating verifiers that use any physically admissible operation. In other words, cheating strategies with unitary operations are the most general ones.

DEFINITION 17. *We say that $\Pi \in QSZK^U$ if there is a protocol $\langle P,V\rangle = \langle P_1,V_1,\ldots,P_n,V_n\rangle$ such that:*

- *Completeness: $\forall x \in \Pi_Y$, the verifier accepts with probability greater than $2/3$.*
- *Soundness: $\forall x \in \Pi_N$, and for all prover's strategies P^*, the verifier accepts with probability smaller than $1/3$.*

- *Zero-knowledge: for any cheating verifier V^* that uses unitaries (where O_{V^*} is the mapping associated to $\langle P, V^* \rangle$), there is a function μ and a mapping $\Sigma : \mathcal{L}(\mathcal{W}) \rightarrow \mathcal{L}(\mathcal{V} \otimes W)$ that can be computed in quantum polynomial time such that $\forall x \in \Pi_Y$, we have*

$$||O_{V^*} - \Sigma||_\diamond \leq \mu(|x|).$$

4.2 Unitary cheating verifiers are as powerful as general cheating verifiers

In this section, we show that in the case of cheating verifiers, coin flips − and more generally any non-unitary operations − do not add anything to the power of quantum Zero-Knowledge. In other words, we show that

PROPOSITION 18. $QSZK = QSZK^U$

PROOF. We have by definition that $QSZK \subseteq QSZK^U$. We show now the other inclusion. The main idea is to say that each time the verifier uses a non-unitary, he can use a larger unitary which will act as a purification of this non-unitary which will only give him more information. More formally, we use the following fact that is a direct corollary of the purification lemma. (see [13]).

LEMMA 19. *Let C a quantum non-unitary circuit acting on a space A. There is a space $\mathcal{B}$ of same dimension as $\mathcal{A}$ and a unitary circuit $\widetilde{C}$ acting on $\mathcal{A} \otimes \mathcal{B}$ such that $Tr_{\mathcal{B}}\widetilde{C} = C$.*

Now consider a protocol $\langle P, V \rangle = \langle P_1, V_1, \ldots, P_n, V_n \rangle$ which has the zero-knowledge property for any unitary cheating verifier V. Consider a cheating verifier V^*, the protocol $\langle P, V^* \rangle = \langle P_1, V_1^*, \ldots, P_n, V_n^* \rangle$ and its associated mapping O_{V^*} from $\mathcal{L}(\mathcal{W})$ to $\mathcal{L}(\mathcal{V} \otimes \mathcal{W})$. Recall that:

$$O_{V^*} = Tr_{\mathcal{P} \otimes \mathcal{M}} \left(P_n \circ V_n^* \circ \ldots \circ P_1 \circ V_1^* \right)$$

Consider now n additional Hilbert spaces $\mathcal{A}_1$ through $\mathcal{A}_n$ and admissible mappings $\widetilde{V}_i^*$ such that

$$\forall i \; Tr_{\mathcal{A}_i} \widetilde{V}_i^* = V_i^*$$

The spaces $\mathcal{A}_i$ are Hilbert spaces that the verifier possesses. Let us look at the protocol $\langle P, \widetilde{V}^* \rangle = \langle P_1, \widetilde{V}_1^*, \ldots, P_n, \widetilde{V}_n^* \rangle$ and $O_{\widetilde{V}^*}$ the associated mapping for the verifier. This mapping is a mapping from $\mathcal{L}(\mathcal{W})$ to $\mathcal{L}(\mathcal{A}_1 \otimes \ldots \otimes \mathcal{A}_n \otimes \mathcal{V} \otimes \mathcal{W})$. We know that there is a mapping Σ computable in quantum polynomial time such that $||O_{\widetilde{V}^*} - \Sigma||_\diamond \leq \mu(|x|)$.

By construction, we know that $O_{V^*} = Tr_{\mathcal{A}_1 \otimes \ldots \otimes \mathcal{A}_n} O_{\widetilde{V}^*}$. Consider $\Sigma' = Tr_{\mathcal{A}_1 \otimes \ldots \otimes \mathcal{A}_n} \Sigma$, we can easily conclude that

$$||O_{V^*} - \Sigma'||_\diamond \leq \mu(|x|)$$

and that Σ' is quantum polynomial time computable which concludes our proof.

References

[1] Michael Ben-Or, Oded Goldreich, Shafi Goldwasser, Johan Hastad, Joe Kilian, Silvio Micali, and Phillip Rogaway. Everything provable is provable in zero-knowledge. In *CRYPTO '88: Proceedings on Advances in cryptology*, pages 37–56, New York, NY, USA, 1990. Springer-Verlag New York, Inc.

[2] Uriel Feige, Dror Lapidot, and Adi Shamir. Multiple noninteractive zero knowledge proofs under general assumptions. *SIAM J. Comput.*, 29(1):1–28, 2000.

[3] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput.*, 18(1):186–208, 1989.

[4] Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity or all languages in np have zero-knowledge proof systems. *J. ACM*, 38(3):690–728, 1991.

[5] Oded Goldreich. *Foundations of Cryptography*, volume Basic Tools. Cambridge University Press, 2001.

[6] Oded Goldreich, Amit Sahai, and Salil Vadhan. Honest-verifier statistical zero-knowledge equals general statistical zero-knowledge. In *STOC '98: Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 399–408, New York, NY, USA, 1998. ACM.

[7] Joe Kilian. Founding crytpography on oblivious transfer. In *STOC '88: Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 20–31, New York, NY, USA, 1988. ACM Press.

[8] Hirotada Kobayashi. Non-interactive quantum perfect and statistical zero-knowledge. *ISAAC '03: International Symposium on Algorithms And Computation*, 2906:178–188, 2003.

[9] Hirotada Kobayashi. General Properties of Quantum Zero-Knowledge Proofs. *ArXiv Quantum Physics e-prints*, quant-ph/0705.1129, May 2007.

[10] A. Yu. Kitaev, A. H. Shen, and M. N. Vyalyi. *Classical and Quantum Computation*. American Mathematical Society, Boston, MA, USA, 2002.

[11] Hoi-Kwong Lo and H. F. Chau. Is quantum bit commitment really possible? *Phys. Rev. Lett.*, 78(17):3410–3413, Apr 1997.

[12] Dominic Mayers. Unconditionally secure quantum bit commitment is impossible. *Phys. Rev. Lett.*, 78(17):3414–3417, Apr 1997.

[13] Michael A. Nielsen and Isaac L. Chuang. *Quantum computation and quantum information*. Cambridge University Press, New York, NY, USA, 2000.

[14] Rafael Pass and abhi shelat. Unconditional Characterizations of Non-Interactive Zero-Knowledge. In *CRYPTO '05*, pages 118–134. Springer Berlin / Heidelberg, 2005.

[15] Adi Shamir. IP = PSPACE. *J. ACM*, 39(4):869–877, 1992.

[16] Salil Pravin Vadhan. *A study of statistical zero-knowledge proofs*. PhD thesis, 1999. Supervisor-Shafi Goldwasser.

[17] John Watrous. Limits on the power of quantum statistical zero-knowledge. In *FOCS '02: Proceedings of the 43rd Symposium on Foundations of Computer Science*, pages 459–468, Washington, DC, USA, 2002. IEEE Computer Society.

[18] John Watrous. Zero-knowledge against quantum attacks. In *STOC '06: Proceedings of the thirty-eighth annual ACM Symposium on Theory of Computing*, pages 296–305, New York, NY, USA, 2006. ACM Press.

Foundations of Software Technology and Theoretical Computer Science (Bangalore) 2008.
Editors: R. Hariharan, M. Mukund, V. Vinay; pp 107-118

Algorithms for Game Metrics*

Krishnendu Chatterjee[1], Luca de Alfaro[1], Rupak Majumdar[2], Vishwanath Raman[1]

[1] University of California, Santa Cruz
{c_krish,luca,vishwa}@soe.ucsc.edu

[2] University of California, Los Angeles
rupak@cs.ucla.edu

ABSTRACT. Simulation and bisimulation metrics for stochastic systems provide a quantitative generalization of the classical simulation and bisimulation relations. These metrics capture the similarity of states with respect to quantitative specifications written in the quantitative μ-calculus and related probabilistic logics.
We present algorithms for computing the metrics on Markov decision processes (MDPs), turn-based stochastic games, and concurrent games. For turn-based games and MDPs, we provide a polynomial-time algorithm based on linear programming for the computation of the one-step metric distance between states. The algorithm improves on the previously known exponential-time algorithm based on a reduction to the theory of reals. We then present PSPACE algorithms for both the decision problem and the problem of approximating the metric distance between two states, matching the best known bound for Markov chains. For the bisimulation kernel of the metric, which corresponds to probabilistic bisimulation, our algorithm works in time $\mathcal{O}(n^4)$ for both turn-based games and MDPs; improving the previously best known $\mathcal{O}(n^9 \cdot \log(n))$ time algorithm for MDPs.
For a concurrent game G, we show that computing the exact distance between states is at least as hard as computing the value of concurrent reachability games and the square-root-sum problem in computational geometry. We show that checking whether the metric distance is bounded by a rational r, can be accomplished via a reduction to the theory of real closed fields, involving a formula with three quantifier alternations, yielding $\mathcal{O}(|G|^{\mathcal{O}(|G|^5)})$ time complexity, improving the previously known reduction with $\mathcal{O}(|G|^{\mathcal{O}(|G|^7)})$ time complexity. These algorithms can be iterated to approximate the metrics using binary search.

1 Introduction

System metrics constitute a quantitative generalization of system relations. The bisimulation relation captures state *equivalence:* two states s and t are bisimilar if and only if they cannot be distinguished by any formula of the μ-calculus [4]. The bisimulation *metric* captures the *degree of difference* between two states: the bisimulation distance between s and t is a real number that provides a tight bound for the difference in value of formulas of the *quantitative* μ-calculus at s and t [9]. A similar connection holds between the simulation relation and the simulation metric.

The classical system relations are a basic tool in the study of *boolean* properties of systems, that is, the properties that yield a truth value. As an example, if a state s of a transition

*This research was supported in part by the NSF grants CCR-0132780 and CNS-0720884.

FSTTCS 2008
IARCS Annual Conference on
Foundations of Software Technology and Theoretical Computer Science
http://drops.dagstuhl.de/opus/volltexte/2008/1745

system can reach a set of target states R, written $s \models \Diamond R$ in temporal logic, and t can simulate s, then we can conclude $t \models \Diamond R$. System metrics play a similarly fundamental role in the study of the quantitative behavior of systems. As an example, if a state s of a Markov chain can reach a set of target states R with probability 0.8, written $s \models \mathbb{P}_{\geq 0.8}\Diamond R$, and if the metric simulation distance from t to s is 0.3, then we can conclude $t \models \mathbb{P}_{\geq 0.5}\Diamond R$. The simulation relation is at the basis of the notions of system refinement and implementation, where qualitative properties are concerned. Similarly, simulation metrics provide a notion of approximate refinement and implementation for quantitative properties.
We consider three classes of systems:

- *Markov decision processes.* In these systems there is one player. At each state, the player can choose a move; the current state and the move determine a probability distribution over the successor states.
- *Turn-based games.* In these systems there are two players. At each state, only one of the two players can choose a move; the current state and the move determine a probability distribution over the successor states.
- *Concurrent games.* In these systems there are two players. At each state, both players choose moves simultaneously and independently; the current state and the chosen moves determine a probability distribution over the successor states.

System metrics were first studied for Markov chains and Markov decision processes (MDPs) [9, 18, 19], and they have recently been extended to two-player turn-based and concurrent games [8]. The fundamental property of the metrics is that they provide a tight bound for the difference in value that formulas belonging to quantitative specification languages assume at the states of a system. Precisely, let $q\mu$ indicate the *quantitative μ-calculus,* a specification language in which many of the classical specification properties, including reachability and safety properties, can be written [7]. The metric bisimulation distance between two states s and t, denoted $[s \simeq_g t]$, has the property that $[s \simeq_g t] = \sup_{\varphi \in q\mu} |\varphi(s) - \varphi(t)|$, where $\varphi(s)$ and $\varphi(t)$ are the values φ assumes at s and t. A metric is associated with a *kernel:* the kernel of a metric is the relation that relates pairs of states at distance 0; to each metric corresponds a metric kernel relation. The kernel of the simulation metric is *probabilistic simulation;* the kernel of the bisimulation metric is *probabilistic bisimulation* [15].

We investigate algorithms for the computation of the metrics. The metrics can be computed in iterative fashion, following the inductive way in which they are defined. A metric d can be computed as the limit of a monotonically increasing sequence of approximations $d_0, d_1, d_2, \ldots$, where $d_0(s,t)$ is the difference in value that variables can have at states s and t. For $k \geq 0$, d_{k+1} is obtained from d_k via $d_{k+1} = H(d_k)$, where the operator H depends on the metric (bisimulation, or simulation), and on the type of system. Our main results are as follows:

1. *Metrics for turn-based games and MDPs.* We show that for turn-based games, and MDPs, the one-step metric operator H for both bisimulation and simulation can be computed in polynomial time, via a reduction to linear programming (LP). The only previously known algorithm, which can be inferred from [8], had EXPTIME complexity and relied on a reduction to the theory of real closed fields; the algorithm thus had more a complexity-theoretic, than a practical value. The key step in obtaining our polynomial-time algorithm consists in transforming the original sup-inf *non-linear* op-

timization problem (which required the theory of reals) into a quadratic-size inf *linear* optimization problem that can be solved via LP. We then present PSPACE algorithms for both the decision problem of the metric distance between two states and for the problem of computing the approximate metric distance between two states for turn-based games and MDPs. Our algorithms match the complexity of the best known algorithms for the sub-class of Markov chains [17].

2. *Metrics for concurrent games.* For concurrent games, our algorithms for the H operator still rely on decision procedures for the theory of real closed fields, leading to an EXPTIME procedure. However, the algorithms that could be inferred from [8] had time-complexity $\mathcal{O}(|G|^{\mathcal{O}(|G|^7)})$, where $|G|$ is the size of a game; we improve this result by presenting algorithms with $\mathcal{O}(|G|^{\mathcal{O}(|G|^5)})$ time-complexity.
3. *Hardness of metric computation in concurrent games.* We show that computing the exact distance of states of concurrent games is at least as hard as computing the value of concurrent reachability games [10], which is known to be at least as hard as solving the square-root-sum problem in computational geometry. These two problems are known to lie in PSPACE, and have resisted many attempts to show that they are in NP.
4. *Kernel of the metrics.* We present polynomial time algorithms to compute the simulation and bisimulation kernel of the metrics for turn-based games and MDPs. Our algorithm for the bisimulation kernel of the metric runs in time $\mathcal{O}(n^4)$ (assuming a constant number of moves) as compared to the previous known $\mathcal{O}(n^9 \cdot \log(n))$ algorithm of [21] for MDPs, where n is the size of the state space. For concurrent games the simulation and the bisimulation kernel can be computed in time $\mathcal{O}(|G|^{\mathcal{O}(|G|^3)})$, where $|G|$ is the size of a game.

Our formulation of probabilistic simulation and bisimulation differs from the one previously considered for MDPs in [1]: there, the names of moves (called "labels") must be preserved by simulation and bisimulation, so that a move from a state has at most one candidate simulator move at another state. Our problem for MDPs is closer to the one considered in [21], where labels must be preserved, but where a label can be associated with multiple probability distributions (moves).

For turn-based games and MDPs, the algorithms for probabilistic simulation and bisimulation can be obtained from the LP algorithms that yield the metrics. For probabilistic simulation, the algorithm we obtain coincides with the algorithm of [21]. The algorithm requires the solution of feasibility-LP problems with a number of variables and inequalities that is quadratic in the size of the system. For probabilistic bisimulation, we are able to improve on this result by providing an algorithm that requires the solution of feasibility-LP problems that have linearly many variables and constraints. Precisely, as for ordinary bisimulation, the kernel is computed via iterative refinement of a partition of the state space [14]. Given two states that belong to the same partition, to decide whether the states need to be split in the next partition-refinement step, we present an algorithm that requires the solution of a feasibility-LP problem with a number of variables equal to the number of moves available at the states, and number of constraints linear in the number of equivalence classes. The proofs omitted due to lack of space are available in [6].

2 Definitions

Valuations and distributions. Let $[\theta_1, \theta_2] \subseteq \mathbb{R}$ be a fixed, non-singleton real interval. Given a set of states S, a *valuation over* S is a function $f : S \mapsto [\theta_1, \theta_2]$ associating with every state $s \in S$ a value $\theta_1 \leq f(s) \leq \theta_2$; we let $\mathcal{F}$ be the set of all valuations. For $c \in [\theta_1, \theta_2]$, we denote by $\mathbf{c}$ the constant valuation such that $\mathbf{c}(s) = c$ at all $s \in S$. We order valuations pointwise: for $f, g \in \mathcal{F}$, we write $f \leq g$ iff $f(s) \leq g(s)$ at all $s \in S$; we remark that $\mathcal{F}$, under $\leq$, forms a lattice. Given $a, b \in \mathbb{R}$, we write $a \sqcup b = \max\{a, b\}$, and $a \sqcap b = \min\{a, b\}$; we extend $\sqcap, \sqcup$ to valuations by interpreting them in pointwise fashion. For a finite set A, let $\mathrm{Dist}(A)$ denote the set of probability distributions over A. We say that $p \in \mathrm{Dist}(A)$ is *deterministic* if there is $a \in A$ such that $p(a) = 1$. We assume a fixed finite set $\mathcal{V}$ of *observation variables*.

Game structures. A (two-player, concurrent) *game structure* $G = \langle S, [\cdot], \mathit{Moves}, \Gamma_1, \Gamma_2, \delta \rangle$ consists of the following components: (a) a finite set S of states; (b) a variable interpretation $[\cdot] : \mathcal{V} \mapsto S \mapsto [\theta_1, \theta_2]$, which associates with each variable $v \in \mathcal{V}$ a valuation $[v]$; (c) a finite set *Moves* of moves; (d) two move assignments Γ_1, Γ_2: $S \mapsto 2^{\mathit{Moves}} \setminus \emptyset$: for $i \in \{1, 2\}$, the assignment Γ_i associates with each state $s \in S$ the nonempty set $\Gamma_i(s) \subseteq \mathit{Moves}$ of moves available to player i at state s; and (e) a probabilistic transition function δ: $S \times \mathit{Moves} \times \mathit{Moves} \mapsto \mathrm{Dist}(S)$, that gives the probability $\delta(s, a_1, a_2)(t)$ of a transition from s to t when player 1 plays move a_1 and player 2 plays move a_2.

At every state $s \in S$, player 1 chooses a move $a_1 \in \Gamma_1(s)$, and simultaneously and independently player 2 chooses a move $a_2 \in \Gamma_2(s)$. The game then proceeds to a successor state $t \in S$ with probability $\delta(s, a_1, a_2)(t)$. We let $\mathrm{Dest}(s, a_1, a_2) = \{t \in S \mid \delta(s, a_1, a_2)(t) > 0\}$. The *propositional distance* $p(s, t)$ between two states $s, t \in S$ is the maximum difference in valuation over all variables: $p(s, t) = \max_{v \in \mathcal{V}} |[v](s) - [v](t)|$. The kernel of the propositional distance induces an equivalence on states: for states s, t, we let $s \equiv t$ if $p(s, t) = 0$. In the following, unless otherwise noted, the definitions refer to a game structure G with components $\langle S, [\cdot], \mathit{Moves}, \Gamma_1, \Gamma_2, \delta \rangle$. We indicate the opponent of a player $i \in \{1, 2\}$ by $\sim i = 3 - i$. We consider the following subclasses of games.

Turn-based game structures and MDPs. A game structure G is *turn-based* if $S = S_1 \cup S_2$ with $S_1 \cap S_2 = \emptyset$ where $s \in S_1$ implies $|\Gamma_2(s)| = 1$, and $s \in S_2$ implies $|\Gamma_1(s)| = 1$, and further, there exists a special variable $\mathit{turn} \in \mathcal{V}$, such that $[\mathit{turn}]s = \theta_1$ iff $s \in S_1$, and $[\mathit{turn}]s = \theta_2$ iff $s \in S_2$. For $i \in \{1, 2\}$, we say that a structure is an i-MDP if $\forall s \in S$, $|\Gamma_{\sim i}(s)| = 1$. For MDPs, we omit the (single) move of the player without a choice of moves, and write $\delta(s, a)$ for the transition function.

Moves and strategies. A *mixed move* is a probability distribution over the moves available to a player at a state. We denote by $\mathcal{D}_i(s) \subseteq \mathrm{Dist}(\mathit{Moves})$ the set of mixed moves available to player $i \in \{1, 2\}$ at $s \in S$, where: $\mathcal{D}_i(s) = \{\mathcal{D} \in \mathrm{Dist}(\mathit{Moves}) \mid \mathcal{D}(a) > 0 \textit{ implies } a \in \Gamma_i(s)\}$. The moves in *Moves* are called *pure moves.* We extend the transition function to mixed moves by defining, for $s \in S$ and $x_1 \in \mathcal{D}_1(s)$, $x_2 \in \mathcal{D}_2(s)$, $\delta(s, x_1, x_2)(t) = \sum_{a_1 \in \Gamma_1(s)} \sum_{a_2 \in \Gamma_2(s)} \delta(s, a_1, a_2)(t) \cdot x_1(a_1) \cdot x_2(a_2)$. A *path* σ of G is an infinite sequence $s_0, s_1, s_2, \ldots$ of states in $s \in S$, such that for all $k \geq 0$, there are mixed moves $x_1^k \in \mathcal{D}_1(s_k)$ and $x_2^k \in \mathcal{D}_2(s_k)$ with $\delta(s_k, x_1^k, x_2^k)(s_{k+1}) > 0$. We write Σ for the set of all paths, and Σ_s the set of all paths starting from state s.

A *strategy* for player $i \in \{1, 2\}$ is a function $\pi_i : S^+ \mapsto \mathrm{Dist}(\mathit{Moves})$ that associates with

every non-empty finite sequence $\sigma \in S^+$ of states, representing the history of the game, a probability distribution $\pi_i(\sigma)$, which is used to select the next move of player i; we require that for all $\sigma \in S^*$ and states $s \in S$, if $\pi_i(\sigma s)(a) > 0$, then $a \in \Gamma_i(s)$. We write Π_i for the set of strategies for player i. Once the starting state s and the strategies π_1 and π_2 for the two players have been chosen, the game is reduced to an ordinary stochastic process, denoted $G_s^{\pi_1,\pi_2}$, which defines a probability distribution on the set Σ of paths. We denote by $\Pr_s^{\pi_1,\pi_2}(\cdot)$ the probability of a measurable event with respect to this process, and denote by $\mathbb{E}_s^{\pi_1,\pi_2}(\cdot)$ the associated expectation operator. For $k \geq 0$, we let $X_k : \Sigma \to S$ be the random variable denoting the k-th state along a path.

One-step expectations and predecessor operators. Given a valuation $f \in \mathcal{F}$, a state $s \in S$, and two mixed moves $x_1 \in \mathcal{D}_1(s)$ and $x_2 \in \mathcal{D}_2(s)$, we define the expectation of f from s under x_1, x_2 by $\mathbb{E}_s^{x_1,x_2}(f) = \sum_{t \in S} \delta(s, x_1, x_2)(t)\, f(t)$. For a game structure G, for $i \in \{1,2\}$ we define the *valuation transformer* $\mathrm{Pre}_i : \mathcal{F} \mapsto \mathcal{F}$: for all $f \in \mathcal{F}$ and $s \in S$, $\mathrm{Pre}_i(f)(s) = \sup_{x_i \in \mathcal{D}_i(s)} \inf_{x_{\sim i} \in \mathcal{D}_{\sim i}(s)} \mathbb{E}_s^{x_i, x_{\sim i}}(f)$. Intuitively, $\mathrm{Pre}_i(f)(s)$ is the maximal expectation player i can achieve of f after one step from s: this is the standard "one-day" or "next-stage" operator of the theory of repeated games [11].

Game bisimulation and simulation metrics. A *directed metric* is a function $d : S^2 \mapsto \mathbb{R}_{\geq 0}$ which satisfies $d(s,s) = 0$ and the *triangle inequality* $d(s,t) \leq d(s,u) + d(u,t)$ for all $s,t,u \in S$. We denote by $\mathcal{M} \subseteq S^2 \mapsto \mathbb{R}$ the space of all directed metrics; this space, ordered pointwise, forms a lattice which we indicate with $(\mathcal{M}, \leq)$. Since $d(s,t)$ may be zero for $s \neq t$, these functions are *pseudo-metrics* as per prevailing terminology [18]. In the following, we omit "directed" and simply say metric when the context is clear.

For a metric d, we indicate with $C(d)$ the set of valuations $k \in \mathcal{F}$ where $k(s) - k(t) \leq d(s,t)$ for every $s,t \in S$. A metric transformer $H_{\preceq_1} : \mathcal{M} \mapsto \mathcal{M}$ is defined as follows, for all $d \in \mathcal{M}$ and $s,t \in S$: $H_{\preceq_1}(d)(s,t) = p(s,t) \sqcup \sup_{k \in C(d)} \big(\mathrm{Pre}_1(k)(s) - \mathrm{Pre}_1(k)(t)\big)$. The *player 1 game simulation metric* $[\preceq_1]$ is the least fixpoint of $H_{\preceq_1}$; the *game bisimulation metric* $[\simeq_1]$ is the least symmetrical fixpoint of $H_{\preceq_1}$ and is defined as follows, for all $d \in \mathcal{M}$ and $s,t \in S$:

$$H_{\simeq_1}(d)(s,t) = H_{\preceq_1}(d)(s,t) \sqcup H_{\preceq_1}(d)(t,s)\ . \tag{1}$$

The operator $H_{\preceq_1}$ is monotonic, non-decreasing and continuous in the lattice $(\mathcal{M}, \leq)$. We can therefore compute $H_{\preceq_1}$ using Picard iteration; we denote by $[\preceq_1^n] = H_{\preceq_1}^n(\mathbf{0})$ the n-iterate of this. From the determinacy of concurrent games with respect to ω-regular goals [12], we have that the game bisimulation metric is *reciprocal*, in that $[\simeq_1] = [\simeq_2]$; we will thus simply write $[\simeq_g]$. Similarly, for all $s,t \in S$ we have $[s \preceq_1 t] = [t \preceq_2 s]$.

The main result in [8] about these metrics is that they are logically characterized by the quantitative μ-calculus of [7]. We omit the formal definition of the syntax and semantics of the quantitative μ-calculus (see [7] for details). Given a game structure G, every closed formula φ of the quantitative μ-calculus defines a valuation $[\![\varphi]\!] \in \mathcal{F}$. Let $q\mu$ (respectively, $q\mu_1^+$) consist of all quantitative μ-calculus formulas (respectively, all quantitative μ-calculus formulas with only the Pre_1 operator and all negations before atomic propositions). The result of [8] shows that for all states $s,t \in S$,

$$[s \preceq_1 t] = \sup_{\varphi \in q\mu_1^+} \left([\![\varphi]\!](s) - [\![\varphi]\!](t)\right) \qquad [s \simeq_g t] = \sup_{\varphi \in q\mu} \left|[\![\varphi]\!](s) - [\![\varphi]\!](t)\right|\ . \tag{2}$$

Metric kernels. The kernel of the metric $[\simeq_g]$ defines an equivalence relation $\simeq_g$ on the states of a game structure: $s \simeq_g t$ iff $[s \simeq_g t] = 0$; the relation $\simeq_g$ is called the *game bisimulation* relation [8]. We define the *game simulation* preorder $s \preceq_1 t$ as the kernel of the directed metric $[\preceq_1]$, that is, $s \preceq_1 t$ iff $[s \preceq_1 t] = 0$. For notational ease, given a relation $R \subseteq S \times S$, we denote by $1_R : S \times S \mapsto \{0,1\}$ its characteristic set, defined as $1_R(s,t) = 1$ iff $(s,t) \in R$. Given a relation $R \subseteq S \times S$, let $B(R) \subseteq \mathcal{F}$ consist of all valuations $k \in \mathcal{F}$ such that, for all $s,t \in S$, if sRt then $k(s) \leq k(t)$.

3 Algorithms for Turn-Based Games and MDPs

In this section, we present algorithms for computing the metric and its kernel for turn-based games and MDPs. We first present a polynomial time algorithm to compute the operator $H_{\preceq_i}(d)$ that gives the *exact* one-step distance between two states, for $i \in \{1,2\}$. We then present a PSPACE algorithm to decide whether the limit distance between two states s and t (i.e., $[s \preceq_1 t]$) is at most a rational value r. Our algorithm matches the best known bound for the special class of Markov chains [17]. Finally, we present improved algorithms for the important case of the kernel of the metrics. For the bisimulation kernel our algorithm is significantly more efficient compared to previous algorithms.

Algorithms for the metrics. For turn-based games and MDPs, only one player has a choice of moves at a given state. We consider two player 1 states. A similar analysis applies to player 2 states. We remark that the distance between states in S_i and $S_{\sim i}$ is always $\theta_2 - \theta_1$ due to the existence of the variable *turn*. For a metric $d \in \mathcal{M}$, and states $s,t \in S_1$, computing $H_{\preceq_1}(d)(s,t)$, given that $p(s,t)$ is trivially computed by its definition, entails evaluating the expression, $\sup_{k \in C(d)} \sup_{x \in \mathcal{D}_1(s)} \inf_{y \in \mathcal{D}_1(t)} (\mathbb{E}^x_s(k) - \mathbb{E}^y_t(k))$. By expanding the expectations, we get the following form,

$$\sup_{k \in C(d)} \sup_{x \in \mathcal{D}_1(s)} \inf_{y \in \mathcal{D}_1(t)} \Big(\sum_{u \in S} \sum_{a \in \Gamma_1(s)} \delta(s,a)(u) \cdot x(a) \cdot k(u) - \sum_{v \in S} \sum_{b \in \Gamma_1(t)} \delta(t,b)(v) \cdot y(b) \cdot k(v) \Big). \quad (3)$$

We observe that the one-step distance as defined in (3) is a *sup-inf non-linear (quadratic)* optimization problem. The following lemma transforms (3) to an *inf linear* optimization problem, which can be solved by linear programming.

Lemma 1 *For all turn-based game structures G, for all player i states s and t, given a metric $d \in \mathcal{M}$, the following equality holds,*

$$\sup_{k \in C(d)} \sup_{x \in \mathcal{D}_i(s)} \inf_{y \in \mathcal{D}_i(t)} (\mathbb{E}^x_s(k) - \mathbb{E}^y_t(k)) = \sup_{a \in \Gamma_i(s)} \inf_{y \in \mathcal{D}_i(t)} \sup_{k \in C(d)} (\mathbb{E}^a_s(k) - \mathbb{E}^y_t(k)) .$$

Therefore, given $d \in \mathcal{M}$, we can write the player 1 one-step distance between states s and t as follows,

$$\mathsf{OneStep}(s,t,d) = \sup_{a \in \Gamma_1(s)} \inf_{y \in \mathcal{D}_1(t)} \sup_{k \in C(d)} (\mathbb{E}^a_s(k) - \mathbb{E}^y_t(k)) . \quad (4)$$

Hence we compute the expression $\mathsf{OneStep}(s,t,d,a) = \inf_{y \in \mathcal{D}_1(t)} \sup_{k \in C(d)} (\mathbb{E}^a_s(k) - \mathbb{E}^y_t(k))$ for all $a \in \Gamma_1(s)$, and then choose the maximum: $\max_{a \in \Gamma_1(s)} \mathsf{OneStep}(s,t,d,a)$. We now

present a lemma that helps to reduce the above inf-sup optimization problem to a linear program. We first introduce some notation. Let λ denote the set of variables $\lambda_{u,v}$, for $u, v \in S$. Given $d \in \mathcal{M}$, $a \in \Gamma_1(s)$, and a distribution $y \in \mathcal{D}_1(t)$, we write $\lambda \in \Phi(d, a, y)$ if the following linear constraints are satisfied:

$$\text{(1) for all } v \in S : \sum_{u \in S} \lambda_{u,v} = \delta(s,a)(v); \qquad \text{(2) for all } u \in S : \sum_{v \in S} \lambda_{u,v} = \sum_{b \in \Gamma_1(t)} y(b) \cdot \delta(t,b)(u);$$

$$\text{(3) for all } u, v \in S : \lambda_{u,v} \geq 0 \,.$$

Lemma 2 *For all turn-based games and MDPs, for all $d \in \mathcal{M}$, and for all $s, t \in S$, we have*

$$\sup_{a \in \Gamma_1(s)} \inf_{y \in \mathcal{D}_1(t)} \sup_{k \in C(d)} (\mathbb{E}_s^a(k) - \mathbb{E}_t^y(k)) = \sup_{a \in \Gamma_1(s)} \inf_{y \in \mathcal{D}_1(t)} \inf_{\lambda \in \Phi(d,a,y)} \Big(\sum_{u,v \in S} d(u,v) \cdot \lambda_{u,v} \Big) \,.$$

Using the above result we obtain the following LP for $\mathsf{OneStep}(s, t, d, a)$ over the variables: (a) $\{\lambda_{u,v}\}_{u,v \in S}$, and (b) y_b for $b \in \Gamma_1(t)$:

$$\text{Minimize} \quad \sum_{u,v \in S} d(u,v) \cdot \lambda_{u,v} \quad \text{subject to} \tag{5}$$

$$\text{(1) for all } v \in S : \sum_{u \in S} \lambda_{u,v} = \delta(s,a)(v); \qquad \text{(2) for all } u \in S : \sum_{v \in S} \lambda_{u,v} = \sum_{b \in \Gamma_1(t)} y_b \cdot \delta(t,b)(u);$$

$$\text{(3) for all } u, v \in S : \lambda_{u,v} \geq 0; \qquad \text{(4) for all } b \in \Gamma_1(t) : y_b \geq 0; \qquad \text{(5)} \sum_{b \in \Gamma_1(t)} y_b = 1 \,.$$

Theorem 1 *For all turn-based games and MDPs, given $d \in \mathcal{M}$, for all states $s, t \in S$, we can compute $H_{\preceq_1}(d)(s,t)$ in polynomial time by the LP (5).*

Iteration of $\mathsf{OneStep}(s, t, d)$ converges to the exact distance. However, in general, there are no known bounds for the rate of convergence. We now present a decision procedure to check whether the exact distance between two states is at most a rational value r. We first show a way to express the predicate $d(s,t) = \mathsf{OneStep}(s, t, d)$, for a given $d \in \mathcal{M}$. We observe that since $H_{\preceq_1}$ is non-decreasing, we have $\mathsf{OneStep}(s, t, d) \geq d(s,t)$. It follows that the equality $d(s,t) = \mathsf{OneStep}(s, t, d)$ holds iff all the linear inequalities of LP (5) are satisfied, and $d(s,t) = \sum_{u,v \in S} d(u,v) \cdot \lambda_{u,v}$ holds. It then follows that $d(s,t) = \mathsf{OneStep}(s, t, d)$ can be written as a predicate in the theory of real closed fields. Given a rational r, two states s and t, we present an existential theory of reals formula to decide whether $[s \preceq_1 t] \leq r$. Since $[s \preceq_1 t]$ is the least fixed point of $H_{\preceq_1}$, we define a formula $\Phi(r)$ that is true iff $[s \preceq_1 t] \leq r$, as follows: $\Phi(r) = \exists d \in \mathcal{M}.[(\mathsf{OneStep}(s, t, d) = d(s,t)) \wedge (d(s,t) \leq r)]$. If the formula $\Phi(r)$ is true, then there is a fixpoint that is bounded by r, which means that the least fixpoint is bounded by r. Conversely, if the least fixpoint is bounded by r, then the least fixpoint is a witness d for $\Phi(r)$ being true. Since the existential theory of reals is decidable in PSPACE [5], we have the following result.

Theorem 2 (Decision complexity for exact distance). *For all turn-based games and MDPs, given a rational r, and two states s and t, whether $[s \preceq_1 t] \leq r$ can be decided in PSPACE.*

Approximation. For a rational $\epsilon > 0$, using binary search and $\mathcal{O}(\log(\frac{\theta_2-\theta_1}{\epsilon}))$ calls to check $\Phi(r)$, we can obtain an interval $[l, u]$ with $u - l \leq \epsilon$ such that $[s \preceq_1 t]$ lies in the interval $[l, u]$.

Algorithms for the kernel. The kernel of the simulation metric $\preceq_1$ can be computed as the limit of the series $\preceq_1^0$, $\preceq_1^1$, $\preceq_1^2$, ..., of relations. For all $s, t \in S$, we have $(s, t) \in \preceq_1^0$ iff $s \equiv t$. For all $n \geq 0$, we have $(s, t) \in \preceq_1^{n+1}$ iff $\mathsf{OneStep}(s, t, 1_{\preceq_1^n}) = 0$. Checking the condition $\mathsf{OneStep}(s, t, 1_{\preceq_1^n}) = 0$, corresponds to solving an LP feasibility problem for every $a \in \Gamma_1(s)$, as it suffices to replace the minimization goal $\gamma = \sum_{u,v \in S} 1_{\preceq_1^n}(u, v) \cdot \lambda_{u,v}$ with the constraint $\gamma = 0$ in the LP (5). This is the same LP feasibility problem that was introduced in [21] as part of an algorithm to decide simulation of probabilistic systems in which each label may lead to one or more distributions over states.

For the bisimulation kernel, we present a more efficient algorithm, which also improves on the algorithms presented in [21]. The idea is to proceed by partition refinement, as usual for bisimulation computations. The refinement step is as follows: given a partition, two states s and t belong to the same refined partition iff every pure move from s induces a probability distribution on equivalence classes that can be matched by mixed moves from t, and vice versa. Precisely, we compute a sequence $\mathcal{Q}^0$, $\mathcal{Q}^1$, $\mathcal{Q}^2$, ..., of partitions. Two states s, t belong to the same class of $\mathcal{Q}^0$ iff they have the same variable valuation (i.e., iff $s \equiv t$). For $n \geq 0$, since by the definition of the bisimulation metric given in (1), $[s \simeq_g t] = 0$ iff $[s \preceq_1 t] = 0$ and $[t \preceq_1 s] = 0$, two states s, t in a given class of $\mathcal{Q}^n$ remain in the same class in $\mathcal{Q}^{n+1}$ iff both (s, t) and (t, s) satisfy the set of feasibility LP problems $\mathsf{OneStepBis}(s, t, \mathcal{Q}^n)$ as given below:

> $\mathsf{OneStepBis}(s, t, \mathcal{Q})$ consists of one feasibility LP problem for each $a \in \Gamma(s)$. The problem for $a \in \Gamma(s)$ has set of variables $\{x_b \mid b \in \Gamma(t)\}$, and set of constraints:
>
> $$\text{(1) for all } b \in \Gamma(t) : \; x_b \geq 0, \qquad \text{(2)} \sum_{b \in \Gamma(t)} x_b = 1,$$
>
> $$\text{(3) for all } V \in \mathcal{Q} : \sum_{b \in \Gamma(t)} \sum_{u \in V} x_b \cdot \delta(t, b)(u) \geq \sum_{u \in V} \delta(s, a)(u) \; .$$

Complexity. The number of partition refinement steps required for the computation of both the simulation and the bisimulation kernel is bounded by $O(|S|^2)$ for turn-based games and MDPs, where S is the set of states. At every refinement step, at most $O(|S|^2)$ state pairs are considered, and for each state pair (s, t) at most $|\Gamma(s)|$ LP feasibility problems needs to be solved. Let us denote by $\mathsf{LPF}(n, m)$ the complexity of solving the feasibility of m linear inequalities over n variables. We obtain the following result.

Theorem 3 *For all turn-based games and MDPs G, the following assertions hold: (a) the simulation kernel can be computed in $\mathcal{O}\big(n^4 \cdot m \cdot \mathsf{LPF}(n^2 + m, n^2 + 2n + m + 2)\big)$ time; and (b) the bisimulation kernel can be computed in $\mathcal{O}\big(n^4 \cdot m \cdot \mathsf{LPF}(m, n + m + 1)\big)$ time; where $n = |S|$ is the size of the state space, and $m = \max_{s \in S} |\Gamma(s)|$.*

Remarks: The best known algorithm for $\mathsf{LPF}(n, m)$ works in time $\mathcal{O}(n^{2.5} \cdot \log(n))$ [20] (assuming each arithmetic operation takes unit time). The previous algorithm for the bisimulation kernel checked two way simulation and hence has the complexity $\mathcal{O}(n^4 \cdot m \cdot (n^2 + m)^{2.5} \cdot \log(n^2 + m))$, whereas our algorithm works in time $\mathcal{O}(n^4 \cdot m \cdot m^{2.5} \cdot \log(m))$. For most

practical purposes, the number of moves at a state is constant (i.e., m is constant). For the case when m is constant, the previous best algorithm worked in $\mathcal{O}(n^9 \cdot \log(n))$ time, whereas our algorithm works in time $\mathcal{O}(n^4)$.

4 Algorithms for Concurrent Games

In this section we first show that the computation of the metric distance is at least as hard as the computation of optimal values in concurrent reachability games. The exact complexity of the latter is open, but it is known to be at least as hard as the square-root sum problem, which is in PSPACE but whose inclusion in NP is a long-standing open problem [10]. Next, we present algorithms based on a decision procedure for the theory of real closed fields, for both checking the bounds of the exact distance and the kernel of the metrics. Our reduction to the theory of real closed fields removes one quantifier alternation when compared to the previous known formula (inferred from [8]). This improves the complexity of the algorithm.

Reduction of reachability games to metrics. We will use the following terms in the result. A *proposition* is a boolean observation variable, and we say a state is labeled by a proposition q iff q is true at s. For a proposition q, let $\Diamond q$ denote the set of paths that visit a state labeled by q at least once. In concurrent reachability games, the objective is $\Diamond q$, for a proposition q.

Theorem 4 *Consider a concurrent game structure G, with a single proposition q. We can construct in linear-time a concurrent game structure G', with one additional state t', such that for all $s \in S$, we have*

$$[s \preceq_1 t'] = \sup_{\pi_1 \in \Pi_1} \inf_{\pi_2 \in \Pi_2} \Pr_s^{\pi_1,\pi_2}(\Diamond q) .$$

Algorithms for the metrics. We present a lemma that helps obtain reduced-complexity algorithms for concurrent games. The lemma states that the distance $[s \preceq_1 t]$ is attained by restricting player 2 to pure moves at state t, for all states $s, t \in S$.

Lemma 3 *Given a game structure G and a distance $d \in \mathcal{M}$, we have*

$$\sup_{k \in C(d)} \sup_{x_1 \in \mathcal{D}_1(s)} \inf_{y_1 \in \mathcal{D}_1(t)} \sup_{y_2 \in \mathcal{D}_2(t)} \inf_{x_2 \in \mathcal{D}_2(s)} (\mathbb{E}_s^{x_1,x_2}(k)) - \mathbb{E}_t^{y_1,y_2}(k))$$
$$= \sup_{k \in C(d)} \sup_{x_1 \in \mathcal{D}_1(s)} \inf_{y_1 \in \mathcal{D}_1(t)} \sup_{b \in \Gamma_2(t)} \inf_{x_2 \in \mathcal{D}_2(s)} (\mathbb{E}_s^{x_1,x_2}(k) - \mathbb{E}_t^{y_1,b}(k)) . \quad (6)$$

We now present algorithms for metrics in concurrent games. Due to the reduction from concurrent reachability games, shown in Theorem 4, it is unlikely that we have an algorithm in NP for the metric distance between states. We therefore construct statements in the theory of real closed fields, firstly to decide whether $[s \preceq_1 t] \leq r$, for a rational r, so that we can approximate the metric distance between states s and t, and secondly to decide if $[s \preceq_1 t] = 0$ in order to compute the kernel of the game simulation and bisimulation metrics.

The statements improve on the complexity that can be achieved by a direct translation of the statements of [8] to the theory of real closed fields. The complexity reduction is based on the observation that using Lemma 3, we can replace a sup operator with finite conjunction, and therefore reduce the quantifier complexity of the resulting formula. Fix a game

structure G and states s and t of G. We proceed to construct a statement in the theory of reals that can be used to decide if $[s \preceq_1 t] \leq r$, for a given rational r.

In the following, we use variables x_1, y_1 and x_2 to denote a set of variables $\{x_1(a) \mid a \in \Gamma_1(s)\}$, $\{y_1(a) \mid a \in \Gamma_1(t)\}$ and $\{x_2(b) \mid b \in \Gamma_2(s)\}$ respectively. We use k to denote the set of variables $\{k(u) \mid u \in S\}$, and d for the set of variables $\{d(u,v) \mid u, v \in S\}$. The variables $\alpha, \alpha', \beta, \beta'$ range over reals. For convenience, we assume $\Gamma_2(t) = \{b_1, \ldots, b_l\}$.

First, notice that we can write formulas that state that a variable x is a mixed move for a player at state s, and k is a constructible predicate (i.e., $k \in C(d)$):

$$\mathsf{IsDist}(x, \Gamma_1(s)) \equiv \bigwedge_{a \in \Gamma_1(s)} x(a) \geq 0 \wedge \bigwedge_{a \in \Gamma_1(s)} x(a) \leq 1 \wedge \sum_{a \in \Gamma_1(s)} x(a) = 1$$

$$\mathsf{kBounded}(k,d) \equiv \bigwedge_{u \in S} \left[k(u) \geq \theta_1 \wedge k(u) \leq \theta_2 \right] \wedge \bigwedge_{u,v \in S} (k(u) - k(v) \leq d(u,v)) \; .$$

In the following, we write bounded quantifiers of the form "$\exists x_1 \in \mathcal{D}_1(s)$" or "$\forall k \in C(d)$" which mean respectively $\exists x_1.\mathsf{IsDist}(x_1, \Gamma_1(s)) \wedge \cdots$ and $\forall k.\mathsf{kBounded}(k,d) \rightarrow \cdots$.

Let $\eta(k, x_1, x_2, y_1, b)$ be the polynomial $\mathbb{E}_s^{x_1,x_2}(k) - \mathbb{E}_t^{y_1,b}(k)$. Notice that η is a polynomial of degree 3. We construct the formula for game simulation in stages. First, we construct a formula $\Phi_1(d,k,x,\alpha)$ with free variables d, k, x, α such that $\Phi_1(d,k,x_1,\alpha)$ holds for a valuation to the variables iff $\alpha = \inf_{y_1 \in \mathcal{D}_1(t)} \sup_{b \in \Gamma_2(t)} \inf_{x_2 \in \mathcal{D}_2(s)} (\mathbb{E}_s^{x_1,x_2}(k) - \mathbb{E}_t^{y_1,b}(k))$. We use the following observation to move the innermost inf ahead of the sup over the finite set $\Gamma_2(t)$ (for a function f):

$$\sup_{b \in \Gamma_2(t)} \inf_{x_2 \in \mathcal{D}_2(s)} f(b, x_2, x) = \inf_{x_2^{b_1} \in \mathcal{D}_2(s)} \ldots \inf_{x_2^{b_l} \in \mathcal{D}_2(s)} \max(f(b_1, x_2^{b_1}, x), \ldots, f(b_l, x_2^{b_l}, x)) \; .$$

Using the above observation the formula $\Phi_1(d,k,x_1,\alpha)$ can be written as a $\forall\exists$ formula (i.e., with one quantifier alternation) in the theory of reals (see [6] for the formula). Using Φ_1, we construct a formula $\Phi(d,\alpha)$ with free variables d and α such that $\Phi(d,\alpha)$ is true iff: $\alpha = \sup_{k \in C(d)} \sup_{x_1 \in \mathcal{D}_1(s)} \inf_{y_1 \in \mathcal{D}_1(t)} \sup_{b \in \Gamma_2(t)} \inf_{x_2 \in \mathcal{D}_2(s)} (\mathbb{E}_s^{x_1,x_2}(k) - \mathbb{E}_t^{y_1,b}(k))$. The formula Φ is defined as follows:

$$\forall k \in C(d).\forall x_1 \in \mathcal{D}_1(s).\forall \beta.\forall \alpha'. \left[\begin{array}{c} \Phi_1(d,k,x_1,\beta) \rightarrow (\beta \leq \alpha) \wedge \\ (\forall k' \in C(d).\forall x_1' \in \mathcal{D}_1(s).\forall \beta'.\Phi_1(d,k',x_1',\beta') \wedge \beta' \leq \alpha') \rightarrow \alpha \leq \alpha' \end{array} \right] . \quad (7)$$

Finally, given a rational r, we can check if $[s \preceq_1 t] \leq r$ by checking if the following sentence is true: $\exists d \in \mathcal{M}.\exists a \in \mathcal{M}.[\Phi(d,a) \wedge (d = a) \wedge (d(s,t) \leq r)]$. The above sentence is true iff the least fixpoint is bounded by r. Like in the case of turn-based games and MDPs, given a rational $\epsilon > 0$, using binary search and $\mathcal{O}(\log(\frac{\theta_2 - \theta_1}{\epsilon}))$ calls to a decision procedure to check the above sentence, we can compute an interval $[l,u]$ with $u - l \leq \epsilon$, such that $[s \preceq_1 t] \in [l,u]$.

Complexity. Note that Φ is of the form $\forall\exists\forall$, because Φ_1 is of the form $\forall\exists$, and appears in negative position in Φ. The formula Φ has $(|S| + |\Gamma_1(s)| + 3)$ universally quantified variables, followed by $(|S| + |\Gamma_1(s)| + 3 + 2(|\Gamma_1(t)| + |\Gamma_2(s)| \cdot |\Gamma_2(t)| + |\Gamma_2(t)| + 2))$ existentially

quantified variables, followed by $2(|\Gamma_1(t)| + |\Gamma_2(s)| \cdot |\Gamma_2(t)| + |\Gamma_2(t)| + 1)$ universal variables. The sentence for the least fixpoint introduces $|S|^2 + |S|^2$ existentially quantified variables ahead of Φ. The matrix of the formula is of length at most quadratic in the size of the game, and the maximum degree of any polynomial in the formula is 3. We define the size of a game G as: $|G| = |S| + |T|$, where $|T| = \sum_{s,t \in S} \sum_{a,b \in Moves} |\delta(s,a,b)(t)|$. From the complexity of deciding a formula in the theory of real closed fields [2] we get the following result.

Theorem 5 (Decision complexity for exact distance). *For all concurrent games G, given a rational r, and two states s and t, whether $[s \preceq_1 t] \leq r$ can be decided in time $\mathcal{O}(|G|^{\mathcal{O}(|G|^5)})$.*

In contrast, the formula to check whether $[s \preceq_1 t] \leq r$, for a rational r, as implied by the definition of $H_{\preceq_1}(d)(s,t)$, that does not use Lemma 3, has five quantifier alternations due to the inner sup, which when combined with the $2 \cdot |S|^2$ existentially quantified variables in the sentence for the least fixpoint, yields a decision complexity of $\mathcal{O}(|G|^{\mathcal{O}(|G|^7)})$.

Computing the kernels. Similar to the case of turn-based games and MDPs, the kernel of the simulation metric $\preceq_1$ for concurrent games can be computed as the limit of the series $\preceq_1^0, \preceq_1^1, \preceq_1^2, \ldots$, of relations. For all $s,t \in S$, we have $(s,t) \in \preceq_1^0$ iff $s \equiv t$. For all $n \geq 0$, we have $(s,t) \in \preceq_1^{n+1}$ iff the following sentence Φ_s is true: $\forall a.\Phi(\preceq^n, a) \rightarrow a \leq 0$, where Φ is defined as in (7). At any step in the iteration, the distance between any pair of states $u, v \in S$ is defined as follows: for all $u, v \in S$ we have $d(u,v) = 0$ if $(s,t) \in \preceq_1^n$, else if $(s,t) \notin \preceq_1^n$ then $d(u,v) = 1$. To compute the bisimulation kernel, we again proceed by partition refinement. For a set of partitions $\mathcal{Q}^0, \mathcal{Q}^1, \ldots$, $(s,t) \in \simeq^{n+1}$ iff the following sentence Φ_b is true for the state pairs (s,t) and (t,s): $\forall a.\Phi(\mathcal{Q}^n, a) \rightarrow a \leq 0$.

Complexity. In the worst case we need $\mathcal{O}(|S|^2)$ partition refinement steps for computing both the simulation and the bisimulation relation. At each partition refinement step the number of state pairs we consider is bounded by $\mathcal{O}(|S|^2)$. We can check if Φ_s and Φ_b are true using a decision procedure for the theory of real closed fields. Therefore, we need $\mathcal{O}(|S|^4)$ decisions to compute the kernels. The partitioning of states based on the decisions can be done by any of the partition refinement algorithms.

Theorem 6 *For all concurrent games G, states s and t, whether $s \preceq_1 t$ can be decided in $\mathcal{O}(|G|^{\mathcal{O}(|G|^3)})$ time, and whether $s \simeq_g t$ can be decided in $\mathcal{O}(|G|^{\mathcal{O}(|G|^3)})$ time.*

References

[1] C. Baier. Polynomial time algorithms for testing probabilistic bisimulation and simulation. In *CAV*, volume 1102 of *LNCS*, pages 50–61. Springer-Verlag, 1996.

[2] S. Basu. New results on quantifier elimination over real closed fields and applications to constraint databases. *J. ACM*, 46(4):537–555, 1999.

[3] D.P. Bertsekas. *Dynamic Programming and Optimal Control*. Athena Scientific, 1995. Vol.1&2.

[4] M.C. Browne, E.M. Clarke, and O. Grumberg. Characterizing finite Kripke structures in propositional temporal logic. *Theoretical Computer Science*, 59:115–131, 1988.

[5] J. F. Canny. Some algebraic and geometric computations in pspace. In *STOC*, pages 460–467. ACM Press, 1988.

[6] Krishnendu Chatterjee, Luca de Alfaro, Rupak Majumdar, and Vishwanath Raman. Algorithms for game metrics (full version). *CoRR*, abs/0809.4326, 2008.
[7] L. de Alfaro and R. Majumdar. Quantitative solution of omega-regular games. *Journal of Computer and System Sciences*, 68:374–397, 2004.
[8] L. de Alfaro, R. Majumdar, V. Raman, and M. Stoelinga. Game relations and metrics. In *LICS*, pages 99–108. IEEE Computer Society Press, 2007.
[9] J. Desharnais, V. Gupta, R. Jagadeesan, and P. Panangaden. Metrics for labelled Markov systems. In *CONCUR*, volume 1664 of *LNCS*, pages 258–273. Springer-Verlag, 1999.
[10] K. Etessami and M. Yannakakis. Recursive concurrent stochastic games. In *ICALP (2)*, volume 4052 of *LNCS*, pages 324–335. Springer-Verlag, 2006.
[11] J. Filar and K. Vrieze. *Competitive Markov Decision Processes*. Springer-Verlag, 1997.
[12] D.A. Martin. The determinacy of Blackwell games. *The Journal of Symbolic Logic*, 63(4):1565–1581, 1998.
[13] J.F. Mertens and A. Neyman. Stochastic games. *International Journal of Game Theory*, 10:53–66, 1981.
[14] R. Milner. Operational and algebraic semantics of concurrent processes. In *Handbook of Theoretical Computer Science*, volume B, pages 1202–1242. Elsevier Science Publishers, 1990.
[15] R. Segala and N.A. Lynch. Probabilistic simulations for probabilistic processes. In *CONCUR*, volume 836 of *LNCS*, pages 481–496. Springer-Verlag, 1994.
[16] L.S. Shapley. Stochastic games. *Proc. Nat. Acad. Sci. USA*, 39:1095–1100, 1953.
[17] F. van Breugel, B. Sharma, and J. Worrell. Approximating a behavioural pseudometric without discount for probabilistic systems. *CoRR*, abs/0803.3796, 2008.
[18] F. van Breugel and J. Worrell. An algorithm for quantitative verification of probabilistic transition systems. In *CONCUR*, volume 2154 of *LNCS*, pages 336–350. Springer-Verlag, 2001.
[19] F. van Breugel and J. Worrell. Towards quantitative verification of probabilistic transition systems. In *ICALP*, volume 2076 of *LNCS*, pages 421–432. Springer-Verlag, 2001.
[20] Y. Ye. Improved complexity results on solving real-number linear feasibility problems. *Math. Program.*, 106(2):339–363, 2006.
[21] L. Zhang and H. Hermanns. Deciding simulations on probabilistic automata. In *ATVA*, volume 4762 of *LNCS*, pages 207–222. Springer-Verlag, 2007.

Foundations of Software Technology and Theoretical Computer Science (Bangalore) 2008.
Editors: R. Hariharan, M. Mukund, V. Vinay; pp 119-130

Pruning 2-Connected Graphs
(Extended Abstract)

Chandra Chekuri*, **Nitish Korula**†
Dept. of Computer Science
University of Illinois
Urbana, IL 61801
{chekuri,nkorula2}@cs.uiuc.edu

ABSTRACT. Given an edge-weighted undirected graph G with a specified set of terminals, let the *density* of any subgraph be the ratio of its weight/cost to the number of terminals it contains. If G is 2-connected, does it contain smaller 2-connected subgraphs of density comparable to that of G? We answer this question in the affirmative by giving an algorithm to *prune* G and find such subgraphs of any desired size, at the cost of only a logarithmic increase in density (plus a small additive factor). We apply the pruning techniques to give algorithms for two NP-Hard problems on finding large 2-vertex-connected subgraphs of low cost; no previous approximation algorithm was known for either problem. In the k-2VC problem, we are given an undirected graph G with edge costs and an integer k; the goal is to find a minimum-cost 2-vertex-connected subgraph of G containing at least k vertices. In the Budget-2VC problem, we are given the graph G with edge costs, and a budget B; the goal is to find a 2-vertex-connected subgraph H of G with total edge cost at most B that maximizes the number of vertices in H. We describe an $O(\log n \log k)$ approximation for the k-2VC problem, and a bicriteria approximation for the Budget-2VC problem that gives an $O(\frac{1}{\epsilon}\log^2 n)$ approximation, while violating the budget by a factor of at most $3+\epsilon$.

1 Introduction

Connectivity and network design problems play an important role in combinatorial optimization and algorithms both for their theoretical appeal and their usefulness in real-world applications. Many of these problems, such as the well-known minimum cost Steiner tree problem, are NP-hard, and there has been a large and rich literature on approximation algorithms. A number of elegant and powerful techniques and results have been developed over the years. In particular, the primal-dual method [1, 17] and iterated rounding [19] have led to some remarkable results. Occasionally, interesting and useful variants of classical problems are introduced, sometimes motivated by their natural appeal and sometimes motivated by practical applications. One such problem is the k-MST problem introduced by Ravi *et al.* [24]: Given an edge-weighted graph G and an integer k, the goal is to find a minimum-cost subgraph of G that contains at least k vertices. It is not hard to see that the k-MST problem is at least as hard as the Steiner tree problem; moreover an α-approximation for the k-MST problem implies an α-approximation for the Steiner tree problem. The k-MST problem has attracted considerable attention in the approximation algorithms literature and its study has led to several new algorithmic ideas and applications [3, 15, 14, 7, 5]. Closely related to the k-MST problem is the budgeted or Max-Prize Tree problem [21, 5]; here we

* Partially supported by NSF grants CCF 0728782 and CNS-0721899, and a US-Israeli BSF grant 2002276.
†Partially supported by NSF grant CCF 0728782.

FSTTCS 2008
IARCS Annual Conference on
Foundations of Software Technology and Theoretical Computer Science
http://drops.dagstuhl.de/opus/volltexte/2008/1746

are given G and a budget B, and the goal is to find a subgraph H of G of total cost no more than B, that maximizes the number of vertices (or terminals) in H. Interestingly, it is only recently that the rooted version of the Max-Prize Tree problem was shown to have an $O(1)$-approximation [5], although an $O(1)$-approximation was known for the k-MST problem much earlier [6].

Recently, Lau *et al.* [22] considered the natural generalization of k-MST to higher connectivity. In particular they defined the (k, λ)-subgraph problem to be the following: Find a minimum-cost subgraph of the given graph G that contains at least k vertices and is λ-edge connected. We use the notation k-λEC to refer to this problem. In [22, 23] a poly-logarithmic approximation was derived for the k-2EC problem. In this paper, we consider the vertex-connectivity generalizations of the k-MST and Budgeted Tree problems. We define the k-λVC problem as follows: Given an integer k and a graph G with edge costs, find the minimum-cost λ-vertex-connected subgraph of G that contains at least k vertices. Similarly, in the Budget-λVC problem, given a budget B and a graph G with edge costs, the goal is to find a λ-vertex-connected subgraph of G of cost at most B, that maximizes the number of vertices it contains. In particular we focus on $k = 2$ and develop approximation algorithms for both the k-2VC and Budget-2VC problems. We note that the k-λEC problem reduces to the k-λVC problem in an approximation preserving fashion, though the opposite reduction is not known. The k-λEC and k-λVC problems are NP-hard and also APX-hard for any $k \geq 1$. Moreover, Lau *et al.* [22] give evidence that, for large λ, the k-λEC problem is likely to be harder to approximate by relating it to the approximability of the dense k-subgraph problem [12].

Problems such as k-MST, Budget-2VC, k-2VC are partly motivated by applications in network design and related areas where one may want to build low-cost networks including (or servicing) many clients, but there are constraints such as a budget on the network cost, or a minimum quota on the number of clients. Algorithms for these problems also find other uses. For instance, a basic problem in vehicle routing applications is the s-t Orienteering problem in which one seeks an s-t path that maximizes the number of vertices in it subject to a budget B on its length. Approximation algorithms for this problem [5, 4, 11] have been derived through approximation algorithms for the k-MST and the related k-stroll problems; in the latter, the goal is to find a minimum-cost path containing k vertices.

How do we solve these problems? The k-MST problem required several algorithmic innovations which eventally led to the current best approximation ratio of 2 [14]. The main technical tool which underlies $O(1)$ approximations for the k-MST problem [6, 15, 14] is a special property that holds for a LP relaxation of the prize-collecting Steiner tree problem [17] which is a Lagrangian relaxation of the Steiner tree problem. Unfortunately, one cannot use these ideas (at least directly) for more general problems such as k-2VC (or the k-Steiner forest problem [18]) since the LP relaxation for the prize-collecting variant is not known to satisfy the above mentioned property. We therefore rely on alternative techniques that take a more basic approach.

Our algorithms for k-2VC and Budget-2VC use the same high-level idea, relying on the notion of *density*: the density of a subgraph is the ratio of its cost to the number of vertices it contains. The algorithms greedily combine subgraphs of low density until the union of these subgraphs has the desired number of vertices or has cost equal to the budget. They

fail only if we find a subgraph H of good density, but that is far too large. One needs, then, a way to *prune* H to find a smaller subgraph of comparable density. Our main structural result for pruning 2-connected graphs is the following:

THEOREM 1. *Let G be a 2-connected edge-weighted graph with density ρ, and a designated vertex $r \in V(G)$ such that every vertex of G has 2 vertex-disjoint paths to r of total weight/cost at most L. There is a polynomial-time algorithm that, given any integer $k \leq |V(G)|$, finds a 2-connected k-vertex subgraph H of G containing r, of total cost at most $O(\log k)\rho k + 2L$.*

Intuitively, the algorithm of Theorem 1 allows us to find a subgraph of *any* desired size, at the cost of only a logarithmic increase in density. Further, it allows us to require any vertex r to be in the subgraph, and also applies if we are given a *terminal* set S, and the output subgraph must contain k terminals. (In this case, the density of a subgraph is the ratio of its cost to the number of terminals it contains.) In addition, it applies if the terminals/vertices have arbitrary weights, and the density of a subgraph is the ratio of its cost to the sum of the weights of its terminals. All our algorithms apply to these weighted instances, but for simplicity of exposition, we discuss the more restricted unweighted versions throughout. We observe that pruning a tree (a 1-connected graph) is easy and one loses only a constant factor in the density; the theorem above allows one to prune 2-connected graphs. A technical ingredient that we develop is the following theorem: we believe that Theorems 1 and 2 are interesting in their own right and will find other applications besides algorithms for k-2VC and Budget-2VC.

THEOREM 2. *Let G be a 2-vertex-connected graph with edge costs and let $S \subseteq V$ be a set of terminals. Then, there is a simple cycle C containing at least 2 terminals (a non-trivial cycle) such that the density of C is at most the density of G. Moreover, such a cycle can be found in polynomial time.*

Using the above theorem and an LP approach we obtain the following.

COROLLARY 3. *Given a graph $G(V, E)$ with edge costs and ℓ terminals $S \subseteq V$, there is an $O(\log \ell)$ approximation for the problem of finding a minimum-density non-trivial cycle.*

Note that Theorem 2 and Corollary 3 are of interest because we seek a cycle with at least *two* terminals. A minimum-density cycle containing only one terminal can be found by using the well-known min-mean cycle algorithm in directed graphs [2]. We remark, however, that although we suspect that the problem of finding a minimum-density non-trivial cycle is NP-hard, we currently do not have a proof. Theorem 2 shows that the problem is equivalent to the dens-2VC problem, defined in the next section.

Armed with these useful structural results, we give approximation algorithms for both the k-2VC and Budget-2VC problems. Our results in fact hold for the more general versions of these problems where the input also specifies a subset $S \subseteq V$ of *terminals* and the goal is to find subgraphs with the desired number of terminals, or to maximize the number of terminals.[‡]

[‡]For k-2EC and k-λEC , the problem with specified terminal set S can be reduced to the problem where every vertex in V is a terminal. Such a reduction does not seem possible for the k-2VC and k-λVC , so we work directly with the terminal version.

THEOREM 4. *There is an $O(\log \ell \cdot \log k)$ approximation for the k-2VC problem, where ℓ is the number of terminals.*

COROLLARY 5. *There is an $O(\log \ell \cdot \log k)$ approximation for the k-2EC problem, where ℓ is the number of terminals.*

THEOREM 6. *There is a polynomial time bicriteria approximation algorithm for Budget-2VC that, for any $0 < \epsilon \leq 1$, outputs a subgraph of edge-weight $(3+\epsilon)B$ containing $\Omega(\epsilon \cdot \mathrm{OPT}/(\log n \log \mathrm{OPT}))$ terminals, where OPT is the number of terminals in an optimum solution of cost B. For the rooted version, the subgraph has weight at most $(2+\epsilon)B$.*

Most of the proofs are omitted from this version due to space limitations. The reader can find a full version on the websites of the authors.

1.1 Overview of Technical Ideas

We focus on the rooted version of k-2VC : the goal is to find a min-cost subgraph that 2-connects at least k terminals to a specified root vertex r. It is easy to reduce k-2VC to its rooted version. We draw inspiration from algorithmic ideas that led to poly-logarithmic approximations for the k-MST problem.

For a subgraph H that contains r, let $k(H)$ be the number of terminals that are 2-connected to r in H. Then the *density* of H is simply the ratio of the cost of H to $k(H)$. The dens-2VC problem is to find a 2-connected subgraph of minimum density. An $O(\log \ell)$ approximation for the dens-2VC problem (where ℓ is the number of terminals) can be derived in a some what standard way by using a bucketing and scaling trick on a linear programming relaxation for the problem. We exploit the known bound of 2 on the integrality gap of a natural LP for the SNDP problem with vertex connectivity requirements in $\{0, 1, 2\}$ [13].

Our algorithm for k-2VC uses a greedy approach at the high level. We start with an empty subgraph G' and use the approximation algorithm for dens-2VC in an iterative fashion to greedily add terminals to G' until at least $k' \geq k$ terminals are in G'. This approach would yield an $O(\log \ell \log k)$ approximation if $k' = O(k)$. However, the last iteration of the dens-2VC algorithm may add many more terminals than desired with the result that $k' \gg k$. In this case we cannot bound the cost of the solution obtained by the algorithm. To overcome this problem, one can try to *prune* the subgraph H added in the last iteration to only have the desired number of terminals. For the k-MST problem, H is a tree and pruning is quite easy.

Our main technical contribution is Theorem 1, to give a pruning step for the k-2VC problem. To accomplish this, we use two algorithmic ideas. The first is encapsulated in the cycle finding algorithm of Theorem 2. Second, we use this cycle finding algorithm to repeatedly merge subgraphs until we get the desired number of terminals in one subgraph; this latter step requires care. The cycle merging scheme is inspired by a similar approach from the work of Lau *et al.* [22] on the k-2EC problem and in our previous work [11] on the directed orienteering problem. These ideas yield an $O(\log \ell \cdot \log^2 k)$ approximation. We give a modified cycle-merging algorithm with a more sophisticated and non-trivial analysis to obtain an improved $O(\log \ell \cdot \log k)$ approximation.

Some remarks are in order to compare our work to that of [22] on the k-2EC problem. The combinatorial algorithm in [22] is based on finding a low-density cycle or a related structure called a bi-cycle. The algorithm in [22] to find such a structure is incorrect. Further, the cycles are contracted along the way which limits the approach to the k-2EC problem (contracting a cycle in 2-node-connected graph may make the resulting graph not 2-node-connected). In our algorithm we do not contract cycles and instead introduce dummy terminals with weights to capture the number of terminals in an already formed component. This requires us to now address the minimum-density non-trivial simple cycle problem which we do via Theorem 2 and Corollary 3. In independent work, Lau *et al.* [23] obtain a new and correct $O(\log n \log k)$-approximation for k-2EC . They also follow the same approach that we do in using the LP for finding dense subgraphs followed by the pruning step. However, in the pruning step they use a very different approach; they use the sophisticated idea of nowhere-zero 6-flows [25]. Although the use of this idea is elegant, the approach works only for the k-2EC problem, while our approach is less complex and leads to an algorithm for the more general k-2VC problem.

2 The Algorithms for the k-2VC and Budget-2VC Problems

We work with graphs in which some vertices are designated as *terminals*. Henceforth, we use 2-connected graph to mean a 2-vertex-connected graph. Recall that the goal of the k-2VC problem is to find a minimum-cost 2-connected subgraph on at least k terminals. In the rooted k-2VC problem, we wish to find a min-cost subgraph on at least k terminals in which every terminal is 2-connected to the specified root r. The (unrooted) k-2VC problem can be reduced to the rooted version by *guessing* 2 vertices u, v that are in an optimal solution, creating a new root vertex r, and connecting it with 0-cost edges to u and v. It is not hard to show that any solution to the rooted problem in the modified graph can be converted to a solution to the unrooted problem by adding 2 minimum-cost vertex-disjoint paths between u and v. (Since u and v are in the optimal solution, the cost of these added paths cannot be more than OPT.) Similarly, one can reduce Budget-2VC to its rooted version. However, note that adding a min-cost set of paths between the guessed vertices u and v might require us to pay an additional amount of B, so to obtain a solution for the unrooted problem of cost $(3+\epsilon)B$, we must find a solution for the rooted instance of cost $(2+\epsilon)B$.

Note that k-2VC and Budget-2VC are equivalent from the viewpoint of exact optimization, but this is not true from an approximation perspective. Still, we solve them both via the dens-2VC problem, where the goal is to find a subgraph H of minimum density in which all terminals of H are 2-connected to the root. We use the following lemma, which relies on a 2-approximation, via a natural LP for the min-cost 2-connectivity problem, due to Fleischer, Jain and Williamson [13], and some standard techniques.

LEMMA 7. *There is an $O(\log \ell)$-approximation algorithm for the dens-2VC problem, where ℓ is the number of terminals in the given instance.*

We first describe our algorithm for the k-2VC problem. Let OPT be the cost of an optimal solution to the k-2VC instance. We assume knowledge of OPT; this can be dispensed with using standard methods. We pre-process the graph by deleting any terminal that does

not have 2 vertex-disjoint paths to the root r of total cost at most OPT. The high-level description of the algorithm for the rooted k-2VC problem is given below.

$k' \leftarrow k$, G' is the empty graph.
While ($k' > 0$):
  Use the approximation algorithm for dens-2VC to find a subgraph H in G.
  If ($k(H) \leq k'$):
    $G' \leftarrow G' \cup H$, $k' \leftarrow k' - k(H)$.
    Mark all terminals in H as non-terminals.
  Else:
    Prune H to obtain H' that contains k' terminals.
    $G' = G' \cup H'$, $k' \leftarrow 0$.
Output G'.

At the beginning of any iteration of the while loop, the graph contains a solution to the dens-2VC problem of density at most $\frac{\text{OPT}}{k'}$. Therefore, the graph H returned always has density at most $O(\log \ell)\frac{\text{OPT}}{k'}$. If $k(H) \leq k'$, we add H to G' and decrement k'; we refer to this as the *augmentation* step. Otherwise, we have a graph H of good density, but with too many terminals. In this case, we prune H to find a graph with the required number of terminals; this is the *pruning step*. A simple set-cover type argument shows the following lemma:

LEMMA 8. *If, at every augmentation step, we add a graph of density at most* $O(\log \ell)\frac{\text{OPT}}{k'}$ *(where* k' *is the number of additional terminals that must be selected), the total cost of all the augmentation steps is at most* $O(\log \ell \cdot \log k)\text{OPT}$.

Therefore, it remains only to bound the cost of the graph H' added in the pruning step, and Theorem 1, proved in Section 4, is precisely what is needed. Our main result for the k-2VC problem, Theorem 4, follows easily from Lemma 8 and Theorem 1.

We now describe the similar algorithm for the Budget-2VC problem. Given budget B, preprocess the graph as before by deleting vertices that do not have 2 vertex-disjoint paths to r of total cost at most B. Let OPT denote the number of vertices in the optimal solution, and $k = \text{OPT}/c \log \ell \log \text{OPT}$, for some constant $c = O(1/\epsilon)$. We run the same greedy algorithm, using the $O(\log \ell)$-approximation for the dens-2VC problem. Note that at each stage, the graph contains a solution to dens-2VC of density at most $B/(\text{OPT} - k) < 2B/\text{OPT}$. Therefore, we have the following lemma:

LEMMA 9. *If, at every augmentation step of the algorithm for Budget-*2VC*, we add a graph of density at most* $O(\log \ell)(2B/\text{OPT})$*, the total cost of all augmentation steps is at most* $O(B/\log \text{OPT}) \leq \epsilon B$.

Again, to prove Theorem 6, giving a bicriteria approximation for Budget-2VC, we only have to bound the cost of the pruning step.

PROOF OF THEOREM 6. From the previous lemma, the total cost of the augmentation steps is at most ϵB. The graph H returned by the dens-2VC algorithm has density at most $O(\log \ell \cdot B/\text{OPT})$, and $k(H) > k'$ terminals. Now, from Theorem 1, we can prune H to find a graph H' containing k' terminals and cost at most $O(\log k' \log \ell \cdot B/\text{OPT}) \cdot k' + 2B$. As $k' \leq k = \text{OPT}/(c \log \ell \log \text{OPT})$, a suitable choice of c ensures that the total cost of the pruning step is at most $\epsilon B + 2B$.

It remains only to prove Lemma 7, that there is an $O(\log \ell)$-approximation for the dens-2VC problem, and the crucial Theorem 1, bounding the cost of the pruning step. We omit the proof of Lemma 7 from this extended abstract. Before the latter is proved in Section 4, we develop some tools in Section 3; chief among these tools is Theorem 2.

3 Finding Low-density Non-trivial Cycles

A cycle $C \subseteq G$ is *non-trivial* if it contains at least 2 terminals. We define the min-density non-trivial cycle problem: Given a graph $G(V, E)$, with $S \subseteq V$ marked as terminals, edge costs and terminal weights, find a minimum-density cycle that contains at least 2 terminals. Note that if we remove the requirement that the cycle be non-trivial (that is, it contains at least 2 terminals), the problem reduces to the min-mean cycle problem in directed graphs, and can be solved exactly in polynomial time (see [2]). Algorithms for the min-density non-trivial cycle problem are a useful tool for solving the k-2VC and k-2EC problems. In this section, we give an $O(\log \ell)$-approximation algorithm for the minimum-density non-trivial cycle problem.

THEOREM 10. *Let G be a 2-connected graph with at least 2 terminals. G contains a simple non-trivial cycle X such that* $\text{density}(X) \leq \text{density}(G)$.

PROOF SKETCH. Let C be an arbitrary non-trivial simple cycle; such a cycle always exists since G is 2-connected and has at least 2 terminals. If $\mathit{density}(C) > \mathit{density}(G)$, we give an algorithm that finds a new non-trivial cycle C' such that $\mathit{density}(C') < \mathit{density}(C)$. Repeating this process gives us the desired cycle. Let G' be the graph formed by contracting the given cycle C to a single vertex v. In G', v is not a terminal, and so has weight 0. Consider the 2-connected components of G' (each such component is formed by adding v to a connected component of $G' - v$), and pick the one of minimum density. If H is this component, $\mathit{density}(H) < \mathit{density}(G)$ by an averaging argument.

H contains at least 1 terminal. If it contains 2 or more terminals, recursively find a non-trivial cycle C' in H such that $\mathit{density}(C') \leq \mathit{density}(H) < \mathit{density}(C)$. If C' exists in the given graph G, we are done. Otherwise, C' contains v, and the edges of C' form an ear of C in the original graph G. The density of this ear is less than the density of C, and we can find a non-trivial cycle in the union of C and the ear of density at most that of G.

Finally, if H has exactly 1 terminal u, find any 2 vertex-disjoint paths using edges of H from u to distinct vertices in the cycle C. (Since G is 2-connected, there always exist such paths.) The cost of these paths is at most $\mathit{cost}(H)$, and concatenating these 2 paths corresponds to an ear of C in G. The density of this ear is less than $\mathit{density}(C)$; again, the union of the ear and C has a desired non-trivial cycle. ∎

We remark that the algorithm of Theorem 10 does not lead to a polynomial-time algorithm, even if all edge costs and terminal weights are polynomially bounded. We give a strongly polynomial time algorithm to find such a cycle in the full version of this paper. Note that neither of these algorithms may directly give a good approximation to the min-density non-trivial cycle problem, because the optimal non-trivial cycle may have density much less than that of G. However, we can use Theorem 10 to prove the following theorem:

THEOREM 11. *There is an α-approximation to the (unrooted) dens-2VC problem if and only if there is an α-approximation to the problem of finding a minimum-density non-trivial cycle.*

Theorem 11 and Lemma 7 imply an $O(\log \ell)$-approximation for the minimum-density non-trivial cycle problem; this proves Corollary 3.

4 Pruning 2-connected Graphs of Good Density

In this section, we prove Theorem 1. We are given a graph G and $S \subseteq V$, a set of at least k terminals. Further, every terminal in G has 2 vertex-disjoint paths to the root r of total cost at most L. Let ℓ be the number of terminals in G, and $cost(G)$ its total cost; $\rho = \frac{cost(G)}{\ell}$ is the density of G. We describe an algorithm that finds a subgraph H of G that contains at least k terminals, each of which is 2-connected to the root, and of total edge cost $O(\log k)\rho k + 2L$.

We can assume $\ell > (8 \log k) \cdot k$, or the trivial solution of taking the entire graph G suffices. The main phase of our algorithm proceeds by maintaining a set of 2-connected subgraphs that we call *clusters*, and repeatedly finding low-density cycles that merge clusters of similar weight to form larger clusters. (The weight of a cluster X, denoted by w_X, is (roughly) the number of terminals it contains.) Clusters are grouped into *tiers* by weight; tier i contains clusters with weight at least 2^i and less than 2^{i+1}. Initially, each terminal is a separate cluster in tier 0. We say a cluster is *large* if it has weight at least k, and *small* otherwise. The algorithm stops when most terminals are in large clusters.

We now describe the algorithm MERGECLUSTERS (see next page). To simplify notation, let α be the quantity $2\lceil \log k \rceil \rho$. We say that a cycle is *good* if it has density at most α; that is, good cycles have density at most $O(\log k)$ times the density of the input graph.

MERGECLUSTERS:
For (each i in $\{0, 1, \ldots, (\lceil \log_2 k \rceil - 1)\}$) do:
  If ($i = 0$):
    Every terminal has weight 1
  Else:
    Mark all vertices as non-terminals
    For (each small 2-connected cluster X in tier i) do:
      Add a (dummy) terminal v_X to G of weight w_X
      Add (dummy) edges of cost 0 from v_X to two (arbitrary) distinct vertices of X
  While (G has a non-trivial cycle C of density at most $\alpha = 2\lceil \log k \rceil \rho$):
    Let $X_1, X_2, \ldots X_q$ be the small clusters that contain a terminal **or an edge** of C.
    (Note that the terminals in C belong to a subset of $\{X_1, \ldots X_q\}$.)
    Form a new cluster Y (of a higher tier) by merging the clusters $X_1, \ldots X_q$
    $w_Y \leftarrow \sum_{j=1}^{q} w_{X_j}$
    If ($i = 0$):
      Mark all terminals in Y as non-terminals
    Else:
      Delete all (dummy) terminals in Y and the associated (dummy) edges.

We briefly remark on some salient features of this algorithm and our analysis before presenting the details of the proofs.

1. In iteration i, terminals correspond to tier i clusters. Clusters are 2-connected subgraphs of G, and by using cycles to merge clusters, we preserve 2-connectivity as the

clusters become larger.

2. When a cycle C is used to merge clusters, all small clusters that contain an edge of C (regardless of their tier) are merged to form the new cluster. Therefore, at any stage of the algorithm, all currently small clusters are edge-disjoint. Large clusters, on the other hand, are *frozen*; even if they intersect a good cycle C, they are not merged with other clusters on C. Thus, at any time, an edge may be in multiple large clusters and up to one small cluster.
3. In iteration i of MERGECLUSTERS, the density of a cycle C is only determined by its cost and the weight of terminals in C corresponding to tier i clusters. Though small clusters of other (lower or higher) tiers might be merged using C, we do *not* use their weight to pay for the edges of C.
4. The ith iteration terminates when no good cycles can be found using the remaining tier i clusters. At this point, there may be some terminals remaining that correspond to clusters which are not merged to form clusters of higher tiers. However, our choice of α (which defines the density of good cycles) is such that we can bound the number of terminals that are "left behind" in this fashion. Therefore, when the algorithm terminates, most terminals are in large clusters.

We prove that after MERGECLUSTERS terminates most terminals are in large clusters and that each large cluster has good density. The proof proceeds via several properties that we establish formally.

Remarks: Throughout the algorithm, the graph G is always 2-connected. The weight of a cluster is at most the number of terminals it contains.

LEMMA 12. *The clusters formed by* MERGECLUSTERS *are all* 2*-connected.*

LEMMA 13. *The total weight of small clusters in tier i that are not merged to form clusters of higher tiers is at most $\frac{\ell}{2\lceil \log k \rceil}$.*

COROLLARY 14. *When the algorithm* MERGECLUSTERS *terminates, the total weight of large clusters is at least $\ell/2 > (4 \log k) \cdot k$.*

So far, we have shown that most terminals reach large clusters, all of which are 2-connected, but we have not argued about the density of these clusters. The next lemma says that if we can find a large cluster of good density, we can find a solution to the k-2VC problem of good density.

LEMMA 15. *Let Y be a large cluster formed by* MERGECLUSTERS*. If Y has density at most δ, we can find a graph Y' with at least k terminals, each of which is* 2*-connected to r, of total cost at most $2\delta k + 2L$.*

The following lemma allows us to show that every large cluster has density at most $O(\log^2 k)\rho$.

LEMMA 16. *For any cluster Y formed by* MERGECLUSTERS *during iteration i, the total cost of edges in Y is at most $(i+1) \cdot \alpha w_Y$.*

Let Y be an arbitrary large cluster; since we have only $\lceil \log k \rceil$ tiers, the previous lemma implies that the cost of Y is at most $\lceil \log k \rceil \cdot \alpha w_Y = O(\log^2 k)\rho w_Y$. That is, the density of Y

is at most $O(\log^2 k)\rho$, and we can use this fact together with Lemma 15 to find a solution to the rooted k-2VC problem of cost at most $O(\log^2 k)\rho k + 2L$. This completes the 'weaker' analysis, but this does not suffice to prove Theorem 1; to prove the theorem, we would need to use a large cluster Y of density $O(\log k)\rho$, instead of $O(\log^2 k)\rho$.

For the purpose of the more careful analysis, implicitly construct a forest $\mathcal{F}$ on the clusters formed by MERGECLUSTERS. Initially, the vertex set of $\mathcal{F}$ is just S, the set of terminals, and $\mathcal{F}$ has no edges. Every time a cluster Y is formed by merging $X_1, X_2, \ldots X_q$, we add a corresponding vertex Y to the forest $\mathcal{F}$, and add edges from Y to each of $X_1, \ldots X_q$; Y is the parent of $X_1, \ldots X_q$. We also associate a cost with each vertex in $\mathcal{F}$; the cost of the vertex Y is the cost of the cycle used to form Y from $X_1, \ldots X_q$. We thus build up trees as the algorithm proceeds; the root of any tree corresponds to a cluster that has not yet become part of a bigger cluster. The leaves of the trees correspond to vertices of G; they all have cost 0. Also, a large cluster Y formed by the algorithm is at the root of its tree; we refer to this tree as T_Y.

For each large cluster Y after MERGECLUSTERS terminates, say that Y is of type i if Y was formed during iteration i of MergeClusters. We now define the *final-stage* clusters of Y: They are the clusters formed during iteration i that became part of Y. (We include Y itself in the list of final-stage clusters; even though Y was formed in iteration i of MERGECLUSTERS, it may contain other final-stage clusters. For instance, during iteration i, we may merge several tier i clusters to form a cluster X of tier $j > i$. Then, if we find a good-density cycle C that contains an edge of X, X will merge with the other clusters of C.) The *penultimate* clusters of Y are those clusters that exist just before the beginning of iteration i and become a part of Y. Equivalently, the penultimate clusters are those formed before iteration i that are the immediate children in T_Y of final-stage clusters. Figure 1 illustrates the definitions of final-stage and penultimate clusters.

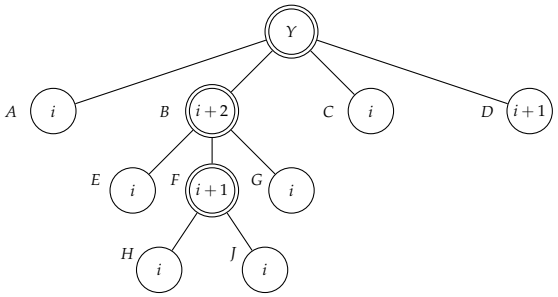


Figure 1: A part of the Tree T_Y corresponding to Y, a large cluster of type i. The number in each vertex indicates the tier of the cluster. Only final-stage and penultimate clusters are shown: final-stage clusters are shown with a double circle; the rest are penultimate.

An edge of a large cluster Y is said to be a *final edge* if it is used in a cycle C that produces a final-stage cluster of Y. All other edges of Y are called *penultimate edges*; note that any penultimate edge is in some penultimate cluster of Y. We define the *final cost* of Y to be the sum of the costs of its final edges, and its *penultimate cost* to be the sum of the costs of its penultimate edges; clearly, the cost of Y is the sum of its final and penultimate costs. We bound the final costs and penultimate costs separately.

Recall that an edge is a final edge of a large cluster Y if it is used by MERGECLUSTERS to

form a cycle C in the final iteration during which Y is formed. The reason we can bound the cost of final edges is that the cost of any such cycle is at most α times the weight of clusters contained in the cycle, and a cluster does not contribute to the weight of more than one cycle in an iteration. (This is also the essence of Lemma 16.) We formalize this intuition below.

LEMMA 17. *The final cost of a large cluster Y is at most αw_Y, where w_Y is the weight of Y.*

LEMMA 18. *If Y_1 and Y_2 are distinct large clusters of the same type, no edge is a penultimate edge of both Y_1 and Y_2.*

THEOREM 19. *After* MERGECLUSTERS *terminates, at least one large cluster has density at most $O(\log k)\rho$.*

PROOF. We define the *penultimate density* of a large cluster to be the ratio of its penultimate cost to its weight. Consider the total penultimate costs of all large clusters: For any i, each edge $e \in E(G)$ can be a penultimate edge of at most 1 large cluster of type i. This implies that each edge can be a penultimate edge of at most $\lceil \log k \rceil$ clusters. Therefore, the sum of penultimate costs of all large clusters is at most $\lceil \log k \rceil cost(G)$. Further, the total weight of all large clusters is at least $\ell/2$. Therefore, the (weighted) average penultimate density of large clusters is at most $2\lceil \log k \rceil \frac{cost(G)}{\ell} = 2\lceil \log k \rceil \rho$, and hence there exists a large cluster Y of penultimate density at most $2\lceil \log k \rceil \rho$. The penultimate cost of Y is, therefore, at most $2\lceil \log k \rceil \rho w_Y$, and from Lemma 17, the final cost of Y is at most αw_Y. Therefore, the density of Y is at most $\alpha + 2\lceil \log k \rceil \rho = O(\log k)\rho$. ∎

Theorem 19 and Lemma 15 together imply that we can find a solution to the rooted k-2VC problem of cost at most $O(\log k)\rho k + 2L$. This completes our proof of Theorem 1.

Acknowledgments: We thank Mohammad Salavatipour for helpful discussions on k-2EC and related problems. We thank Erin Wolf Chambers for useful suggestions on notation.

References

[1] A. Agrawal, P. N. Klein, and R. Ravi. When trees collide: An Approximation Algorithm for the Generalized Steiner Problem on Networks. *SIAM J. on Computing*, 24(3):440–456, 1995.

[2] R. Ahuja, T. Magnanti, and J. Orlin. Network Flows: Theory, Algorithms, and Applications. Prentice Hall, Upper Saddle River, NJ, 1993

[3] B. Awerbuch, Y. Azar, A. Blum and S. Vempala. New Approximation Guarantees for Minimum Weight k-Trees and Prize-Collecting Salesmen. *SIAM J. on Computing*, 28(1):254–262, 1999. Preliminary version in *Proc. of ACM STOC*, 1995.

[4] N. Bansal, A. Blum, S. Chawla, and A. Meyerson. Approximation Algorithms for Deadline-TSP and Vehicle Routing with Time-Windows. *Proc. of ACM STOC*, 166–174, 2004.

[5] A. Blum, S. Chawla, D. Karger, T. Lane, A. Meyerson, and M. Minkoff. Approximation Algorithms for Orienteering and Discounted-Reward TSP. *SIAM J. on Computing*, 37(2):653–670, 2007.

[6] A. Blum, R. Ravi and S. Vempala. A Constant-factor Approximation Algorithm for the k-MST Problem. *J. of Computer and System Sciences*, 58:101–108, 1999.

[7] K. Chaudhuri, B. Godfrey, S. Rao, and K. Talwar. Paths, trees, and minimum latency tours. *Proc. of IEEE FOCS*, 36–45, 2003.

[8] C. Chekuri, G. Even, A. Gupta, and D. Segev. Set Connectivity Problems in Undirected Graphs and the Directed Steiner Network Problem. *Proc. of ACM-SIAM SODA*, 532–541, 2008.

[9] C. Chekuri, M. T. Hajiaghayi, G. Kortsarz, and M. R. Salavatipour. Approximation algorithms for Non-uniform Buy-at-bulk Network Design. *Proc. of IEEE FOCS*, 677–686, 2006.

[10] C. Chekuri, M. T. Hajiaghayi, G. Kortsarz, and M. R. Salavatipour. Approximation Algorithms for Node-weighted Buy-at-bulk Network Design. *Proc. of ACM-SIAM SODA*, 1265–1274, 2007.

[11] C. Chekuri, N. Korula, and M. Pál. Improved Algorithms for Orienteering and Related Problems. *Proc. of ACM-SIAM SODA*, 661–670, 2008.

[12] U. Feige, G. Kortsarz and D. Peleg. The Dense k-Subgraph Problem. *Algorithmica*, 29(3):410–421, 2001. Preliminary version in *Proc. of IEEE FOCS*, 1993.

[13] L. Fleischer, K. Jain, D. P. Williamson. Iterative Rounding 2-approximation Algorithms for Minimum-cost Vertex Connectivity Problems. *J. of Computer and System Sciences*, 72(5):838–867, 2006.

[14] N. Garg. Saving an ϵ: A 2-approximation for the k-MST Problem in Graphs. *Proc. of ACM STOC*, 396–402, 2005.

[15] N. Garg. A 3-approximation for the Minimum Tree Spanning k Vertices. *Proc. of IEEE FOCS*, 302–309, 1996.

[16] M. X. Goemans and D. P. Williamson. A General Approximation Technique for Constrained Forest Problems. *SIAM J. on Computing*, 24(2):296–317, 1995.

[17] M. X. Goemans and D. P. Williamson. The Primal-Dual method for Approximation Algorithms and its Application to Network Design Problems. In D. S. Hochbaum, editor, *Approximation Algorithms for NP-Hard Problems*. PWS Publishing Company, 1996.

[18] M. T. Hajiaghayi and K. Jain. The Prize-Collecting Generalized Steiner Tree Problem via a New Approach of Primal-Dual Schema. *Proc of ACM-SIAM SODA*, 631–640, 2006.

[19] K. Jain. A Factor 2 Approximation Algorithm for the Generalized Steiner Network Problem *Combinatorica*, 21(1):39–60, 2001.

[20] D. S. Johnson. Approximation Algorithms for Combinatorial Problems. *J. of Computer and System Sciences*, 9(3):256–278, 1974.

[21] D. S. Johnson, M. Minkoff, and S. Phillips. The Prize Collecting Steiner Tree Problem: Theory and Practice. *Proc. of ACM-SIAM SODA*, 760–769, 2000.

[22] L.C. Lau, J. Naor, M. Salavatipour and M. Singh. Survivable Network Design with Degree or Order Constraints. *Proc. of ACM STOC*, 2007.

[23] L.C. Lau, J. Naor, M. Salavatipour and M. Singh. Survivable Network Design with Degree or Order Constraints. To Appear in *SIAM J. on Computing*.

[24] R. Ravi, R. Sundaram, M. Marathe, D. Rosenkrantz, and S. Ravi. Spanning trees short and small. *SIAM J. Disc. Math.* 9 (2): 178–200, 1996.

[25] P. D. Seymour. Nowhere-zero 6-flows. *J. Comb. Theory B*, 30: 130–135, 1981.

Foundations of Software Technology and Theoretical Computer Science (Bangalore) 2008.
Editors: R. Hariharan, M. Mukund, V. Vinay; pp 131-142

Single-Sink Network Design with Vertex Connectivity Requirements
(Extended Abstract)

Chandra Chekuri[*], **Nitish Korula**[†]
Dept. of Computer Science
University of Illinois
Urbana, IL 61801
{chekuri,nkorula2}@cs.uiuc.edu

ABSTRACT. We study single-sink network design problems in undirected graphs with vertex connectivity requirements. The input to these problems is an edge-weighted undirected graph $G = (V, E)$, a sink/root vertex r, a set of terminals $T \subseteq V$, and integer k. The goal is to connect each terminal $t \in T$ to r via k *vertex-disjoint* paths. In the *connectivity* problem, the objective is to find a min-cost subgraph of G that contains the desired paths. There is a 2-approximation for this problem when $k \leq 2$ [9] but for $k \geq 3$, the first non-trivial approximation was obtained in the recent work of Chakraborty, Chuzhoy and Khanna [4]; they describe and analyze an algorithm with an approximation ratio of $O(k^{O(k^2)} \log^4 n)$ where $n = |V|$.
In this paper, inspired by the results and ideas in [4], we show an $O(k^{O(k)} \log |T|)$-approximation bound for a simple greedy algorithm. Our analysis is based on the dual of a natural linear program and is of independent technical interest. We use the insights from this analysis to obtain an $O(k^{O(k)} \log |T|)$-approximation for the more general single-sink *rent-or-buy* network design problem with vertex connectivity requirements. We further extend the ideas to obtain a poly-logarithmic approximation for the single-sink *buy-at-bulk* problem when $k = 2$ and the number of cable-types is a fixed constant; we believe that this should extend to any fixed k. We also show that for the non-uniform buy-at-bulk problem, for each fixed k, a small variant of a simple algorithm suggested by Charikar and Kargiazova [5] for the case of $k = 1$ gives an $2^{O(\sqrt{\log |T|})}$ approximation for larger k. These results show that for each of these problems, simple and natural algorithms that have been developed for $k = 1$ have good performance for small $k > 1$.

1 Introduction

We consider several *single-sink* network design problems with *vertex connectivity* requirements. Let $G = (V, E)$ be a given undirected graph on n nodes with a specified sink/root vertex r and a subset of terminals $T \subseteq V$, with $|T| = h$. Each terminal t has a demand $d_t > 0$ that needs to be routed to the root along k vertex-disjoint paths (d_t is sent on each of the k paths). In the following discussion, we assume for simplicity that $d_t = 1$ for each terminal t. The goal in all the problems is to find a routing (a selection of paths) for the terminals so as to minimize the cost of the routing. We obtain problems of increasing generality and complexity based on the cost function on the edges. In the basic connectivity problem, each edge e has a non-negative cost c_e, and the objective is to find a minimum-cost subgraph H

[*] Partially supported by NSF grants CCF 0728782 and CNS 0721899, and a US-Israeli BSF grant 2002276.
[†] Partially supported by NSF grant CCF 0728782.

FSTTCS 2008
IARCS Annual Conference on
Foundations of Software Technology and Theoretical Computer Science
http://drops.dagstuhl.de/opus/volltexte/2008/1747

of G that contains the desired disjoint paths for each terminal. We then consider generalizations of the connectivity problem where the cost of an edge depends on the number of terminals whose paths use it. In the rent-or-buy problem there is a parameter M with the following interpretation: An edge e can either be bought for a cost of $c_e \cdot M$, in which case any number of terminals can use it, or e can be rented at the cost of c_e per terminal. In other words, the cost of an edge e is $c_e \cdot \min\{M, |T_e|\}$ where T_e is the set of terminals whose paths use e. In the uniform buy-at-bulk problem, the cost of an edge e is $c_e \cdot f(|T_e|)$ for some given sub-additive function $f : R^+ \to R^+$. In the non-uniform buy-at-bulk problem the cost of an edge e is $f_e(|T_e|)$ for some edge-dependent sub-additive function $f_e : R^+ \to R^+$. All of the above problems are NP-hard and also APX-hard to approximate even for $k = 1$. Note that when $k = 1$ the connectivity problem is the well-known Steiner tree problem. In this paper we focus on polynomial-time approximation algorithms for the above network design problem when $k > 1$. We refer to the above three problems as SS-k-CONNECTIVITY, SS-k-RENT-OR-BUY and SS-k-BUY-AT-BULK respectively.

Motivation: Our work is motivated by several considerations. First, connectivity and network design problems are of much interest in algorithms and combinatorial optimization. A very general problem in this context is the survivable network design problem (SNDP). An instance of SNDP consists of an edge-weighted graph $G = (V, E)$ and an integer connectivity requirement r_{uv} for each pair of nodes uv. The goal is to find a minimum-cost subgraph H of G such that H contains r_{uv} disjoint paths between u and v for each pair uv. EC-SNDP refers to the variant in which the paths are required only to be edge-disjoint and VC-SNDP refers to the variant where the paths are required to be vertex-disjoint. SNDP captures many connectivity problems as special cases. Jain's [13] seminal work on iterated rounding showed a 2-approximation for EC-SNDP, improving previous results [18]. This was extended to element-connectivity SNDP and to VC-SNDP when $r_{uv} \in \{0, 1, 2\}$ [9]. An important question is to understand the approximability of VC-SNDP when the connectivity requirements exceed 2.

Kortsarz, Krauthgamer and Lee [14] showed that VC-SNDP is hard to approximate to within a factor of $2^{\log^{1-\epsilon} n}$ even when $r_{uv} \in \{0, k\}$ for all uv. However, the hardness requires k to be n^δ for some constant $\delta > 0$; in this same setting they show that SS-k-CONNECTIVITY is hard to approximate to within $\Omega(\log n)$ factor. A natural question to ask is whether SS-k-CONNECTIVITY and more generally VC-SNDP admits a good approximation when k (or in general, the maximum requirement) is small. This question is quite relevant from a practical and theoretical perspective. In fact, no counterexample is known to the possibility of iterated rounding yielding a ratio of $\max_{uv} r_{uv}$ for VC-SNDP (see [9] for more on this). Although there is a 2-approximation for VC-SNDP when $\max_{uv} r_{uv} \leq 2$, until very recently there was no non-trivial (that is, $o(|T|)$) approximation for SS-k-CONNECTIVITY even when $k = 3$! Chakraborty, Chuzhoy and Khanna [4] developed some fundamental new insights in recent work and showed an $O(k^{O(k^2)} \log^4 n)$-approximation for SS-k-CONNECTIVITY via the setpair relaxation; we mention other relevant results from [4] later. Our paper is inspired by the results and ideas in [4]. We show that a simple greedy algorithm yields an improved approximation for SS-k-CONNECTIVITY. Perhaps of equal importance is our analysis, which is based on the dual of the linear programming relaxation. This new dual-based perspective

allows us to analyze simple algorithms for the more complex problems SS-k-RENT-OR-BUY and SS-k-BUY-AT-BULK.

Another motivation for these problems comes from the buy-at-bulk network design problem [17]; this arises naturally in the design of telecommunication networks [17, 1, 6]. Economies of scale imply that bandwidth on a link can be purchased in integer units of different *cable-types*; that is, there are some b cable-types with capacities $u_1 < u_2 < \ldots < u_b$ and costs $w_1 < w_2 < \ldots < w_b$ such that $w_1/u_1 > \ldots > w_b/u_b$. Antonakapoulos *et al.* [2], motivated by real-world fault-tolerant models in optical network design [6] introduced the *protected* buy-at-bulk network design problem. In [2] this problem was reduced to the corresponding single-sink problem at the expense of a poly-logarithmic ratio in the approximation. An $O(1)$ approximation for the single-sink problem was derived in [2], however, the techniques in [2] were applicable only to the case of a single-cable. An open question raised in [2] is whether one can find a good approximation for the single-sink problem even for the case of two cable-types. In this paper we show that natural and simple algorithms can be obtained for this problem for any fixed number of cable-types. We also analyze a simple randomized greedy inflation algorithm (suggested by Charikar and Kargiazova [5] for $k = 1$) for the non-uniform buy-at-bulk problem and show that it achieves a non-trivial approximation for each fixed k. Our starting point for the buy-at-bulk problem is the rent-or-buy cost function which can be modeled with two cable-types, one with unit capacity and the other with essentially infinite capacity. This simple cost function, in addition to its inherent interest, has played an important role in the development of algorithms for several problems [12].

Results and Technical Contributions: We analyze simple combinatorial algorithms for the three single-sink vertex-connectivity network design problems that we described. We prove bounds on the approximation ratio of the algorithms using the dual of natural LP relaxations; the LP relaxation is used only for the analysis. This leads to the following results:

- An $O(k^{2k} \log |T|)$ approximation for SS-k-CONNECTIVITY.
- An $O(k^{2k} \log |T|)$ approximation for SS-k-RENT-OR-BUY.
- An $O((\log |T|)^{O(b)})$ approximation for the SS-k-BUY-AT-BULK with b cable-types when $k = 2$.
- A $2^{O(\sqrt{\log h})}$ approximation for the non-uniform SS-k-BUY-AT-BULK for each fixed k.

Our result for SS-k-CONNECTIVITY improves the ratio of $O(k^{O(k^2)} \log^4 n)$ from [4]. For the SS-k-RENT-OR-BUY problem, ours is the first non-trivial result for any $k \geq 2$. For the SS-k-BUY-AT-BULK problem, an $O(1)$ approximation is known for $k = 2$ in the single-cable setting, but no non-trivial algorithm was known even for the case of $k = 2$ with two or more cables. Some other results can be derived from the above. Following the observation in [4], the SS-k-CONNECTIVITY approximation ratio applies also to the subset-k-connectivity problem; here the objective is to find a min-cost subgraph such that T is k-connected. It is also easy to see that the approximation ratio only worsens by a factor of k if the terminals have different connectivity requirements in $\{1, 2, \ldots, k\}$. For $k = 2$, our algorithms for rent-or-buy and buy-at-bulk can be used to obtain algorithms for the multicommodity setting using the ideas in [2].

Our algorithms are natural extensions of known combinatorial algorithms for the $k = 1$ case. For SS-k-CONNECTIVITY a (online) greedy algorithm is to order the terminals arbitrar-

ily and add terminals one by one while maintaining a feasible solution for the current set of terminals. This greedy algorithm gives an $O(\log |T|)$ approximation for the Steiner tree problem which is the same as SS-k-CONNECTIVITY when $k = 1$. However, it can be shown easily that this same algorithm, and in fact any deterministic online algorithm, can return solutions of value $\Omega(|T|)\text{OPT}$ even for $k = 2$. Interestingly, we show that a small variant that applies the greedy strategy in *reverse* yields a good approximation ratio! For SS-k-RENT-OR-BUY, our algorithm is a straightforward generalization of the simple random-marking algorithm of Gupta *et al.* [12] for $k = 1$. Our algorithm for SS-k-BUY-AT-BULK is also based on a natural clustering strategy previously used for $k = 1$. We remark that the hardness results of [14] imply that the approximation ratio has to depend on k in some form. The exponential dependence on k is an artifact of the analysis. In particular, we extend a combinatorial lemma from [4]; we believe that the analysis of this lemma can be tightened to show a polynomial dependence on k. Some very recent work [8] achieves results in this direction; see the end of this section for more on this subject.

Although the algorithms are simple and easy extensions of the known algorithms for $k = 1$, the analysis requires several new sophisticated ideas even for $k = 2$. The main technical difference between $k = 1$ and $k > 1$ is the following. For $k = 1$, metric methods can be used since the problem remains unchanged even if we take the metric closure of the given graph G. However this fails for $k > 1$ in a fundamental way. Chakaraborty, Chuzhoy and Khanna [4] developed new insights for $k > 1$. Unfortunately we are unable to elaborate on their ideas due to space limitations. We do mention that they use a *primal* approach wherein they use an optimal fractional solution to argue about the costs of connecting a terminal t to other terminals via disjoint paths. Our analysis is different and is based on analyzing the *dual* of a natural linear programming relaxation. This is inspired by the dual-packing arguments that have been used earlier for connectivity problems. These prior arguments were for $k = 1$, where distance-based arguments via balls grown around terminals can be used. For $k \geq 2$ these arguments do not apply. Nevertheless, we show the effectiveness of the dual-packing approach by using non-uniform balls.

Due to space limitations we defer discussion of the large literature on network design and related work to a full version of the paper. We refer the reader to [15] for a recent survey and to [4, 8]. Chuzhoy and Khanna [8] have independently and concurrently obtained results for SS-k-CONNECTIVITY; they obtain an $O(k \log |T|)$-approximation with edge-costs, and an $O(k^7 \log^2 n)$-approximation with vertex-costs. Their result for SS-k-CONNECTIVITY has a much better dependence on k than ours. Our dual-based analysis differs from their analysis, and is crucial to our algorithms for SS-k-RENT-OR-BUY and SS-k-BUY-AT-BULK which are not considered in [8].

We omit all proofs and many technical details in this extended abstract. The reader can find a longer version on the websites of the authors.

2 Connectivity

In this section we analyze a simple reverse greedy algorithm for SS-k-CONNECTIVITY. Formally, the input to the problem is an edge-weighted graph $G = (V, E)$, an integer k, a specified root vertex r, and a set of terminals $T \subseteq V$. The goal is to find a min-cost edge-

induced subgraph H of G such that H contains k vertex-disjoint paths from each terminal t to r.

The key concept is that of augmentation. Let $T' \subseteq T$ be a subset of terminals and let H' be a subgraph of G that is feasible for T'. For a terminal $t \in T \setminus T'$, a set of k paths $p_1, \ldots, p_k$ is said to be an augmentation for t with respect to T' if (i) p_i is a path from t to some vertex in $T' \cup \{r\}$ (ii) the paths are internally vertex disjoint and (iii) a terminal $t' \in T'$ is the endpoint of at most one of the k paths. Note that the root is allowed to be the endpoint of more than one path. The following proposition is easy to show via a simple min-cut argument.

PROPOSITION 1. *If $p_1, p_2, \ldots, p_k$ is an augmentation for t with respect to T' and H' is a feasible solution for T' then $H \cup (\bigcup_i p_i)$ is a feasible solution for $T' \cup \{t\}$.*

Given T' and t, the augmentation cost of t with respect to T' is the cost of a min-cost set of paths that augment t w.r.t. to T'. If T' is not mentioned, we implicitly assume that $T' = T \setminus \{t\}$. With this terminology and Proposition 1, it is easy to see that the algorithm below finds a feasible solution.

REVERSE-GREEDY:
Let $t \in T$ be a terminal of minimum augmentation cost.
Recursively solve the instsance of SS-k-CONNECTIVITY on G, with terminal set $T' = T - \{t\}$.
Augment t with respect to T', paying (at most) its augmentation cost.

The rest of the section is devoted to showing that REVERSE-GREEDY achieves a good approximation. As we mentioned already, there is an $\Omega(|T|)$ lower bound on the performance of any online algorithm. Thus, the order of terminals is of considerable importance in the performance of the greedy algorithm. Note that for $k = 1$, namely the Steiner tree problem, the greedy online algorithm does have a performance ratio of $O(\log |T|)$.

The key step in the analysis of the algorithm is to bound the augmentation cost of terminals. We do this by constructing a natural linear program for the problem and using a dual-based argument. The primal and its dual linear programs for SS-k-CONNECTIVITY are shown below. We remark that our linear program is based on a path-formulation unlike the standard cut-based (setpair) formulation for VC-SNDP [10, 9]. However, the optimal solution values of the two relaxations are the same. The path-formulation is more appropriate for our analysis.

In the primal linear program below, and throughout the paper, we let $\mathcal{P}_t^k$ denote the collection of all sets of k vertex-disjoint paths from t to the root r. We use the notation $\vec{P}$ to abbreviate $\{p_1, p_2, \ldots p_k\}$, an unordered set of k disjoint paths in $\mathcal{P}_t^k$. Finally, we say that an edge $e \in \vec{P}$ if there is some $p_j \in \vec{P}$ such that $e \in p_j$. In the LP, the variable x_e indicates whether or not the edge e is in the solution. For each $\vec{P} \in \mathcal{P}_t^k$, the variable $f_{\vec{P}}$ is 1 if terminal t selects the k paths of $\vec{P}$ to connect to the root, and 0 otherwise.

Primal-Conn

$$\min \sum_{e \in E} c_e x_e$$

$$\sum_{\vec{P} \in \mathcal{P}_t^k} f_{\vec{P}} \geq 1 \qquad (\forall t \in T)$$

$$\sum_{\vec{P} \in \mathcal{P}_t^k | e \in \vec{P}} f_{\vec{P}} \leq x_e \qquad (\forall t \in T, e \in E)$$

$$x_e, f_{\vec{P}} \in [0,1]$$

Dual-Conn

$$\max \sum_{t \in T} \alpha_t$$

$$\sum_t \beta_e^t \leq c_e \qquad (\forall e \in E)$$

$$\alpha_t \leq \sum_{e \in \vec{P}} \beta_e^t \qquad \left(\forall \vec{P} \in \mathcal{P}_t^k\right)$$

$$\alpha_t, \beta_t^e \geq 0$$

The value $f_{\vec{P}}$ can be thought of as the amount of "flow" sent from t to the root along the set of paths in $\vec{P}$. The first constraint requires that for each terminal, a total flow of at least 1 unit must be sent along various sets of k disjoint paths. Our analysis of the algorithm REVERSE-GREEDY is based on the following technical lemma.

LEMMA 2. *Given an instance of* SS-k-CONNECTIVITY *with h terminals, let* OPT *be the cost of an optimal fractional solution to* ***Primal-Conn****. For each terminal t, let $Cost(t)$ denote the augmentation cost of t. Then $\min_t Cost(t) \leq f(k)k^2 \cdot \frac{\text{OPT}}{h}$ where $f(k) = 3^k k!$. It also follows that $\sum_t Cost_t \leq 2f(k)k^2 \log h \cdot \text{OPT}$.*

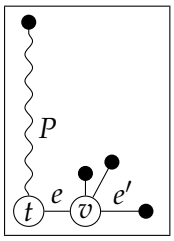
Lemma 2 and a simple inductive proof give the following theorem.

THEOREM 3. REVERSE-GREEDY *is an $O(f(k)k^2 \log h)$-approximation for* SS-k-CONNECTIVITY.

2.1 Overview of the Dual-Packing Analysis

We prove Lemma 2 based on a dual-packing argument. In order to do this we first interpret the variables and constraints in **Dual-Conn**. There is a dual variable α_t for each $t \in T$. We interpret α_t as the total cost that t is willing to pay to connect to the root. In addition there is a variable β_e^t which is the amount that t is willing to pay on edge e. The dual constraint $\sum_t \beta_e^t \leq c_e$ requires that the total payment on an edge from all terminals is at most c_e. In addition, for each terminal t, the total payment α_t should not exceed the min-cost k-disjoint paths to the root with costs given by the β_e^t payments of t on the edges.

Let $\alpha = \min_t Cost(t)$. To prove Lemma 2 it is sufficient to exhibit a feasible setting for the dual variables in which $\alpha_t \geq \alpha/(f(k)k^2)$. How do we do this? To understand the overall plan and intuition, we first consider the Steiner tree problem (the case of $k = 1$). In this case, $\alpha = \min_t Cost(t)$ is the shortest distance between any two terminals. For each t consider the ball of radius $\alpha/2$ centered around t; these balls are *disjoint*. Hence, setting $\alpha_t = \alpha/2$ and $\beta_e^t = c_e$ for each e in t's ball (and $\beta_e^t = 0$ for other edges) yields a feasible dual solution. This interpretation is well-known and underlies the $O(\log |T|)$ bound on the competitiveness of the greedy algorithm for the online Steiner tree problem. Extending the above intuition to $k > 1$ is substantially more complicated. We again to wish to define balls of radius $\Omega(\alpha)$ that are disjoint. As we remarked earlier, for $k = 1$ one can work with distances in the graph and the ball of radius $\alpha/2$ is well defined.



For $k > 1$, there may be multiple terminals at close distance d from a terminal t, but nevertheless $Cost(t)$ could be much larger than d. The reason for this is that t needs to reach k terminals via *vertex disjoint* paths and there may be a vertex v whose removal disconnects t from *all* the nearby terminals. Consider the example in the figure, where filled circles denote other terminals: The terminal t is willing to pay for e and edges on P but not e'. There does not appear to be a natural notion of a ball; however, we show that one can define some auxiliary costs on the edges (that vary based on t) which can then be used to define a ball for t. The complexity of the analysis comes from the fact that the balls for different t are defined by different auxiliary edge costs. Now we show how the auxiliary costs can be defined.

We can obtain the augmentation cost of a terminal t via a min-cost flow computation in an associated *directed* graph $G_t(V_t, E_t)$ constructed from G in the following standard way: make 2 copies v^+ and v^- of each vertex $v \neq t$, with a single edge/arc between them, and for each undirected edge uv in G, edges from u^+ to v^- and v^+ to u^-. Further, we add a new vertex r_t as sink, and for each terminal $\hat{t}$ other than t, add a 0-cost edge from $\hat{t}^+$ to r_t. Recall that an augmentation for t is a set of k disjoint paths from t that end at distinct terminals in $T \setminus \{t\}$, or the root. While constructing G_t, then, the root is also considered a terminal, and we make k copies of it to account for the fact that multiple paths in the augmentation can end at the root; each such copy is also connected to the sink r_t. We now ask for a minimum cost set of k disjoint paths from t to r_t[‡]; these correspond to a minimum-cost augmentation for t. It is useful to use a linear programming formulation for the min-cost flow computation. The linear program for computing the augmentation cost of t, and its dual are shown below. We refer to these as **Primal-Aug**(t) and **Dual-Aug**(t) respectively.

$$\min \sum_{e \in E_t} c_e f_e$$

$$\begin{aligned}
\textstyle\sum_{e \in \delta^-(r_t)} f_e &\geq k \\
\textstyle\sum_{e \in \delta^-(v)} f_e &= \textstyle\sum_{e = \delta^+(v)} f_e \quad (\forall v \neq t, r_t) \\
f_e &\leq 1 \quad (\forall e \in E_t) \\
f_e &\geq 0 \quad (\forall e \in E_t)
\end{aligned}$$

$$\max k \cdot \Pi - \sum_e z_e^t$$

$$\begin{aligned}
\Pi - \pi_t(u) &\leq c_e + z_e^t \quad (\forall e = (u, r_t)) \\
\pi_t(v) - \pi_t(u) &\leq c_e + z_e^t \quad (\forall e = (u, v), \\
&\qquad\qquad u \neq t, v \neq r_t) \\
\pi_t(v) &\leq c_e + z_e^t \quad (\forall e = (t, v) \in E_t) \\
z_e^t &\geq 0 \quad (e \in E_t)
\end{aligned}$$

Note that the cost of an optimal solution to **Primal-Aug**(t) is equal to $Cost(t)$. The interesting aspect is the interpretation of the dual variables. The variables z_e^t are auxiliary costs on the edges. One can then interpret the dual **Dual-Aug**(t) as setting z_e^t values such that the distance from t, with modified cost of each edge e set to $c_e + z_e^t$, is equal to Π for every other terminal t'. Thus the modified costs create a ball around t in which all terminals are at equal distance!

Thus, the overall game plan of the proof is the following. For each t solve **Primal-Aug**(t) and find an appropriate solution to **Dual-Aug**(t) (this requires some care). Use

[‡]Note that we do not make two copies of t, as we will never use an incoming edge to t in a min-cost set of paths. All edges are directed out of the unique copy of t.

these dual variables to define a non-uniform ball around t in the original graph G. This leads to a feasible setting of variables in **Dual-Conn** (with the balls being approximately disjoint). Although the scheme at a high level is fairly natural, the technical details are non-trivial and somewhat long. In particular, one requires an important combinatorial lemma on intersecting path systems that was formulated in [4] — here we give an improved proof of a slight variant that we need. The use of this lemma leads to the exponential dependence on k. A certain natural conjecture regarding the non-uniform balls, if true, would lead to a polynomial dependence on k. We refer the reader to the full version for the details.

3 Rent-or-Buy

In this section we describe and analyze a simple algorithm for the SS-k-RENT-OR-BUY problem. Recall that the input to this problem is the same as that for SS-k-CONNECTIVITY with an additional parameter M. The goal is to find for each terminal $t \in T$, k vertex-disjoint paths $\vec{P} \in \mathcal{P}_t^k$ to the root r. The objective is to minimize the total cost of the chosen paths where the cost of an edge e is $c_e \cdot \min\{M, |T_e|\}$ where T_e is the set of terminals whose paths contain e. In other words an edge can either be bought at a price of Mc_e in which case any number of terminals can use it or an edge can be rented at a cost of c_e per terminal. Our algorithm given below is essentially the same as the random marking algorithm that has been shown to give an $O(1)$ approximation for the case of $k = 1$ [12].

RENT-OR-BUY-SAMPLE:
1. Sample each terminal independently with probability $1/M$.
2.1 Find a subgraph H in which every sampled terminal is k-connected to the root.
2.2 *Buy* the edges of H, paying Mc_e for each edge $e \in H$.
3. For each non-sampled terminal, greedily *rent* disjoint paths to k distinct sampled terminals.

It is easy to see that the algorithm is correct. Note that a non-sampled terminal can always find feasible paths since the root can be the endpoint of all k paths. The algorithm and analysis easily generalize to the case where each terminal t has a demand d_t to be routed to the root. The algorithm can be analyzed using the *strict cost-shares* framework of Gupta *et al.* [12] for sampling algorithms for rent-or-buy and related problems. It is not hard to show that the REVERSE-GREEDY algorithm directly implies the desired strict-cost shares needed for the framework. This allows us to conclude that the approximation ratio of RENT-OR-BUY-SAMPLE is no more than two times that of REVERSE-GREEDY.

THEOREM 4. *There is a $O(f(k)k^2 \log h)$-approximation for the* SS-k-RENT-OR-BUY *problem.*

We omit the formal proof of the above theorem in this version. In fact we give a direct and somewhat complex analysis that proves a slightly weaker bound than the above for reasons that we discuss now. One of our motivations to understand SS-k-RENT-OR-BUY is for its use in obtaining algorithms for the SS-k-BUY-AT-BULK problem. For $k = 1$, previous algorithms for SS-k-BUY-AT-BULK [11, 12] could use an algorithm for SS-k-RENT-OR-BUY essentially as a black box. However, for $k \geq 2$ there are important technical differences and challenges that we outline in Section 4. We cannot, therefore, use an algorithm for SS-k-RENT-OR-BUY as a black box. In a nutshell, the extra property that we need is the following. In the sampling algorithm RENT-OR-BUY-SAMPLE, there is no bound on the

number of unsampled terminals that may route to any specific sampled terminal. In the buy-at-bulk application we need an extra *balance* condition which ensures that unsampled terminals route to sampled terminals in such a way that no sampled terminal receives more than βM demand where $\beta \geq 1$ is not too large. We prove the following technical lemma that shows that β can be chosen to be $O(f(k)k\log^2 h)$.

LEMMA 5. *Consider an instance of* RENT-OR-BUY *and let* OPT *be the value of an optimal fractional solution to the given instance. Then for each terminal t we can find paths $P_1^t, P_2^t, \ldots, P_o^t$ with the following properties: (i) $o \geq (k - 1/2)M$ and (ii) the paths originate at t and end at distinct terminals or the root and (iii) no edge e is contained in more than M paths for any terminal t. Moreover the total rental cost of the paths is $O(f(k)e^{O(k^2)} \cdot k^5 \log h) \cdot M \cdot$ OPT and no terminal is the end point of more than $O(f(k)k\log^2 h \cdot M)$ paths.*

The proof of the above lemma is non-trivial. We are able to prove it by first analyzing the sampling based algorithm directly via the natural LP relaxation for SS-k-RENT-OR-BUY. Although the underlying ideas are inspired by the ones for SS-k-CONNECTIVITY, the proof itself is fairly technical.

4 Buy-at-Bulk Network Design

In this section we consider the SS-k-BUY-AT-BULK problem. We first consider the uniform version; Section 4.1 discusses the non-uniform version.

Each terminal $t \in T$ wishes to route one unit of demand to the root along k vertex disjoint paths. More generally, terminals may have different demands, but we focus on the unit-demand case for ease of exposition. There are b cable-types; the ith cable has capacity u_i and cost w_i per unit length. Let $f : R^+ \to R^+$ be a sub-additive function[§] where $f(x)$ is the minimum-cost set of cables whose total capacity is at least x. The goal is to find a routing for the terminals so that $\sum_e c_e \cdot f(x_e)$ is minimized where x_e is the total flow on edge e. One can assume that the cables exhibit economy of scale; that is, $w_i/u_i > w_{i+1}/u_{i+1}$ for each i. Therefore, there is some parameter g_{i+1}, with $u_i < g_{i+1} < u_{i+1}$, such that if the flow on an edge is at least g_{i+1}, it is more cost-effective to use a single cable of type $i+1$ than g_{i+1}/u_i cables of type i. Consistent with this notation, we set $g_1 = 1$; since all our cables have capacity at least u_1, if an edge has non-zero flow, it must use a cable of type at least 1.

Our overall algorithm follows the same high-level approach as that of the previous single-sink algorithms for the $k = 1$ problem [11, 12]. The basic idea is as follows: Given an instance in which the demand at each terminal is of value at least g_i, it is clear that cable types 1 to $i-1$ can be effectively ignored. The goal is now to aggregate or cluster the demand from the terminals to some cluster centers such that the aggregated demand at the cluster centers is at least g_{i+1}. Suppose we can argue the following two properties of the aggregation process: (i) the cost of sending the demand from the current terminals to the cluster centers is comparable to that of OPT and (ii) there exists a solution on the cluster centers of cost not much more than OPT. Then we have effectively reduced the problem to one with fewer cables, since the demand at the cluster centers is at least g_{i+1}. We can thus recurse on this problem. For $k = 1$ this outline can be effectively used to obtain an

[§]Any sub-additive f can conversely be approximated by a collection of cable-types.

$O(1)$ approximation *independent* of the number of cable types. There are several obstacles to using this approach for $k > 1$. The most significant of these is that it is difficult to argue that there is a solution on the new cluster centers of cost not much more than OPT. In the case of $k = 1$, this is fairly easy, as the new cluster centers can pretend to randomly send the demand back to the original terminals; for higher k, since centers need to send demand along k disjoint paths, this is no longer straightforward.

To deal with these issues, we perform a 2-stage aggregation process that is more complex than previous methods: First, given centers with demand g_i, we cluster demand to produce a new set of centers with demand u_i, using a result of [2]. Second, given centers with demand u_i, we use some ideas from Section 3 for RENT-OR-BUY to produce a new set of centers with demand g_{i+1}. The algorithm of [2] that we use in the first stage applies only for $k = 2$; our ideas can be extended to arbitrary k. We describe the two-stage aggregation process to go from a set of centers with demand g_i to a new set of centers with demand g_{i+1} below; we can then recurse.

Given an instance of SS-k-BUY-AT-BULK with center set T in which all demands are at least g_i, we can effectively assume that an optimal solution only uses cables of type i to b; let OPT_i denote the cost of an optimal solution to this instance. Let H denote an optimal solution to the SS-k-CONNECTIVITY instance with terminal set T, where the cost of edge e is $w_i c_e$; the cost of H is a lower bound on OPT_i. (Consider an optimal solution to the SS-k-BUY-AT-BULK instance; the set of edges with installed cables k-connects T to the root, and the cost on each edge is at least $w_i c_e$.) It follows from a clustering algorithm of [2] that for $k = 2$, we can find a new set of centers T' in polynomial time such that: (i) every $t \in T$ can route flow to 2 centers in T' via disjoint paths in H; (ii) the total flow on any edge in H is $O(1)u_i$; (iii) the demand at each $t' \in T'$ is at least u_i and at most $7u_i$; and (iv) There is a solution to the new buy-at-bulk instance on T' of expected cost at most $O(1)\mathrm{OPT}_i$.[¶] This completes the first aggregation stage.

We now have an instance of SS-k-BUY-AT-BULK with center set T in which each center has demand $\approx u_i$, and with an optimal solution of cost at most $\mathrm{OPT}'_i = O(1)\mathrm{OPT}_i$. Consider a modified instance in which all demands are set equal to u_i, the cable capacity u_{i+1} is set to infinity and the cable-types $i+2$ to ℓ are eliminated. Clearly, the cost of an optimal solution to this modified instance is no more than OPT'_i; simply replace each cable of higher capacity with a single cable of type $i+1$. However, we now have an instance of RENT-OR-BUY with $M = g_{i+1}/u_i$. We can now perform our second stage of aggregation; the key idea here is to use Lemma 5 from Section 3 which guarantees a desired balance condition. This is sufficient for the above described scheme to go through and yield the following result. Unlike the $k = 1$ case, each aggregation step loses a logarithmic factor in the approximation and hence the approximation we can guarantee is exponential in the number of cables.

THEOREM 6. *There is an* $(O(\log h))^{3b}$*-approximation for* SS-2-BUY-AT-BULK *with* b *cable-types.*

[¶]The algorithm as described in [2] enforces a weaker version of condition (iii); the demand at each $t' \in T'$ is at least u_i, and at many centers, the demand is at most $7u_i$. The centers of so-called star-like jumbo clusters may have higher demand, but the algorithm can be extended so that such high demand centers have their demand split into smaller units.

4.1 Non-uniform Buy-at-Bulk

We now consider the non-uniform version of SS-k-BUY-AT-BULK. In this version, for each edge e of the graph G there is a given sub-additive cost function f_e and routing x units of demand on e results in a cost of $f_e(x)$. The uniform version is a special case where $f_e = c_e \cdot f$ for a single sub-additive function f. The non-uniform buy-at-bulk problem is considerably harder than its uniform variant and we refer the reader to [16, 5, 7] for prior work and related pointers. We have already mentioned that prior to this work, for $k \geq 2$ the SS-k-BUY-AT-BULK problem did not admit a non-trivial approximation even for the (uniform) 2-cable problem. For the non-uniform single-sink problem there are essentially two approximation algorithms known for $k = 1$, one from [16] and the other from [5]. The algorithm of Charikar and Kargiazova [5] admits a natural generalization for $k \geq 2$ that we analyze using our result for SS-k-CONNECTIVITY. We obtain a ratio of $2^{O(\sqrt{\log h})}$ which is essentially the same as the one shown in [5] for the multi-commodity problem (due to a similar recurrence in the analysis). We remark that the [5] proves a bound of $O(\log^2 h)$ for the single-sink problem. However, for $k \geq 2$ the analysis of the recurrence changes dramatically from that for $k = 1$. Although the bound we show is not impressive, the randomized inflated greedy algorithm of [5] is extremely simple and elegant. It is easy to implement and amenable to heuristic improvement and has shown to be effective in some empirical evaluation [3]. We now describe the algorithm of [5] adapted to SS-k-BUY-AT-BULK. We assume that each terminal has unit demand to begin with.

RANDOM-INFLATED-GREEDY:
1. Pick a random permutation π of the terminals in T.
2. For $i = 1$ to h in that order, greedily route h/i units of demand from t_i to the root r along k disjoint paths using the cheapest cost paths in the network built by the previous $i - 1$ terminals.

Note that the algorithm routes h/i units of demand for t_i although only one unit of demand is required to be routed. We refer the reader to [5] for the background and intuition behind the design of the above algorithm. Each terminal is routed greedily but the cost of routing on an edge depends on the routing of the previous terminals. More precisely, if x_e^{i-1} is the amount of demand routed on an edge e by the first $i - 1$ terminals then the cost of routing an additional h/i units for terminal i on e is given by $c_e^i = f_e(x_e^{i-1} + h/i) - f_e(x_e^{i-1})$. One can use a min-cost flow computation with costs c_e^i to find the cheapest k disjoint paths from t_i to r. It is easy to see that the algorithm is correct; in the case of $k = 1$, it is known to have an approximation ratio of $O(\log^2 h)$ for $k = 1$ [5]. However, for $k \geq 2$ we are able to establish the following theorem.

THEOREM 7. *For any fixed k,* RANDOM-INFLATED-GREEDY *is a $2^{O(\sqrt{\log h})}$-approximation for the non-uniform version of* SS-k-BUY-AT-BULK *with unit-demands. For arbitrary demands there is a $\log D \cdot 2^{O(\sqrt{\log h})}$ approximation algorithm where D is the ratio of the maximum to minimum demands.*

Acknowledgments: We are grateful to Sanjeev Khanna for several discussions on the k-connectivity problem and for explaining the results and ideas in [4]. We also thank Anupam Gupta for several useful discussions.

References

[1] M. Andrews and L. Zhang. The access network design problem. *Proc. of IEEE FOCS*, 40–49, 1998.

[2] S. Antonakapoulos, C. Chekuri, B. Shepherd and L. Zhang. Buy-at-Bulk Network Design with Protection. *Proc. of IEEE FOCS*, 634–644, 2007.

[3] S. Antonakapoulos and L. Zhang. Heuristics for Fiber Installation in Optical Network Optimization. *Proc. of IEEE Globecom*, 2342–2347, 2007.

[4] T. Chakraborty, J. Chuzhoy and S. Khanna. Network Design for Vertex Connectivity. *Proc. of ACM STOC*, 2008.

[5] M. Charikar and A. Karagiozova. On non-uniform multicommodity buy-at-bulk network design. *Proc. of ACM STOC*, 176–182, 2005.

[6] C. Chekuri, P. Claisse, R. Essiambre, S. Fortune, D. Kilper, W. Lee, K. Nithi, I. Saniee, B. Shepherd, C. White, G. Wilfong, and L. Zhang. Design tools for transparent optical networks. *Bell Labs Technical Journal*, 11(2):129–143, 2006.

[7] C. Chekuri, M. Hajiaghayi, G. Kortsarz, and M. Salavatipour. Approximation algorithms for non-uniform buy-at-bulk network design problems. *Proc. of IEEE FOCS*, 677–686, 2006.

[8] J. Chuzhoy and S. Khanna. Algorithms for Single-Source Vertex-Connectivity. *Proc. of IEEE FOCS*, October 2008.

[9] L. Fleischer, K. Jain, and D.P. Williamson. An iterative rounding 2-approximation algorithm for the element connectivity problem. *JCSS*, 72(5):838–867, 2006.

[10] A. Frank and T. Jordan. Minimal edge-covers pairs of sets. *J. of Combinatorial Theory, B*, 65(1):73–110, 1995.

[11] S. Guha, A. Meyerson, and K. Munagala. A constant factor approximation for the single sink edge installation problem. *Proc. of ACM STOC*, 383–388, 2001.

[12] A. Gupta, A. Kumar, M. Pál, and T. Roughgarden. Approximation via cost-sharing. *JACM*, 54(3), 2007.

[13] K. Jain. A factor 2 approximation algorithm for the generalized steiner network problem. *Combinatorica*, 21(1):39–60, 2001.

[14] G. Kortsarz, R. Krauthgamer and J. Lee. Hardness of approximation for vertex-connectivity network design problems. *SIAM J. on Computing*, 33(3):704–720, 2004.

[15] G. Kortsarz and Z. Nutov. Approximating min-cost connectivity problems. Chapter in *Handbook on Approximation Algorithms and Metaheuristics*, CRC Press, 2006.

[16] A. Meyerson, K. Munagala, and S. Plotkin. Cost-distance: Two metric network design. *Proc. of IEEE FOCS*, 383–388, 2000.

[17] F.S. Salman, J. Cheriyan, R. Ravi, and S. Subramanian. Buy at bulk network design: Approximating the single-sink edge installation problem. *SIAM J. on Optimization*, 11(3):595–610, 2000.

[18] D.P. Williamson, M.X. Goemans, M. Mihail and V.V. Vazirani. A primal-dual approximation algorithm for generalized Steiner network problems. *Combinatorica*, 15:534–454, December 1995.

Foundations of Software Technology and Theoretical Computer Science (Bangalore) 2008.
Editors: R. Hariharan, M. Mukund, V. Vinay; pp 143-154

Graph Games on Ordinals*

Julien Cristau[1], **Florian Horn**[1,2]
{jcristau,horn}@liafa.jussieu.fr

[1] LIAFA, Université Paris 7, Case 7014, F-75205 Paris 13, France

[2] Lehrstuhl für Informatik VII, RWTH, Ahornstraße 55, 52056 Aachen, Germany

ABSTRACT.
We consider an extension of Church's synthesis problem to ordinals by adding limit transitions to graph games. We consider game arenas where these limit transitions are defined using the sets of cofinal states. In a previous paper, we have shown that such games of ordinal length are determined and that the winner problem is PSPACE-complete, for a subclass of arenas where the length of plays is always smaller than ω^ω. However, the proof uses a rather involved reduction to classical Muller games, and the resulting strategies need infinite memory.
We adapt the LAR reduction to prove the determinacy in the general case, and to generate strategies with finite memory, using a reduction to games where the limit transitions are defined by priorities. We provide an algorithm for computing the winning regions of both players in these games, with a complexity similar to parity games. Its analysis yields three results: determinacy without hypothesis on the length of the plays, existence of memoryless strategies, and membership of the winner problem in NP ∩ co-NP.

1 Introduction

Church's problem, introduced in [Chu63], is fundamental in the theory of automata over infinite strings. It considers a specification $\phi(X, Y)$ — usually a MSO formula — over pairs of infinite sequences. A solution to this problem is a circuit which computes an output sequence Y using a letter-by-letter transformation of the input sequence X. The Büchi-Landweber theorem shows the decidability of this problem, and provides an automatic procedure to compute a solution [BL69]. The proof builds on McNaughton's game-theoretic presentation of this problem [McN65]. McNaughton games are perfect information two-player games where at every stage $n < \omega$, player X chooses first whether he accepts n, and Y replies in kind. Player Y wins a play if the sets of integers accepted by X and Y verify ϕ. A winning strategy for Y gives a solution to Church's problem. Additionally, a winning strategy can be computed by a finite ω-automaton with output or, equivalently, defined using an MSO formula.

Church's problem can be extended to sequences of arbitrary ordinal length. One possible extension is to fix the length of the plays in advance: in [RS08], Rabinovich and Shomrat use a compositional method to show that for any countable ordinal α and MSO formula $\phi(X, Y)$, the corresponding McNaughton game is determined and the winner problem is decidable. Moreover, if $\alpha < \omega^\omega$ it's possible to compute a formula defining a winning strategy.

*This paper was supported in part by the French ANR DOTS.

FSTTCS 2008
IARCS Annual Conference on
Foundations of Software Technology and Theoretical Computer Science
http://drops.dagstuhl.de/opus/volltexte/2008/1768

Our approach is based on the graph games used in verification [Tho95], and Büchi's automata on words of ordinal length [Bü73]. In addition to the usual successor transitions, limit transitions allow the game to continue past any limit ordinal. In this model, the length of the plays is not fixed a priori, but depends on the actions of the players: the game stops when one of the players wins. In the paper, we only consider reachability winning conditions; however, any regular condition can be represented this way, because the addition of limit transitions allows to embed more complex transitions in the game arena itself. In [CH08], we studied a restriction of these games, disallowing limit transitions of the form $P \to q \in P$, which effectively reduces the scope of this work to plays shorter than ω^ω. Another drawback is that the strategies we obtained needed infinite memory. In this paper, we lift this restriction and prove the determinacy and existence of strategies with finite memory. We first solve a particular case of games where the limit transitions are defined using priorities, which are closely related to parity games of length ω. We present an algorithm computing the winning regions for both players in these games. Determinacy follows from its correctness, and further analysis gives positional winning strategies for both players in their winning regions. We derive from this the membership of the winner problem in $\mathsf{NP} \cap \text{co-}\mathsf{NP}$. We also derive an alternative proof of the results of Rabinovich and Shomrat on McNaughton games of length smaller than ω^ω: determinacy, decidability, definable strategy and strategy synthesis. Using an adaptation of the *Latest Appearance Records* of Gurevich and Harrington [GH82], we give a reduction from ordinal games to priority ordinal games. From this extended LAR reduction and our former results, we derive the determinacy of games of ordinal length, as well as the existence of finite-memory strategies for both players.

Overview of the paper. In Section 2, we recall the definitions of graph games of ordinal length, as well as their variant with priority-controlled transitions. Section 3 presents an algorithm for solving priority ordinal games, and its theoretical consequences. Section 4 shows how to adapt the LAR reduction to games of ordinal length, in order to get general determinacy and finite-memory strategies for all ordinal games. Finally, Section 5 summarises our results, and presents perspectives for future work.

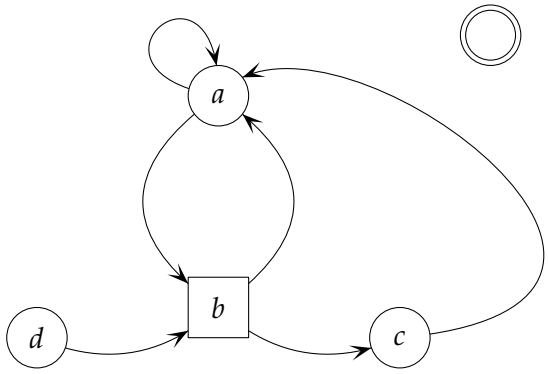
2 Definitions

Ordinals and words of ordinal length

An ordinal α is a set equipped with a well-founded linear order. Ordinals can be ordered in a natural way, and we take the usual convention identifying each ordinal with the set of the smaller ordinals. A *word* ρ of length α over an alphabet Σ is a mapping $(\rho_\beta)_{\beta<\alpha}$ from α to Σ. We denote by $|\rho|$ the length of the word. The *prefix* of ρ of length $\beta \leq |\rho|$, noted $\rho_{<\beta}$, is defined as $(\rho_\gamma)_{\gamma<\beta}$. Similarly, we write $\rho_{\geq\beta}$ for the *suffix* $(\rho_\gamma)_{\beta\leq\gamma<|\rho|}$. The subset of Σ appearing in ρ is $\mathrm{Occ}(\rho) = \{\rho_\beta \mid \beta < |\rho|\}$. Finally, if $|\rho|$ is a limit ordinal, the *limit* of ρ, noted $\lim\rho$, is the set $\{s \in \Sigma \mid \forall\beta < |\rho|,\ \exists\gamma > \beta,\ \rho_\gamma = s\}$.

Games of ordinal length

A reachability game of ordinal length (*ordinal game*) G is played by two players called Eve and Adam on an *arena* of the form $(Q, Q_E, Q_A, T, \lambda)$. The tuple (Q, T) is a directed graph and the set of vertices is partitioned between Adam's vertices (Q_A, graphically represented by $\Box$), Eve's vertices (Q_E, graphically represented by $\bigcirc$), and a *target* vertex $\circledcirc$. The function λ represents *limit transitions*; it maps $\mathcal{P}(Q_A \cup Q_E)$ to Q. We assume that every vertex except $\circledcirc$ has at least one successor. We give an example of reachability game of ordinal length in Figure 1. Some limit transitions aren't possible in this particular example, and are thus omitted. In this paper, we consider the case where Q is finite.



$$\lambda(\{a\}) = d \qquad \lambda(\{a,b\}) = c \qquad \lambda(\{a,b,c\}) = c$$
$$\lambda(\{a,b,d\}) = a \qquad \lambda(\{a,b,c,d\}) = \circledcirc$$

Figure 1: Game of ordinal length

A *play* ρ on a game G is an ordinal word on Q such that for any $\alpha < |\rho|$:

- if $\alpha = \beta + 1$, then $(\rho_\beta, \rho_\alpha) \in T$;
- if α is a limit ordinal, then $\lambda(\lim \rho_{<\alpha}) = \rho_\alpha$.

The set of all plays is noted Ω. It can be divided into four disjoint subsets:

1. The set of plays which have a last state in Q_E. These plays can be extended through a successor transition, chosen by Eve — a *move* of Eve.
2. The set of plays which have a last state in Q_A. These plays can be extended through a successor transition, chosen by Adam — a *move* of Adam.
3. The set of plays which have $\circledcirc$ as last state. These plays are said to be winning for Eve. Any other play is winning for Adam.
4. The set of plays without a last state. These plays can be extended through a unique limit transition.

Notice that our definition for winning plays deviates from the classical interpretation of infinite games, where there is no winner for the partial plays. This is necessary here, as non-winning plays can go on without ever ending, even in the transfinite sense. It's not possible to easily distinguish between plays where Eve has not yet won, and plays that are definitely won by Adam.

A *strategy* for Eve is a function $\sigma : \Omega \to Q$ such that if ρ ends in $q \in Q_E$, then $(q, \sigma(\rho)) \in$

T. A strategy with finite memory M for Eve is a finite transducer working over the states of M. It is defined by three functions:

- $\sigma^{\mathtt{m}} : M \times Q \to M$ is the memory update for successor transitions.
- $\sigma^{\mathtt{l}} : \mathcal{P}(M) \to M$ is the memory update for limit transitions.
- $\sigma^{\mathtt{n}} : M \times Q_E \to Q$ outputs Eve's next move.

One can define in the same way strategies and strategies with finite memory for Adam. If M has only one element, σ is a *positional* strategy. A play ρ of length α is *consistent* with a strategy σ for Eve if $\rho_{\beta+1} = \sigma(\rho_{<\beta})$ for every β such that $\beta + 1 < \alpha$ and $\rho_\beta \in Q_E$. A strategy σ is *winning* for Eve if there is an ordinal α such that any play consistent with σ has length less than α. Notice that this condition imposes that $\circledcirc$ is eventually reached, as plays can otherwise always be extended. Conversely, a strategy τ is winning for Adam if no play consistent with τ ends in $\circledcirc$. A game is *determined* if there is always a winning strategy for one of the players.

If all limit transitions lead either to $\circledcirc$ or to a sink state, then all plays are shorter than ω, and the game is a traditional Muller game.

In [CH08], we showed that a subclass of ordinal games are determined:

THEOREM 1.*[CH08] Reachability games of ordinal length without transitions of the form $\lambda(P) = q \in P$ are determined.*

Notice that this subclass is not a mere syntactic condition: it restricts the scope of Theorem 1 to games where the plays have length less than ω^ω, as can be deduced from Theorem 2:

THEOREM 2.*[Cho78] In an automaton with n states where no limit transition is of the form $\lambda(P) = q \in P$, all runs are shorter than ω^n.*

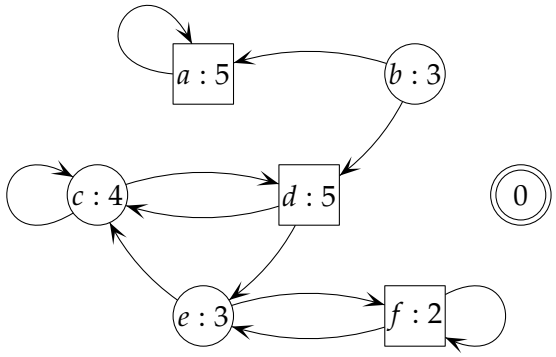
Priority ordinal games. A reachability game of ordinal length with priority transitions (*priority ordinal game*) is a game where the limit transition function λ is defined in a specific way: a *colouring function* χ maps each state to a colour in $\{0, \dots, d-1\}$; another function δ maps each colour to a state of Q. Then, for any set $P \subseteq Q$, the limit transition is given by $\lambda(P) = \delta(\min\{\chi(q) \mid q \in P\})$.

Figure 2 gives an example of a priority game, with $d = 6$. In this game, Adam wins from states c and d: from d he can go to c, and any limit transition will take the token back to d. Eve wins from everywhere else: from b she goes to a and the token will reach the target after playing $(ba^\omega)^\omega$; from e she goes to f and the token will eventually reach a.

Not all ordinal games can be represented by priority transitions. In the game of Figure 1, for example, the set $\{a, b, c, d\}$ leads neither to the destination of $\{a, b, c\}$, nor to the one of $\{a, b, d\}$. This cannot occur in a priority game, as $\min \chi(\{a, b, c, d\})$ would be either $\min \chi(\{a, b, c\})$, or $\min \chi(\{a, b, d\})$.

Subgames, attractors and traps. We recall here some classical concepts for infinite games (see e.g. [Tho95]), which we use in the context of ordinal games.

Let $G = (Q, Q_E, Q_A, T, \lambda)$ be an ordinal game, and Q' a subset of Q. The tuple $G' = (Q', Q'_E, Q'_A, T', \lambda')$ —where $Q'_E = Q' \cap Q_E$, $Q'_A = Q' \cap Q_A$, and T' and λ' are the restrictions



$\delta(2) = a \qquad \delta(3) = \circledcirc \qquad \delta(4) = d \qquad \delta(5) = b$

Figure 2: Priority Game

to Q' of the transitions in G— is an *ω-subgame* of G if every state in G' has a successor. We write $G' = G \setminus P$ if $Q' = Q \setminus P$.

Let P be a subset of states in a game G. The *ω-attractor* to P for Eve, noted $\mathrm{Attr}_E^G(P)$, is the set of states such that Eve can ensure that P is reached after a finite number of moves; it is defined as $\bigcup_{i\geq 0} \mathrm{Attr}_i$, where Attr_i is the least subset of Q such that:

- $\mathrm{Attr}_0 = P$;
- $\mathrm{Attr}_i \subseteq \mathrm{Attr}_{i+1}$;
- if $q \in Q_E$ and q has a successor in Attr_i, then $q \in \mathrm{Attr}_{i+1}$;
- if $q \in Q_A$ and all successors of q in G are in Attr_i, then $q \in \mathrm{Attr}_{i+1}$.

PROPOSITION 3. *In an ω-attractor for Eve, she has a positional strategy to ensure that P is reached in a finite number of moves.*

An *ω-trap* for Eve is a subset P of states such that Adam can ensure that the token stays in P for at least ω moves:

- if $q \in P \cap Q_E$, all its successors are in P;
- if $q \in P \cap Q_A$, q has a successor in P.

In all this paper, we use the terms *attractor* and *trap* to refer to ω-attractors and ω-traps, without any assumption about what happens beyond ω. Computing ordinal attractors is, indeed, the point of this work.

3 Solving priority ordinal games

Algorithm. Our main result is an algorithm computing the winning regions of both players in a priority ordinal game. It is inspired by Zielonka's algorithm for infinite (ω-length) parity games in [Zie98].

In order to make the algorithm simpler, we assume without loss of generality that the target $\circledcirc$ is the only state with priority 0, and that no state has priority 1. The priority 1 represents a virtual sink state, which is used to terminate the computation. The value of $\delta(1)$ is thus unused, and we assume that it is not defined.

Input: The game G
Output: The winning regions of Eve and Adam

$\mathtt{v}[0] \leftarrow E$
$\mathrm{To}[0] \leftarrow \mathrm{Attr}_E^G(\chi^{-1}(0))$
for $0 < i < d$ **do** $\mathrm{To}[i] \leftarrow \emptyset$
$\mathtt{H} \leftarrow G \setminus \mathrm{To}[0]$
$\mathtt{i} \leftarrow 0$
while $(\mathrm{To}[0] \cup \mathrm{To}[1]) \neq G$ **do**
  while $(\mathtt{H} \neq \emptyset)$ **do**
    $\mathtt{i}++$
    if $\exists j \mid \delta(\mathtt{i}) \in \mathrm{To}[j]$ **then**
      $\mathtt{v}[\mathtt{i}] \leftarrow \mathtt{v}[j]$
    else
      $\mathtt{v}[\mathtt{i}] \leftarrow A$
    $\mathrm{To}[\mathtt{i}] \leftarrow \mathrm{Attr}^{\mathtt{H}}_{\mathtt{v}[\mathtt{i}]}(\chi^{-1}(\mathtt{i}))$
    $\mathtt{H} \leftarrow \mathtt{H} \setminus \mathrm{To}[\mathtt{i}]$
  $Tmpto \leftarrow \mathrm{To}[\mathtt{i}]$
  $Tmpv \leftarrow \mathtt{v}[\mathtt{i}]$
  repeat
    $\mathtt{H} \leftarrow \mathtt{H} \cup \mathrm{To}[\mathtt{i}]$
    $\mathrm{To}[\mathtt{i}] \leftarrow \emptyset$
    $\mathtt{v}[\mathtt{i}] \leftarrow A$
    $\mathtt{i}--$
  until $(\mathtt{v}[\mathtt{i}] = Tmpv)$
  $\mathrm{To}[\mathtt{i}] \leftarrow \mathrm{To}[\mathtt{i}] \cup \mathrm{Attr}^{\mathtt{H}}_{Tmpv}(Tmpto)$
  $\mathtt{H} \leftarrow \mathtt{H} \setminus \mathrm{To}[\mathtt{i}]$
return $(\mathrm{To}[0], \mathrm{To}[1])$

Figure 3: Algorithm for Priority Games

The major difference with Zielonka's algorithm is that we have to determine which player wants to reach a colour j. The algorithm uses two arrays, $\mathtt{v}$ and To, indexed by colours, to store this information. At every step of the algorithm, $\mathtt{v}[j]$ is the player which is presumed to want to reach j. This can change each time the attractor to j is computed: $\mathtt{v}[j] = E$ if and only if there is a smaller colour i such that $\mathtt{v}[i] = E$ and $\delta(j) \in \mathrm{To}[i]$. Given a colour j, $\mathrm{To}[j] \subseteq Q$ is a set of states where player $\mathtt{v}[j]$ can guarantee an invariant, which will be precised later.

We compute embedded attractors, starting with the smallest (*i.e.* most important) colour: we compute an attractor to that colour, then remove these states from the graph and start again with the next colour. When all the graph is covered, the last computed attractor, $\mathrm{To}[k]$, is merged with a former one, $\mathrm{To}[j]$. We recompute then the attractors to colours greater than j. The algorithm ends when all the states are either in $\mathrm{To}[0]$ or in $\mathrm{To}[1]$. The former contains the winning region for Eve, and the latter is the winning region for Adam. The

termination is guaranteed by the fact that a state can only be removed from "To$[i]$" when another is added to "To$[j]$" with $j < i$, and that whenever line 14 is reached, every state belongs to exactly one of the To$[i]$. Figure 3 presents this algorithm in pseudo-code.

Correctness. In order to prove the correctness of the algorithm, we use the notation H^j for $Q \setminus \cup_{k<j} \text{To}[k]$, and the following easy but useful properties obtained from the construction of the array To.

PROPOSITION 4. *H^j is an ω-subgame of G.*

PROPOSITION 5. *For any colour j,* $\text{Attr}^{H^j}_{\mathrm{v}[j]}(\text{To}[j]) = \text{To}[j]$.

The correctness of the algorithm is proved separately for the two players: the arguments involved, while close, cannot be easily unified. In both cases, however, we define a predicate referring to the plays of the game and a property derived from it, and show that the property holds along the whole run. Its interpretation at the end of a run implies correctness.

In Eve's proof, we use the loop invariant $\mathcal{I}$ and the predicates $\mathcal{M}^j$ on the plays. The corresponding predicates for Adam are $\mathcal{J}$ and $\mathcal{N}^j$, respectively.

Informally, the predicates $\mathcal{M}^j$ correspond to strong **until** predicates of the form "$(> j)\mathcal{U}(j)$". Adam's predicates $\mathcal{N}^j$ correspond to weak **until** predicates of the form "$(> j)\mathcal{W}(j)$".

DEFINITION 6. *The predicate $\mathcal{M}^j(\rho)$ is defined as "*$\mathrm{v}[j] = E$*, and $\exists \alpha < \omega^{d-j}$ such that* $\text{Occ}(\rho_{<\alpha}) \subseteq \text{To}[j]$ *and either:*

- $|\rho| = \alpha$*, or*
- $\rho_\alpha \in \text{To}[j] \cap \chi^{-1}(j)$*, or*
- $\exists k < j$ *such that $\mathcal{M}^k(\rho_{\geq\alpha})$ holds".*

DEFINITION 7. *$\mathcal{I}$ is the property "Eve has a positional strategy σ such that, for any j such that* $\mathrm{v}[j] = E$*, and any play ρ starting in* To$[j]$ *and consistent with σ, $\mathcal{M}^j(\rho)$ holds".*

$\mathcal{I}$ holds at the beginning of a run: before line 6, To$[0]$ is the only non-empty set, and it contains only $\text{Attr}_E(\chi^{-1}(0))$. Propositions 8 and 9 guarantee that it remains true throughout the execution.

PROPOSITION 8. *Let us suppose that $\mathcal{I}$ holds at line 8. Then $\mathcal{I}$ holds at the next visit of line 14.*

SKETCH OF PROOF. If $\mathrm{v}[i] = A$, the property $\mathcal{I}$ is unchanged after the iteration. If $\mathrm{v}[i] = E$, an attractor strategy for Eve guarantees a visit to i in less than ω moves, unless Adam chooses to send the token outside of To$[i]$. If he does that, the structure of the array To as a series of alternating embedded ω-attractors guarantees that the token is sent to a To$[j]$ such that $j < i$ and $\mathrm{v}[j] = E$. ∎

PROPOSITION 9. *Let us suppose that $\mathcal{I}$ holds before a visit to line 17. Then $\mathcal{I}$ holds at the next visit of line 24.*

SKETCH OF PROOF. Once again, the interesting cases are those where $Tmpv = E$ and the token remains in $Tmpto$ long enough. As $Tmpto$ is a trap for Adam, Eve's strategy guarantees an infinite number of visits to i (the corresponding colour) in less than ω^{d-i+1} moves. The token is then sent to $\delta(i)$ which by definitions of v and i belongs to a $\mathrm{To}[j]$ such that $j < i$ and $\mathrm{v}[j] = E$. ∎

The structure of the proof for the states of Adam is quite similar, although the predicates are slightly weaker. We can prove that $\mathcal{J}$ holds along the whole run.

DEFINITION 10. *The predicate $\mathcal{N}^j(\rho)$ is defined as "$\mathrm{v}[j] = A$ and $\exists \alpha$ such that $\mathrm{Occ}(\rho_{<\alpha}) \subseteq \mathrm{To}[j]$ and either:*

- *$|\rho| = \alpha$, or*
- *$\rho_\alpha \in \mathrm{To}[j] \cap \chi^{-1}(j)$, or*
- *$\exists k < j$ such that $\mathcal{N}^k(\rho_{\geq \alpha})$ holds."*

DEFINITION 11. *$\mathcal{J}$ is the property "Adam has a positional strategy τ such that, for any j such that $\mathrm{v}[j] = A$, and any play ρ starting in $\mathrm{To}[j]$ and consistent with τ, $\mathcal{N}^j(\rho)$ holds".*

Consequences. The interpretation of $\mathcal{I}$ and $\mathcal{J}$ at the end of a run leads to the following Theorem:

THEOREM 12. *Let $G = (Q, Q_E, Q_A, T, \chi, \delta)$ be a priority ordinal game such that $\chi(Q \setminus \circledcirc) \subseteq [2, d-1]$. Then*

1. *G is determined;*
2. *Eve and Adam have positional winning strategies;*
3. *If Eve can win, then she can reach $\circledcirc$ in less than ω^d moves.*

COROLLARY 13. *The problem of the winner in reachability games of ordinal length with priority transitions belongs to* NP ∩ co-NP.

PROOF. Let's consider a game $G = (Q, Q_E, Q_A, T, \chi, \delta)$ and a state $q \in Q$. The problem is "Does Eve have a winning strategy if the token starts in q?"

*co-*NP*-membership.* If q is not winning for Eve, a winning strategy for Adam is a polynomial counter-example. Using it, we define the automaton $\mathcal{A}_\tau$ from G by removing the successor transitions of the form (r, s) with $r \in Q_A$ and $s \neq \tau(r)$. If τ is winning, then $\mathcal{L}(\mathcal{A}_\tau) = \emptyset$. On the other hand, if q is winning for Eve, $\mathcal{L}(\mathcal{A}_\tau) \neq \emptyset$ for any τ. As the emptiness problem for ordinal automata is decidable in polynomial time [Col07], this is a co-NP procedure.

NP*-membership.* We guess a positional strategy σ for Eve, and use it to define the automaton $\mathcal{A}_\sigma$ by removing from G successor transitions of the form (r, s) with $r \in Q_E$ and $s \neq \sigma(r)$, and making every state except $\circledcirc$ final. The strategy σ is winning if and only if the language accepted by the product of $\mathcal{A}_\sigma$ and an automaton accepting runs of length greater than ω^d is empty. ∎

This algorithm also yields an alternate proof of the following theorem by Rabinovich and Shomrat, in the case of ordinals less than ω^ω. They consider two-player games $\mathcal{G}_\phi^\alpha$

defined by an MSO formula ϕ and an ordinal α. In such a game, each player builds a subset of α in the following way: for every ordinal $\beta < \alpha$, player 0 chooses whether he wants to pick β, and player 1 responds. The set of positions picked by player 0 is noted X, those positions chosen by player 1 form the set Y. Player 0 wins if $\phi(X, Y)$ is true.

THEOREM 14. *[Theorem 29 of [RS08]] Let α be a countable ordinal and $\phi(X, Y)$ be a MSO-formula.*

Determinacy: One of the players has a winning strategy in the game $\mathcal{G}_\phi^\alpha$.
Decidability: It is decidable which of the players has a winning strategy.
Definable strategy: If $\alpha < \omega^\omega$, then the player who has a winning strategy also has a definable winning strategy. For every $\alpha \geq \omega^\omega$, there is a formula for which this fails.
Synthesis algorithm: If $\alpha < \omega^\omega$, we can compute a formula $\psi(X, Y)$ that defines a winning strategy for the winning player in $\mathcal{G}_\phi^\alpha$.

PROOF. Both the MSO-formula ϕ and the ordinal $\alpha < \omega^\omega$ can be represented as finite automata over ordinal words with priority transitions. The automaton corresponding to ϕ gets pairs (X_i, Y_i) of letters as input. The automaton for α accepts words of length exactly α. The product of these two automata can be seen as an ordinal game, where at every step $i < \alpha$ Adam chooses X_i and Eve then chooses Y_i. Eve needs to ensure that after α moves from each player, the ϕ automaton is in an accepting state, and thus she has a winning strategy if and only if Y has a winning strategy in $\mathcal{G}_\phi^\alpha$.

Determinacy and decidability follow from the correctness of Algorithm 3. Positional strategies in this finite arena can be represented as finite automata, and thus are definable. A synthesis algorithm can be derived from the proof of Algorithm 3. ∎

Notice that it is not possible to represent a single ordinal greater than ω^ω as a finite automaton on ordinal words. Thus, we can't interpret McNaughton games of length greater than ω^ω, as defined in [RS08], as finite graph games.

4 From priority transitions to ordinal games

In this section, we extend the results of Section 3 to the more general case of games of ordinal length, through an *ordinal game reduction*: there is a bisimulation between the graphs, such that equivalent states belong to the same player, and equivalent plays have limit transitions to equivalent states.

The LAR reduction. The Latest Appearance Records (LAR) were introduced by Gurevich and Harrington [GH82], in order to prove the *Forgetful Determinacy* of Muller ω-games (*i.e.* the existence of finite memory strategies). A LAR for a game G with n states is a pair (π, i), where π is a permutation over the states of G and i is an integer such that $1 \leq i \leq n$. We reduce any ordinal game $G = (Q, Q_E, Q_A, T, \lambda)$, to a priority ordinal game $\mathbb{G} = (\mathbb{Q}, \mathbb{Q}_E, \mathbb{Q}_A, \mathbb{T}, \chi, \delta)$. The states and successor transitions are defined in the same way as in the original reduction:

- $\mathbb{Q} = \{(\pi, i) \mid \pi \text{ is a permutation over } Q, 1 \leq i \leq n\}$
- $\mathbb{Q}_E = \{(\pi, i) \in \mathbb{Q} \mid \pi(1) \in Q_E\}$
- $\mathbb{Q}_A = \{(\pi, i) \in \mathbb{Q} \mid \pi(1) \in Q_A\}$

- $(\pi, i) \xrightarrow{\mathbb{T}} (\mu, j)$ if and only if:
  - $\pi(1) \xrightarrow{T} \mu(1)$
  - $\forall q, r \in Q \setminus \{\mu(1)\},\ \pi^{-1}(q) < \pi^{-1}(r) \Leftrightarrow \mu^{-1}(q) < \mu^{-1}(r)$
  - $\pi(j) = \mu(1)$

The limit transitions, by contrast, are much more involved than in the infinite setting. As we need to keep track of some of the memory *after* a limit transition, we use a different colour for each state of $\mathbb{G}$ — ordered so that $i < j \Rightarrow \chi(\pi, i) > \chi(\mu, j)$ (the exact ordering is unimportant, as long as this condition is verified). By abuse of notation we describe δ as a function from $\mathbb{Q}$ to $\mathbb{Q}$: $\delta(\pi, i)$ is the LAR (μ, j) such that:

- $\lambda(\cup_{j=1}^{i} \{\pi(j)\}) = \mu(1)$
- $\forall q, r \in Q \setminus \{\mu(1)\},\ \pi^{-1}(q) < \pi^{-1}(r) \Leftrightarrow \mu^{-1}(q) < \mu^{-1}(r)$
- $\pi(j) = \mu(1)$

The target states for Eve are the states (π, i) such that $\pi(1)$ is the target state in the original game G. Obviously these states can be merged to get a unique target state.

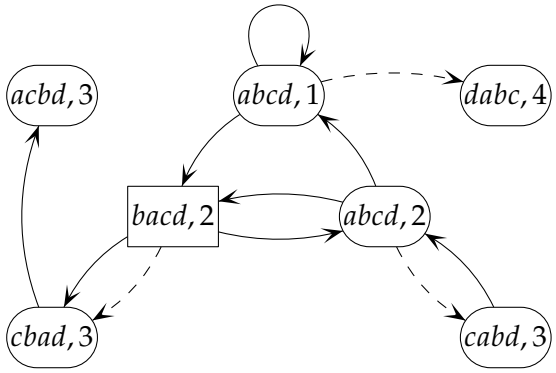


Figure 4: Detail of the LAR reduction of the game of Figure 1

Figure 4 gives a detail of the reduction of the game of Figure 1 (dashed arrows represent limit transitions). The game bisimulation is proved as in the case of infinite games, with some added fun due to the limit transitions.

LEMMA 15. *There is a bisimulation between G and $\mathbb{G}$ such that two bisimilar plays without a last state have a limit transition to two bisimilar states.*

Results for ordinal games. The LAR reduction allows us to extend Theorem 12 to any ordinal game:

THEOREM 16. *Let $G = (Q, Q_E, Q_A, T, \lambda)$ be an ordinal reachability game with n states. Then*

1. *G is determined;*
2. *Eve and Adam have winning strategies with memory $n!$;*
3. *if Eve can win, then she can reach $\circledcirc$ in less than $\omega^{n!}$ moves.*

SKETCH OF PROOF. Theorem 16 follows from Theorem 12 and the LAR reduction. The LAR reduction preserves the winner, which gives us the determinacy. The third part is ensured

because the length of plays is preserved. For the second part, we translate a positional strategy σ in $\mathbb{G}$ into a strategy $\varsigma = (\varsigma^{\mathtt{m}}, \varsigma^{\mathtt{l}}, \varsigma^{\mathtt{n}})$ in G:

- the memory states are LARs: $M = \mathbb{Q}$
- the memory updates mimic the transitions of $\mathbb{G}$:
  - $\varsigma^{\mathtt{m}}((\pi, i), q) = \{(\mu, j) \in \mathbb{T}(\pi, i) \mid \mu(1) = q\}$
  - $\varsigma^{\mathtt{l}}(P) = \delta(\min\{\chi(P)\})$
- the next-move function follows σ: $\varsigma^{\mathtt{n}}((\pi, i), q) = \sigma(\varsigma^{\mathtt{m}}((\pi, i), q))$

This amounts to considering an equivalent play on the reduced game $\mathbb{G}$, keeping the whole LAR in memory. ∎

5 Conclusion

We present a new model of two-player games on finite graphs. These games have plays of ordinal length, thanks to the addition of limit transitions. Our model is not comparable in general with McNaughton games. In McNaughton games, the length of plays is fixed a priori by the arena; this is not the case here, and it's not even possible to define an arena where plays have a fixed length α as soon as $\alpha \geq \omega^\omega$. In the case of games of length less than ω^ω, though, these two models are close, as shows our alternative proof of Rabinovich and Shomrat's theorem from [RS08].

In comparison to [CH08], this work lifts the syntactic restriction on arenas which limited the scope of our games to plays of length less than ω^ω. We introduce the central problem of arenas with priority transitions, which are a natural extension of parity games, and prove that they admit positional strategies. We derive from their study some interesting results, for example the existence of strategies with finite memory in the general case. This leads to our solution to Church's synthesis problem; and the existence of a bound on the number of steps needed to reach the target.

Our approach is closer to techniques commonly used in verification and model-checking than the composition method used in [RS08], and we think it could be useful in the context of verification of timed open systems. This would allow us to deal with Zeno behaviours, whereas most of the time, works on the subject consider models where they are forbidden [dAFH+03], or exclude Zeno runs from their results [AM99]. We would like to have a more constructive approach on the problem, following for example [JT07].

Bibliography

[AM99] Eugène Asarin and Oded Maler. As soon as possible: Time optimal control for timed automata. In *Proceedings of the Second International Workshop on Hybrid Systems: Computation and Control, HSCC'99*, volume 1569 of *Lecture Notes in Computer Science*, pages 19–30. Springer, 1999.

[BL69] J. Richard Büchi and Lawrence H. Landweber. Solving sequential conditions by finite-state strategies. *Trans. Amer. Math. Soc.*, 138:295–311, 1969.

[Bü73] J. R. Büchi. The monadic second-order theory of all countable ordinals. volume 328 of *Lecture Notes in Mathematics*, pages 1–127. Springer, 1973.

[CH08] Julien Cristau and Florian Horn. On reachability games of ordinal length. In *Prodeedings of the 34th International Conference on Current Trends in Theory and Practice of Computer Science, SOFSEM'08*, volume 4910 of *Lecture Notes in Computer Science*, pages 211–221. Springer, 2008.

[Cho78] Yaacov Choueka. Finite automata, definable sets, and regular expressions over ω^n-tapes. *Journal of Computer and Systems Sciences*, 17(1):81–97, 1978.

[Chu63] Alonzo Church. Logic, arithmetic, and automata. In *Proc. Int. Congr. Math. 1962*, pages 23–35, 1963.

[Col07] Thomas Colcombet. Factorisation forests for infinite words. In *Proceedings of the 16th International Symposium on Fundamentals of Computation Theory, FCT'07*, volume 4639 of *Lecture Notes in Computer Science*, pages 226–237. Springer, 2007.

[dAFH+03] Luca de Alfaro, Marco Faella, Thomas A. Henzinger, Rupak Majumdar, and Marielle Stoelinga. The element of surprise in timed games. In *Proceedings of the 14th International Conference on Concurrency Theory, CONCUR'03*, volume 2761 of *Lecture Notes in Computer Science*. Springer-Verlag, 2003.

[GH82] Yuri Gurevich and Leo Harrington. Trees, automata, and games. In *Proceedings of the 14th Annual ACM Symposium on Theory of Computing, STOC'82*, pages 60–65. ACM, 1982.

[JT07] Marcin Jurdziński and Ashutosh Trivedi. Reachability-time games on timed automata. In *Proceedings of the 34th International Colloquium on Automata, Languages and Programming, ICALP'07*, volume 4596 of *Lecture Notes in Computer Science*, pages 838–849. Springer, 2007.

[McN65] Robert McNaughton. Finite-state infinite games. Technical report, Massachusetts Institute of Technology, 1965. Project MAC.

[RS08] Alexander Rabinovich and Amit Shomrat. Selection and uniformization problems in the monadic theory of ordinals: A survey. In Arnon Avron, Nachum Dershowitz, and Alexander Rabinovich, editors, *Pillars of Computer Science*, volume 4800 of *Lecture Notes in Computer Science*, pages 571–588. Springer, 2008.

[Tho95] Wolfgang Thomas. On the synthesis of strategies in infinite games. In *Proceedings of the 12th Annual Symposium on Theoretical Aspects of Computer Science, STACS'95*, pages 1–13. Springer, 1995.

[Zie98] Wieslaw Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *Theor. Comput. Sci.*, 200(1-2):135–183, 1998.

Foundations of Software Technology and Theoretical Computer Science (Bangalore) 2008.
Editors: R. Hariharan, M. Mukund, V. Vinay; pp 155-162

3-connected Planar Graph Isomorphism is in Log-space

Samir Datta[1], **Nutan Limaye**[2], **Prajakta Nimbhorkar**[2]

[1] Chennai Mathematical Institute
Chennai, India

[2] The Institute of Mathematical Sciences,
Chennai, India

ABSTRACT. We consider the isomorphism and canonization problem for 3-connected planar graphs. The problem was known to be L -hard and in $\mathsf{UL} \cap \mathsf{coUL}$ [TW08]. In this paper, we give a deterministic log-space algorithm for 3-connected planar graph isomorphism and canonization. This gives an L -completeness result, thereby settling its complexity.
The algorithm uses the notion of universal exploration sequences from [Kou02] and [Rei05]. To our knowledge, this is a completely new approach to graph canonization.

1 Introduction

The general graph isomorphism problem is a well studied problem in computer science. Given two graphs, it deals with finding a bijection between the sets of vertices of these two graphs, such that the adjacencies are preserved. The problem is in NP , but it is not known to be complete for NP . In fact, it is known that if it is complete for NP , then the polynomial hierarchy collapses to its second level. On the other hand, no polynomial time algorithm is known. For general graph isomorphism NL and PL hardness is known [Tor00], whereas for trees, L and NC^1 hardness is known, depending on the encoding of the input [JT98].

In literature, many special cases of this general graph isomorphism problem have been studied. In some cases like trees [Lin92], [Bus97], or graphs with coloured vertices and bounded colour classes [Luk86], NC algorithms are known. We are interested in the case where the graphs under consideration are planar graphs. In [Wei66], Weinberg presented an $O(n^2)$ algorithm for testing isomorphism of 3-connected planar graphs. Hopcroft and Tarjan [HT74] extended this for general planar graphs, improving the time complexity to $O(n \log n)$. Hopcroft and Wong [HW74] further improved it to give a linear time algorithm. Its parallel complexity was first considered by Miller and Reif [MR91] and Ramachandran and Reif [RR90]. They gave an upper bound of AC^1. Verbitsky [Ver07] gave an alternative proof for the same bound. Recently Thierauf and Wagner [TW08] improved it to $\mathsf{UL} \cap \mathsf{coUL}$ for 3-connected planar graphs. They also proved that this problem is hard for L .

In this paper, we give a log-space algorithm for 3-connected planar graph isomorphism, thereby proving L -completeness. Thus the main result of our paper can be stated as follows:

FSTTCS 2008
IARCS Annual Conference on
Foundations of Software Technology and Theoretical Computer Science
http://drops.dagstuhl.de/opus/volltexte/2008/1749

THEOREM 1. *Given two 3-connected planar graphs G and H, deciding whether G is isomorphic to H is complete for* L .

Thierauf and Wagner use shortest paths between pairs of vertices of a graph to obtain a canonical spanning tree. A systematic traversal of this tree generates a canonical form for the graph. The best known upper bound for shortest paths in planar graphs is UL ∩ coUL [TW08]. Thus the total complexity of their algorithm goes to UL ∩ coUL , despite the fact that all other steps can be done in L .

We identify that their algorithm hinges on making a systematic traversal of the graph in canonical way. Thus we bypass the step of finding shortest paths and give an orthogonal approach for finding such a traversal. We use the notion of universal exploration sequences (UXS) defined in [Kou02]. Given a graph on n vertices with maximum degree d, a UXS is a polynomial length string over $\{0, \ldots, d-1\}$, that can be used to traverse the graph for a chosen combinatorial embedding ρ, starting vertex u and a starting edge $e = \{u, v\}$. Reingold [Rei05] proved that such a universal sequence can be constructed in L . Using this result, we canonize a 3-connected planar graph in log-space.

In Section 2, we give some basic definitions that we use in the later sections. In Section 3, we describe our log-space algorithm. We conclude with a discussion of open problems in Section 4.

2 Preliminaries

In this section, we recall some basic definitions related to graphs and universal exploration sequences.

2.1 The Graph Isomorphism Problem

DEFINITION 2. *Graph isomorphism: Two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are said to be isomorphic if there is a bijection $\phi : V_1 \to V_2$ such that $(u, v) \in E_1$ if and only if $(\phi(u), \phi(v)) \in E_2$.*

Let GI be the problem of finding such a bijection ϕ given two graphs G_1, G_2. Let Planar-GI be the special case of GI when the given graphs are planar. 3-connected planar graph isomorphism problem is a special case of Planar-GI when the graphs are 3-connected planar graphs. We recall the definition and properties of 3-connected planar graphs in the following section.

2.2 3-connected planar graphs

A graph $G = (V, E)$ is *connected* if there is a path between any two vertices in G. A vertex $v \in V$ is an *articulation point* if $G(V \setminus \{v\})$ is not connected. A pair of vertices $u, v \in V$ is a *separation pair* if $G(V \setminus \{u, v\})$ is not connected. A *biconnected* graph contains no articulation points. A 3-*connected* graph contains no separation pairs.

A *planar combinatorial embedding* ρ for a planar graph G specifies the cyclic (say, clockwise) ordering of edges around each vertex in some plane embedding of G. A graph G with a fixed combinatorial embedding ρ is called an *oriented graph* (G, ρ).

In general, a planar graph can have exponentially many planar embeddings. In [Whi33], Whitney proved that 3-connected planar graphs have precisely two combinatorial embeddings. This is a special property of 3-connected planar graphs which we crucially use in our log-space algorithm.

2.3 Universal Exploration Sequences

Let $G = (V, E)$ be a d-regular graph, with given combinatorial embedding ρ. The edges around any vertex u can be numbered $\{0, 1, \ldots, d-1\}$ according to ρ arbitrarily in clockwise order. A sequence $\tau_1\tau_2 \ldots \tau_k \in \{0, 1, \ldots, d-1\}^k$ and a starting edge $e_0 = (v_{-1}, v_0) \in E$, define a walk $v_{-1}, v_0, \ldots v_k$ as follows: For $0 \leq i \leq k$, if (v_{i-1}, v_i) is the s^{th} edge of v_i, let $e_i = (v_i, v_{i+1})$ be $(s + \tau_i)^{th}$ edge of v_i modulo d.

DEFINITION 3. *Universal Exploration sequences (UXS): A sequence $\tau_1\tau_2 \ldots \tau_l \in \{0, 1, \ldots d-1\}^l$ is a universal exploration sequence for d-regular graphs of size at most n if for every connected d-regular graph on at most n vertices, any numbering of its edges, and any starting edge, the walk obtained visits all the vertices of the graph. Such a sequence is called an (n, d)-universal exploration sequence.*

Following lemma suggests that UXS can be constructed in L [Rei05]:

LEMMA 4. *There exists a log-space algorithm that takes as input $(1^n, 1^d)$ and produces an (n, d)-universal exploration sequence.*

3 Log-space Algorithm for 3-connected Planar-GI

In this section, we give a log-space algorithm for 3-connected planar graph isomorphism. This, combined with the L -hardness result by [TW08] proves our main theorem:

Theorem 1 *Given two 3-connected planar graphs G and H, deciding whether G is isomorphic to H is complete for* L .

For general planar graphs, the best known parallel algorithm runs in AC^1 [MR91]. Thierauf and Wagner [TW08] recently improved the bound for the case of 3-connected planar graphs to UL ∩ coUL . This case is easier due to a result by Whitney [Whi33] that every planar 3-connected graph has precisely two planar embeddings on a sphere, where one embedding is the mirror image of the other. Moreover, one can compute these embeddings in L [AM00].

3.1 Overview of the UL ∩ coUL algorithm of [TW08]

For a 3-connected planar graph G, the algorithm by Thierauf and Wagner starts by constructing a code for every edge of G and for any of the two combinatorial embeddings. Of all these codes, the lexicographically smallest one is the code for G. The codes for two graphs are equal if and only if they are isomorphic. A code with this property is called a *canonical code* for the graph.

The main steps involved in their algorithm are as follows:

1. Construct a canonical spanning tree T, which depends upon the planar embedding of the graph and a fixed starting edge.
2. Traverse the tree and output a canonical list of edges.
3. Relabel the vertices of the graph according to this list to get the canonical code.

A canonical spanning tree in step 1 involves computation of shortest paths between pairs of vertices of G. Bourke, Tewari and Vinodchandran [BTV07] proved that planar reachability is in $\mathsf{UL} \cap \mathsf{coUL}$. Thierauf and Wagner extend their result for computing distances in planar graphs in $\mathsf{UL} \cap \mathsf{coUL}$. Once this spanning tree is constructed, the remaining steps can be executed in L.

3.2 Outline of our approach

Our approach bypasses the spanning tree construction step in the algorithm of [TW08] outlined above and thus eliminates distance computations. In that sense, we believe that this is a completely new approach for computing canonical codes for 3-connected planar graphs.

Our algorithm can be outlined as follows:

1. Given a 3-connected planar graph $G = (V, E)$, find a planar embedding ρ of G.
2. Make the graph 3-regular canonically for this embedding ρ to obtain an edge-coloured graph G' as described in Algorithm 1.
3. Find the canon of G' using Algorithm 2.

The step 1 is in log-space due to a result by Allender and Mahajan [AM00]. We prove that steps 2 and 3 can also be done in log-space. Step 3 uses the idea of UXS introduced by Koucký [Kou02]. Step 2 essentially does the preprocessing in order to make step 3 applicable.

The canonical code thus constructed is specific to the choice of the combinatorial embedding, the starting edge, and the starting vertex. Let the given 3-connected planar graphs be G and H. For G, we fix an embedding, a starting edge, and a starting vertex arbitrarily and cycle through both embeddings and all choices of the starting edge and the starting vertex for H, comparing the codes for each of them. As there are only polynomially many choices, a log-space transducer executing this loop runs only for polynomially many steps. If the canonical codes of G and H match for any of the choices, we say that G and H are isomorphic.

3.3 Making the graph 3-regular

In this section, we describe the procedure to make the graph 3-regular. In Section 3.4, we use Reingold's construction for UXS [Rei05] to come up with a canonical code. As Reingold's construction [Rei05] for UXS requires the graph to have constant degree, we do this preprocessing step. In Lemma 5, we prove that two graphs are isomorphic if and only if they are isomorphic after the preprocessing step. We note that after the preprocessing step, the graph does not remain 3-connected, however, the embedding of the new graph is inherited from the given graph. Hence even the new graph has only two possible embeddings.

We describe the preprocessing steps in Algorithm 1. Note that the new graph thus obtained has $2|E|$ vertices.

Algorithm 1 Procedure to get a 3-regular planar graph G' from 3-connected planar graph G.

Input: A 3-connected planar graph G with planar combinatorial embedding ρ.
Output: A 3-regular planar graph G' on $2m$ vertices, with edges coloured 1 and 2 and planar combinatorial embedding ρ'.

1: **for all** $v_i \in V$ **do**
2: Replace v_i of by a cycle $\{v_{i1}, \ldots, v_{id_i}\}$ on d_i vertices, where d_i is the degree of v_i.
3: The d_i edges $\{e_{i1}, \ldots, e_{id_i}\}$ incident to v_i in G are now incident to $\{v_{i1}, \ldots, v_{id_i}\}$ respectively.
4: Colour the cycle edges with colour 1.
5: Colour $e_{i1}, \ldots, e_{id_i}$ by colour 2.
6: **end for**

LEMMA 5. *Given two 3-connected planar graphs G_1, G_2, $G_1 \cong G_2$ if and only if $G_1' \cong G_2'$ where the isomorphism between G_1' and G_2' respects colours of the edges.*

PROOF. Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two 3-connected planar graphs with planar combinatorial embeddings ρ_1 and ρ_2 respectively. Let $\phi : V_1 \to V_2$ be an isomorphism between the oriented graphs (G_1, ρ_1) and (G_2, ρ_2). By isomorphism of oriented graphs we mean that the graphs are isomorphic for the fixed embeddings, in our case ρ_1 and ρ_2.

Construct G_1' and G_2' as described in Algorithm 1, replacing each vertex v of degree d by a cycle of length d, and colouring the new cycle edges with colour 1 and original edges with colour 2. The algorithm preserves the orientation of original edges from G_1 and G_2 and outputs the coloured oriented graphs (G_1', ρ_1') and (G_2', ρ_2').

Given an isomorphism ϕ between (G_1, ρ_1) and (G_2, ρ_2), we show how to derive an isomorphism ϕ' between (G_1', ρ_1') and (G_2', ρ_2'). By our construction, edges around a vertex in G_1(respectively G_2) get the same combinatorial embedding around the corresponding cycle in G_1' (G_2'). Consider an edge $\{v_i, v_j\}$ in E_1. Let $\phi(v_i) = u_k$ and $\phi(v_j) = u_l$. $\{u_k, u_l\} \in E_2$. Let corresponding edge in G_1' be $\{v_{i_p}, v_{i_q}\}$ and that in G_2' be $\{u_{k_r}, u_{k_s}\}$. Then we define a map $\phi' : V_1' \to V_2'$ which is inherited from ϕ such that $\phi'(v_{i_p}) = u_{k_r}$ and $\phi'(v_{j_q}) = u_{k_s}$. It is easy to see that ϕ' is an isomorphism for edge-coloured oriented graphs (G_1', ρ_1') and (G_2', ρ_2').

Now we show how to obtain an isomorphism ϕ between (G_1, ρ_1) and (G_2, ρ_2), given an isomorphism ϕ' between (G_1', ρ_1') and (G_2', ρ_2'). Let $e = \{v_{i_p}, v_{i_q}\} \in E_1'$ and the corresponding edge $e' = \{\phi'(v_{i_p}), \phi'(v_{i_q})\} \in E_2'$. Let v_{i_p} and v_{i_q} correspond to the same vertex v_i in G_1. Then colour of e and e' is 1. Thus ϕ' maps copies of the same vertex of G_1 to copies of a single vertex of G_2. Hence a map ϕ can be derived from ϕ' in a natural way. It is easy to see that ϕ is an isomorphism between oriented graphs (G_1, ρ_1) and (G_2, ρ_2).

3.4 Obtaining the canonical code

Lemma 5 from the previous section suggests that for given embeddings ρ_1, ρ_2 of G_1 and G_2, it suffices to check the 3-regular oriented graphs (G_1', ρ_1') and (G_2', ρ_2') for isomorphism. The Procedure $canon(G, \rho, v, e = (u, v))$ described in Algorithm 2 does this using universal exploration sequences.

Algorithm 2 Procedure $canon(G, \rho, v, e = (u, v))$

Input: Edge-coloured graph $G = (V, E)$ with maximum degree 3 and combinatorial embedding ρ, starting vertex v, starting edge $e = (u, v)$.
Output: Canon of G.

1: Construct a $(n, 3)$-universal exploration sequence U.
2: With starting vertex $v \in V$ and edge $e = (u, v)$ incident to it, traverse G according to U and ρ outputting the labels of the vertices.
3: Relabel the vertices according to their first occurrence in this output sequence, as in step 3 of [TW08].
4: For every (i, j) in this labelling, output whether (i, j) is an edge or not. If it is an edge, output its colour. This gives a canon for the graph.

We prove correctness of Algorithm 2 in the following lemma:

LEMMA 6. *Let $\sigma_1 = canon(G_1', \rho_1', v_1, e_1 = (u_1, v_1))$ and $\sigma_2 = canon(G_2', \rho_2', v_2, e_2 = (u_2, v_2))$. If $\sigma_1 = \sigma_2$ then $G_1' \cong G_2'$. Further, if $G_1' \cong G_2'$ then for some choice of ρ_2', v_2, e_2, $\sigma_1 = \sigma_2$.*

PROOF. If $G_1' \cong G_2'$, then there is a bijection $\phi : V_1' \to V_2'$ for corresponding embeddings ρ_1', ρ_2'. Let $e_1 = (u, v) \in E_1'$. Then $e_2 = (\phi(u), \phi(v)) \in E_2'$. Let e_1 and e_2 be chosen as starting edges and v and $\phi(v)$ as starting vertices for traversal using UXS U for (G_1', ρ_1') and (G_2', ρ_2') respectively. Let T_1 and T_2 be the output sequences. If a vertex $w \in V_1'$ occurs at position l in T_1 then $\phi(w) \in V_2'$ occurs at position l in T_2 as the oriented graphs are isomorphic, and the same UXS is used for their traversal. Thus the sequences are canonical when projected down to the first occurrences and hence $\sigma_1 = \sigma_2$.

Let $\sigma_1 = \sigma_2 = \sigma$. The labels of vertices in σ are just a relabelling of vertices of V_1' and V_2'. These relabellings are some permutations, say π_1 and π_2. Then $\pi_1 \cdot \pi_2^{-1} : V_1' \to V_2'$ is a bijection.

After constructing canonical code σ' for a graph G', it remains to construct canonical code σ for the original graph G. For this, we need to give a unique label to every vertex of graph G. It suffices to pick the minimum label among the labels of all its copies in G'. All copies of a vertex can be found by traversing colour 1 edges, starting from one of its copies. Thus the canonical code for graph G can be constructed in log-space as follows:

For each edge (i, j) of colour 2 in σ', traverse along the edges coloured 1 starting from i and find the minimum label among the vertices visited. Let it be p. Repeat the process for j. Let the minimum label among the vertices visited along edges of colour 1 be q. Thus the canonical labels for i and j are p and q respectively. Output the edge (p, q). The sequence thus obtained contains n distinct labels for vertices, each between $\{1, 2, \ldots, 2m\}$. This can further be converted into a sequence with labels for vertices between $\{1, 2, \ldots, n\}$ by finding the rank of each of the labels. This gives us σ. Correctness follows from the fact that vertices connected with edges of colour 1 are copies of the same vertex in G, hence they should get the same number.

Clearly, each of the above steps can be performed in L and hence the algorithm runs in L. This proves Theorem 1.

4 Conclusion

Our work settles the open question of the complexity of 3-connected planar graph isomorphism mentioned in [TW08] by giving a log-space algorithm. One of the most challenging questions is to settle the complexity of the general graph isomorphism problem. The other important goal is to improve upon the AC^1 upper bound of [MR91] for planar graph isomorphism.

5 Acknowledgement

We thank Jacobo Torán, V. Arvind, and Meena Mahajan for helpful discussions, and the anonymous referees for helpful comments.

References

[AM00] Eric Allender and Meena Mahajan. The complexity of planarity testing. In *STACS '00: Proceedings of the 17th Annual Symposium on Theoretical Aspects of Computer Science*, pages 87–98, 2000.

[BTV07] Chris Bourke, Raghunath Tewari, and N V Vinodchandran. Directed planar reachability is in unambiguous logspace. In *to appear in Proceedings of IEEE Conference on Computational Complexity CCC*, pages –, 2007.

[Bus97] Samuel R. Buss. Alogtime algorithms for tree isomorphism, comparison, and canonization. In *KGC '97: Proceedings of the 5th Kurt Gödel Colloquium on Computational Logic and Proof Theory*, pages 18–33, 1997.

[HT74] John Hopcroft and Robert Tarjan. Efficient planarity testing. *J. ACM*, 21(4):549–568, 1974.

[HW74] J. E. Hopcroft and J. K. Wong. Linear time algorithm for isomorphism of planar graphs (preliminary report). In *STOC '74: Proceedings of the sixth annual ACM symposium on Theory of computing*, pages 172–184, 1974.

[JT98] P. McKenzie B. Jenner and J. Torán. A note on the hardness of tree isomorphism. In *COCO '98: Proceedings of the Thirteenth Annual IEEE Conference on Computational Complexity*. IEEE Computer Society, 1998.

[Kou02] Michal Koucký. Universal traversal sequences with backtracking. *J. Comput. Syst. Sci.*, 65(4):717–726, 2002.

[Lin92] Steven Lindell. A logspace algorithm for tree canonization (extended abstract). In *STOC '92: Proceedings of the twenty-fourth annual ACM symposium on Theory of computing*, pages 400–404, 1992.

[Luk86] Eugene M. Luks. Parallel algorithms for permutation groups and graph isomorphism. In *FOCS*, pages 292–302, 1986.

[MR91] Gary L. Miller and John H. Reif. Parallel tree contraction part 2: further applications. *SIAM J. Comput.*, 20(6):1128–1147, 1991.

[Rei05] Omer Reingold. Undirected st-connectivity in log-space. In *STOC '05: Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, pages 376–385, 2005.

[RR90] Vijaya Ramachandran and John Reif. Planarity testing in parallel. Technical report, 1990.

[Tor00] J. Torán. On the hardness of graph isomorphism. In *FOCS '00: Proceedings of the 41st Annual Symposium on Foundations of Computer Science*, page 180, 2000.

[TW08] Thomas Thierauf and Fabian Wagner. The isomorphism problem for planar 3-connected graphs is in unambiguous logspace. In *STACS*, pages 633–644, 2008.

[Ver07] Oleg Verbitsky. Planar graphs: Logical complexity and parallel isomorphism tests. In *STACS*, pages 682–693, 2007.

[Wei66] H. Weinberg. A simple and efficient algorithm for determining isomorphism of planar triply connected graphs. *Circuit Theory*, 13:142148, 1966.

[Whi33] H. Whitney. A set of topological invariants for graphs. *American Journal of Mathematics*, 55:235–321, 1933.

Foundations of Software Technology and Theoretical Computer Science (Bangalore) 2008.
Editors: R. Hariharan, M. Mukund, V. Vinay; pp 163-174

A new upper bound for 3-SAT*

J. Díaz[1], L. Kirousis[2,3], D. Mitsche[1], X. Pérez-Giménez[1]

[1] Llenguatges i Sistemes Informàtics, UPC
E-08034 Barcelona
{diaz,dmitsche,xperez}@lsi.upc.edu

[2] RA Computer Technology Institute
GR-26504 Rio, Patras

[3] Computer Engineering and Informatics, University of Patras
GR-26504 Rio, Patras
kirousis@ceid.upatras.gr

ABSTRACT. We show that a randomly chosen 3-CNF formula over n variables with clauses-to-variables ratio at least 4.4898 is asymptotically almost surely unsatisfiable. The previous best such bound, due to Dubois in 1999, was 4.506. The first such bound, independently discovered by many groups of researchers since 1983, was 5.19. Several decreasing values between 5.19 and 4.506 were published in the years between. The probabilistic techniques we use for the proof are, we believe, of independent interest.

1 Introduction

Satisfiability of Boolean formulas is a problem universally believed to be hard. Determining the source of this hardness will lead, as is often stressed, to applications in domains even outside the realm of mathematics or computer science; moreover, and perhaps more importantly, it will enhance our understanding of the foundations of computing.

In the beginning of the 90's several groups of experimentalists chose to examine the source of this hardness from the following viewpoint: consider a *random* 3-CNF formula with a given clauses-to-variables ratio, which is known as the *density* of the formula. What is the probability of it being satisfied and how does this probability depend on the density? Their simulation results led to the conclusion that if the density is fixed and below a number approximately equal to 4.27, then for large n, a randomly chosen formula is almost always satisfiable, whereas if the density is fixed and above 4.27, a randomly chosen formula is, for large n, almost always unsatisfiable. More importantly, around 4.27 the complexity of several well known complete algorithms for checking satisfiability reaches a steep peak (see e.g. [10, 15]). So, in a certain sense, 4.27 is the point where from an empirical, statistical viewpoint the "hard" instances of SAT are to be found. Similar results were obtained for other combinatorial problems, and also for k-SAT for values of $k > 3$.

*Partially supported by the and the Spanish CYCIT: TIN2007-66523 (FORMALISM). The first and second authors were also partially supported by *La distinció per a la promoció de la recerca* de la Generalitat de Catalunya, 2002, and by the EU within the 7th Framework Programme under contract 215270 (FRONTS).

These experimental results were followed by an intense activity to provide "rigorous results" (the expression often used in this context to refer to theorems). Perhaps the most important advance is due to Friedgut: in [7] he proved that there is a sequence of reals $(c_n)_n$ such that for any $\epsilon > 0$ the probability of a randomly chosen 3-CNF-formula with density $c_n - \epsilon$ being satisfiable approaches 1 (as $n \to \infty$), whereas for density $c_n + \epsilon$, it approaches 0. Intuitively, this means that the transition from satisfiability to unsatisfiability is sharp, however it is still not known if $(c_n)_n$ converges.

Despite the fact that the convergence of $(c_n)_n$ is still an open problem, increasingly improved upper and lower bounds on its terms have been computed in a rigorous way by many groups of researchers. The currently best lower bound is 3.52 [9, 2].

With respect to upper bounds, which is the subject of this work, the progress was slower but better, in the sense that the experimentally established threshold is more closely bounded from above, rather than from below. A naïve application of the first moment method yields an upper bound of 5.191 (see e.g., [6]). An important advance was made in [8], where the upper bound was improved to 4.76. In the sequel, the work of several groups of researchers, based on more refined variants of the first moment method, culminated in the value of 4.571 [4, 11] (see the nice surveys [12, 3] for a complete sequence of the events). The core idea in these works was to use the first moment method by computing the expected number of not all satisfying truth assignments, but only of those among them that are local maxima in the sense of a lexicographic ordering, within a degree of locality determined by the Hamming distance between truth assignments (considered as binary sequences). For degree of locality 1, this amounts to computing the expected number of satisfying assignments that become unsatisfying assignments by flipping any of their "false" values (value 0) to "true" (value 1). Such assignments are sometimes referred to as *single-flip* satisfying assignments.

The next big step was taken by Dubois et al. [5], who showed that 4.506 is an upper bound. Instead of considering further variations of satisfiability, they limited the domain of computations to formulas that have a typical *syntactic* characteristic. Namely, they considered formulas where the cardinality of variables with given numbers of occurrences as positive and negative literals, respectively, approaches a two dimensional Poisson distribution. Asymptotically almost all formulas have this typical property (we say that such formulas have a Poisson *2D degree sequence*). It turns out that the expectation of the number of single-flip satisfying assignments is exponentially reduced when computed for such formulas. To get the afore mentioned upper bound, Dubois et al. further limited the domain of computations to formulas that are *positively unbalanced*, i.e. formulas where every variable has at least as many occurrences as a positive literal as it has as a negated one.

A completely different direction was recently taken in [13]. Their work was motivated by results on the geometry of satisfying assignments, and especially the way they form clusters (components where one can move from one satisfying assignment to another by hops of small Hamming distance). Most of these results were originally based on analytical, but non-rigorous, techniques of Statistical Physics; lately however important rigorous advances were made [1, 14]. The value of the upper bound obtained by Maneva and Sinclair (see [13]) was 4.453, far below any other upper bound presently known (including the one in this paper). However it was proved assuming a conjecture on the geometry of the satisfying truth

assignments which is presently proved only for k-SAT for $k \geq 8$ in [1].

In this paper, we show that 4.4898 is an upper bound. Our approach builds upon previous work. It makes use (i) of single-flip satisfying truth assignments, (ii) of formulas with a Poisson 2D degree sequence and (iii) of positively unbalanced formulas.

We add to these previously known techniques two novel elements that when combined further reduce the expectation computed. Our approach is rigorous: although we make use of computer programs, the outputs we use are formally justified. What is interesting is not that much the numerical value we get, although it constitutes a further improvement to a long series of results. The main interest lies, we believe, on one hand in the new techniques themselves and on the other in the fact that putting together so many disparate techniques necessitates a delicately balanced proof structure.

First, we start by recursively eliminating one-by-one the occurrences of pure literals from the random formula, until we get its *impure core*, i.e. the largest sub-formula with no pure literals (a pure literal is one that has at least one occurrence in the formula but whose negation has none). Obviously this elimination has no effect on the satisfiability of the formula. Since we consider random formulas with a given 2D degree sequence, we first have to determine what is the 2D degree sequence of the impure core. For this, we use the differential equation method. The setting of the differential equations is more conveniently carried out in the so called *configuration* model, where the random formula is constructed by starting with as many labelled copies of each literal as its occurrences and then by considering random 3D matchings of these copies. The matchings define the clauses. The change of models from the standard one to the configuration model with a Poisson 2D degree sequence is formalized in Lemma 2. We also take care of the fact that the configuration model allows formulas with (i) multiple clauses and (ii) multiple occurrences of the same variable in a clause, whereas we are interested in simple formulas, i.e. formulas where neither (i) nor (ii) holds. For our purposes, it is enough to bound from below the probability of getting a simple formula in the configuration model by $e^{-\Theta(n^{1/3}\log n)}$, see Lemma 3. The differential equations are then analytically solved, and we thus obtain the 2D degree sequence of the core, see Proposition 4.

Second, we require that not only the 2D degree sequence is Poisson, but also that the numbers of clauses with none, one, two and three positive literals, respectively, are close to the expected numbers. Notice that these expected numbers have to reflect the fact that we consider positively unbalanced formulas. This is formalized in Lemma 5.

The expectation of the number of satisfying assignments, in the framework determined by all the restrictions above, is computed in Lemma 6. This expectation turns out to be given by a sum of polynomially many terms of functions that are exponential in n. We estimate this sum by its maximum term, using a standard technique. However in this case, finding the maximum term entails maximizing a function of many variables whose number depends on n. To avoid a maximization that depends on n we prove a truncation result which allows us to consider formulas that have a Poisson 2D degree sequence only for *light* variables, i.e. variables whose number of occurrences, either as positive or negated literals, is at most a constant independent of n.

Then we carry out the maximization. The technique we use is the standard one by Lagrange multipliers. We get a complex 3×3 system which can be solved numerically. We

formally prove that the system does not maximize on the boundary of the system and we make a sweep over the domain which confirms the results of the numerical solution.

Due to lack of space, all proofs are omitted or just sketched in this extended abstract. As usual, *asymptotically almost surely* (a.a.s.) will mean with probability tending to 1 as $n \to \infty$. All asymptotic expressions as $1 - o(1)$ are always with respect to n. Our main result in the paper is the following:

THEOREM 1. *Let $\gamma = 4.4898$ and $m = \lfloor \gamma n \rfloor$. A random 3-CNF formula in $\mathcal{F}_{n,m}$ (i.e. with n variables and m clauses, no repetition of clauses and no repetition of variables in a clause) is not satisfiable a.a.s.*

2 Background and Technical highlights.

Consider a given set of n Boolean variables, and let $m = \lfloor \gamma n \rfloor$. Let $\mathcal{F}_{n,m}$ be the set of 3-CNF formulas with n variables and m clauses, where repetition of clauses or repetition of variables in a clause is not allowed. We also denote by $\mathcal{F}_{n,m}$ the probability space of formulas in $\mathcal{F}_{n,m}$ drawn with uniform probability. Throughout the paper, we fix the value $\gamma = 4.4898$ and prove that for that value a random 3-CNF formula is not satisfiable with high probability.

Throughout the paper, *scaled* will always mean divided by n, and a *scaled natural* will be a member of $\frac{1}{n}\mathbb{N}$. Given a formula $\phi \in \mathcal{F}_{n,m}$, we define the following parameters which depend on ϕ: For any $i, j \in \mathbb{N}$, let $d_{i,j}$ be the scaled number of variables with i positive occurrences and j negative occurrences in ϕ. Then,

$$\sum_{i,j \in \mathbb{N}} d_{i,j} = 1. \tag{1}$$

The sequence $\boldsymbol{d} = (d_{i,j})_{i,j \in \mathbb{N}}$ is called the *degree sequence* of ϕ. The scaled number of clauses of ϕ is denoted by c, and can be expressed by

$$c(\boldsymbol{d}) = \frac{1}{3} \sum_{i,j \in \mathbb{N}} (i+j) d_{i,j}. \tag{2}$$

Note that if $\phi \in \mathcal{F}_{n,m}$, then c must additionally satisfy $c = \lfloor \gamma n \rfloor / n$.

Given $\epsilon_1 > 0$ and any sequence $\boldsymbol{\xi} = (\xi_{i,j})_{i,j \in \mathbb{N}}$ of nonnegative reals with $\sum_{i,j \in \mathbb{N}} \xi_{i,j} = 1$, define

$$\mathcal{N}(n, \boldsymbol{\xi}, \epsilon_1) = \Big\{ \boldsymbol{d} = (d_{i,j})_{i,j \in \mathbb{N}} \ : \ \sum_{i,j \in \mathbb{N}} d_{i,j} = 1, \quad \frac{n}{3} \sum_{i,j \in \mathbb{N}} (i+j) d_{i,j} \in \mathbb{N}, \quad \forall i, j \in \mathbb{N} \quad d_{i,j} n \in \mathbb{N},$$
$$\text{and} \quad |d_{i,j} - \xi_{i,j}| \le \epsilon_1, \quad \text{and if } i > n^{1/6} \text{ or } j > n^{1/6} \text{ then } d_{i,j} = 0 \Big\}.$$

Intuitively $\mathcal{N}(n, \boldsymbol{\xi}, \epsilon_1)$ can be interpreted as the set of degree sequences $\boldsymbol{d}$ which are close to the ideal sequence $\boldsymbol{\xi}$, which in general is not a degree sequence since its entries $\xi_{i,j}$ need not be scaled naturals. However, if n is large enough, then $\mathcal{N}(n, \boldsymbol{\xi}, \epsilon_1) \neq \emptyset$. Now we consider the 2D Poisson ideal sequence $\boldsymbol{\delta}$ defined by $\delta_{i,j} = e^{-3\gamma}(3\gamma/2)^{i+j}/(i!j!)$. The following lemma reflects the fact that almost all $\phi \in \mathcal{F}_{n,m}$ have a degree sequence $\boldsymbol{d}$ which is close to $\boldsymbol{\delta}$. A proof of an analogous result can be found in [5].

LEMMA 2. *Let $\boldsymbol{d}$ be the degree sequence of a random $\phi \in \mathcal{F}_{n,m}$. For any $\epsilon_1 > 0$, we have that $\mathbf{Pr}_{\mathcal{F}_{n,m}}(\boldsymbol{d} \in \mathcal{N}(n, \delta, \epsilon_1)) = 1 - o(1)$.*

Given a fixed degree sequence $\boldsymbol{d} = (d_{i,j})_{i,j\in\mathbb{N}}$ satisfying (1) and such that $c = c(\boldsymbol{d})$ defined by (2) is also a scaled natural, we wish to generate 3-CNF formulas with that particular degree sequence $\boldsymbol{d}$. A natural approach to this is to use the *configuration model*. A *configuration* φ with degree sequence $\boldsymbol{d} = (d_{i,j})_{i,j\in\mathbb{N}}$ is constructed as follows: consider n variables and the corresponding $2n$ literals $x_1, \bar{x}_1 \ldots, x_n, \bar{x}_n$; each literal has a certain number of distinct labelled *copies* in a way that the scaled number of variables with i positive copies and j negative copies is $d_{i,j}$; then partition the set of copies into sets of size 3, which we the call clauses of φ. Let $\mathcal{C}_{n,\boldsymbol{d}}$ be the set of all configurations with degree sequence $\boldsymbol{d}$, and we also denote by $\mathcal{C}_{n,\boldsymbol{d}}$ the probability space on the set $\mathcal{C}_{n,\boldsymbol{d}}$ with the uniform distribution.

A 3-CNF *multi-formula* is a formula with possible repetition of variables in one clause and/or possible repetition of clauses. A *simple formula* is a formula in $\mathcal{F}_{n,m}$. Let π be the projection from $\mathcal{C}_{n,\boldsymbol{d}}$ to 3-CNF multi-formulas obtained by unlabelling the copies of each literal. A configuration $\varphi \in \mathcal{C}_{n,\boldsymbol{d}}$ is satisfiable if $\phi = \pi(\varphi)$ is satisfiable. A configuration $\varphi \in \mathcal{C}_{n,\boldsymbol{d}}$ is *simple* iff $\phi = \pi(\varphi)$ is a simple formula, i.e. does not have repetition of variables or clauses. Notice that the number of anti-images of a simple formula ϕ with degree sequence $\boldsymbol{d}$ under π does not depend on the particular choice of ϕ. Hence,

$$\mathbf{Pr}_{\mathcal{F}_{n,m}}(\phi \text{ is SAT} \mid \boldsymbol{d}) = \mathbf{Pr}_{\mathcal{C}_{n,\boldsymbol{d}}}(\varphi \text{ is SAT} \mid \text{SIMPLE}). \tag{3}$$

We need a lower bound on the probability that a configuration is simple. The following result gives a weak bound which is enough for our purposes.

LEMMA 3.
Let $\epsilon_1 > 0$ and $\boldsymbol{d} \in \mathcal{N}(n, \delta, \epsilon_1)$. Then

$$\mathbf{Pr}_{\mathcal{C}_{n,\boldsymbol{d}}}(\text{SIMPLE}) \geq e^{-\Theta(n^{1/3}\log n)},$$

where the $e^{-\Theta(n^{1/3}\log n)}$ bound is uniform for all $\boldsymbol{d} \in \mathcal{N}(n, \delta, \epsilon_1)$.

Given $\varphi \in \mathcal{C}_{n,\boldsymbol{d}}$, a *pure* variable of φ is a variable which has a non-zero number of occurrences which are either all syntactically positive or all syntactically negative. The only literal occurring in φ and all its copies are also called *pure*. If φ is satisfiable and x is a pure variable of φ, then there exists some satisfying truth assignment of φ which satisfies all copies of x in φ. Hence, in order to study the satisfiability of a $\varphi \in \mathcal{C}_{n,\boldsymbol{d}}$, we can satisfy each pure variable in φ and remove all clauses containing a copy of that variable. For each $\varphi \in \mathcal{C}_{n,\boldsymbol{d}}$, let $\widetilde{\varphi}$ be the configuration obtained by greedily removing all pure variables and their corresponding clauses from φ. This $\widetilde{\varphi}$ is independent of the particular elimination order of pure literals and is called the *impure core* of φ. In fact, in our analysis we will eliminate only one clause containing one copy of a pure literal at a time (the $\widetilde{\varphi}$ obtained still remains the same). Note that φ is satisfiable iff $\widetilde{\varphi}$ is satisfiable. Moreover, if φ is simple then $\widetilde{\varphi}$ is also simple (but the converse is not necessarily true).

Furthermore, let $\widehat{\varphi}$ be the configuration obtained from $\widetilde{\varphi}$ by positively unbalancing all variables, i.e. switching the syntactic sign of those variables having initially more negative than positive occurrences in $\widetilde{\varphi}$. Let $\widehat{\mathcal{C}}_{n,\boldsymbol{d}}$ denote the probability space of configurations $\widehat{\varphi}$,

where φ was chosen from $\mathcal{C}_{n,\boldsymbol{d}}$ with uniform probability. Note that the probability distribution in $\widehat{\mathcal{C}}_{n,\boldsymbol{d}}$ is not necessarily uniform. Since the simplicity and the satisfiability of a configuration are not affected by positively unbalancing the variables, we have

$$\mathbf{Pr}_{\mathcal{C}_{n,\boldsymbol{d}}}(\varphi \text{ is SAT} \wedge \text{SIMPLE}) \leq \mathbf{Pr}_{\widehat{\mathcal{C}}_{n,\boldsymbol{d}}}(\widehat{\varphi} \text{ is SAT} \wedge \text{SIMPLE}). \tag{4}$$

Let the random variable $\widehat{\boldsymbol{d}}$ be the degree sequence of a random configuration in $\widehat{\mathcal{C}}_{n,\boldsymbol{d}}$. We prove in the following result that if the original $\boldsymbol{d}$ is close to the ideal sequence $\boldsymbol{\delta}$, then with high probability $\widehat{\boldsymbol{d}}$ must be close to the ideal sequence $\widehat{\boldsymbol{\delta}} = (\widehat{\delta}_{i,j})_{i,j\in\mathbb{N}}$ defined by

$$\widehat{\delta}_{i,j} = \begin{cases} 2e^{-3\gamma b}\frac{(3\gamma b/2)^{i+j}}{i!j!}, & \text{if } i > j, \\ e^{-3\gamma b}\frac{(3\gamma b/2)^{i+j}}{i!j!}, & \text{if } i = j, \\ 0, & \text{if } i < j, \end{cases}$$

where $b = (1 - t_{\mathcal{D}}/\gamma)^{2/3}$ and $t_{\mathcal{D}}$ is the scaled number of steps in the pure literal elimination algorithm.

PROPOSITION 4. *Given $\epsilon_2 > 0$, there exists $\epsilon_1 > 0$ and $0 < \beta < 1$ such that for any $\boldsymbol{d} \in \mathcal{N}(n, \boldsymbol{\delta}, \epsilon_1)$*

$$\mathbf{Pr}_{\widehat{\mathcal{C}}_{n,\boldsymbol{d}}}\left(\widehat{\boldsymbol{d}} \in \mathcal{N}(n, \widehat{\boldsymbol{\delta}}, \epsilon_2)\right) = 1 - O(\beta^{n^{1/2}}).$$

Moreover, for each $\widehat{\boldsymbol{d}} \in \mathcal{N}(n, \widehat{\boldsymbol{\delta}}, \epsilon_2)$, the probability space $\widehat{\mathcal{C}}_{n,\boldsymbol{d}}$ conditional upon having degree sequence $\widehat{\boldsymbol{d}}$ has the uniform distribution (i.e. $\widehat{\mathcal{C}}_{n,\boldsymbol{d}}$ conditional upon a fixed $\widehat{\boldsymbol{d}}$ behaves exactly as $\mathcal{C}_{n,\widehat{\boldsymbol{d}}}$).

Let $\widehat{\boldsymbol{d}} \in \mathcal{N}(n, \widehat{\boldsymbol{\delta}}, \epsilon_2)$. Then, each $\varphi \in \mathcal{C}_{n,\widehat{\boldsymbol{d}}}$ has a scaled number of clauses of $\widehat{c} = c(\widehat{\boldsymbol{d}})$ (see (2)). Moreover, let ℓ_{p} and ℓ_{n} be the scaled number of copies in φ of positive and of negative literals respectively. Then

$$\ell_{\mathrm{p}}(\widehat{\boldsymbol{d}}) = \sum_{i,j\in\mathbb{N}} i\widehat{d}_{i,j}, \qquad \ell_{\mathrm{n}}(\widehat{\boldsymbol{d}}) = \sum_{i,j\in\mathbb{N}} j\widehat{d}_{i,j}. \tag{5}$$

Given any fixed $\varphi \in \mathcal{C}_{n,\widehat{\boldsymbol{d}}}$ and for $k \in \{0, \ldots, 3\}$, let $\widehat{c}_k$ be the scaled number of clauses in φ containing exactly k positive copies (clauses of *syntactic type* k). We call $\widehat{\boldsymbol{c}} = (\widehat{c}_0, \ldots, \widehat{c}_3)$ the *clause-type sequence* of φ. By definition

$$\widehat{c}_1 + 2\widehat{c}_2 + 3\widehat{c}_3 = \ell_{\mathrm{p}}, \qquad 3\widehat{c}_0 + 2\widehat{c}_1 + \widehat{c}_2 = \ell_{\mathrm{n}}, \tag{6}$$

and by adding the equations in (6), $\widehat{c}_0 + \cdots + \widehat{c}_3 = \widehat{c}$. The $\widehat{c}_0, \ldots, \widehat{c}_3$ are random variables in $\mathcal{C}_{n,\widehat{\boldsymbol{d}}}$, but the next result shows that if $\widehat{\boldsymbol{d}}$ is close enough to $\widehat{\boldsymbol{\delta}}$, then $\widehat{c}_0, \cdots, \widehat{c}_3$ as well as their sum $\widehat{c}_0 + \cdots + \widehat{c}_3 = \widehat{c}$ are concentrated with high probability. In order to see this, we need to define $\widehat{\gamma} = c(\widehat{\boldsymbol{\delta}})$, $\lambda_{\mathrm{p}} = \ell_{\mathrm{p}}(\widehat{\boldsymbol{\delta}})$ and $\lambda_{\mathrm{n}} = \ell_{\mathrm{n}}(\widehat{\boldsymbol{\delta}})$ (see (2) and (5)), which can be interpreted as the limit of $\widehat{c}$, ℓ_{p} and ℓ_{n} respectively when $\widehat{\boldsymbol{d}}$ approaches $\widehat{\boldsymbol{\delta}}$. In terms of these numbers, we thus define for all $k \in \{0, \ldots, 3\}$

$$\widehat{\gamma}_k = \binom{3}{k}\frac{{\lambda_{\mathrm{p}}}^k{\lambda_{\mathrm{n}}}^{3-k}}{(\lambda_{\mathrm{p}} + \lambda_{\mathrm{n}})^3}\,\widehat{\gamma} \tag{7}$$

and also $\widehat{\gamma} = (\widehat{\gamma}_0, \ldots, \widehat{\gamma}_3)$. Then we have $\widehat{\gamma}_1 + 2\widehat{\gamma}_2 + 3\widehat{\gamma}_3 = \lambda_{\mathrm{p}}$, $3\widehat{\gamma}_0 + 2\widehat{\gamma}_1 + \widehat{\gamma}_2 = \lambda_{\mathrm{n}}$ and $\widehat{\gamma}_0 + \widehat{\gamma}_1 + \widehat{\gamma}_2 + \widehat{\gamma}_3 = \widehat{\gamma}$.

The next result shows that when $\widehat{\boldsymbol{d}}$ is close enough to $\widehat{\delta}$, then each $\widehat{c}_k$ is close to the corresponding $\widehat{\gamma}_k$. Indeed, given $\epsilon > 0$ and for any $\widehat{\boldsymbol{d}} \in \mathcal{N}(n, \widehat{\delta}, \epsilon_2)$, let $\mathcal{C}^{\epsilon}_{n,\widehat{\boldsymbol{d}}}$ be the set of all $\varphi \in \mathcal{C}_{n,\widehat{\boldsymbol{d}}}$ such that for $k \in \{0, \ldots, 3\}$, $|\widehat{c}_k - \widehat{\gamma}_k| \leq \epsilon$. We also denote by $\mathcal{C}^{\epsilon}_{n,\widehat{\boldsymbol{d}}}$ the corresponding uniform probability space.

LEMMA 5. *Given $\epsilon > 0$, there is $\epsilon_2 > 0$ and $0 < \beta < 1$ such that for any $\widehat{\boldsymbol{d}} \in \mathcal{N}(n, \widehat{\delta}, \epsilon_2)$,*

$$\mathbf{Pr}_{\mathcal{C}_{n,\widehat{\boldsymbol{d}}}}(\mathcal{C}^{\epsilon}_{n,\widehat{\boldsymbol{d}}}) = 1 - O(\beta^n).$$

All the previous lemmata establish a connection between the uniform probability spaces $\mathcal{F}_{n,m}$ and $\mathcal{C}^{\epsilon}_{n,\widehat{\boldsymbol{d}}}$. In order to prove Theorem 1, it remains to bound the probability that a configuration $\varphi \in \mathcal{C}^{\epsilon}_{n,\widehat{\boldsymbol{d}}}$ is simple and satisfiable, as it is done in the following result.

LEMMA 6. *There exists $\epsilon > 0$ and $0 < \beta < 1$ such that for any $\widehat{\boldsymbol{d}} \in \mathcal{N}(n, \widehat{\delta}, \epsilon)$,*

$$\mathbf{Pr}_{\mathcal{C}^{\epsilon}_{n,\widehat{\boldsymbol{d}}}}(\textsc{Sat} \wedge \textsc{Simple}) = O(\beta^n).$$

The proof of Lemma 6 is sketched in Section 3 below. The proof of Theorem 1 then follows from all the previous lemmata (see the full version for the proof).

3 Proof of Lemma 6

Let $\mathcal{N}(n, \widehat{\delta}, \widehat{\gamma}, \epsilon)$ be the set of tuples $(\widehat{\boldsymbol{d}}, \widehat{\boldsymbol{c}})$ such that $\widehat{\boldsymbol{d}} \in \mathcal{N}(n, \widehat{\delta}, \epsilon)$ and $\widehat{\boldsymbol{c}} = (\widehat{c}_k)_{0 \leq k \leq 3}$ is a tuple of scaled naturals satisfying (6) (recall also from (5) the definition of ℓ_{p} and ℓ_{n}), and moreover $|\widehat{c}_k - \widehat{\gamma}_k| \leq \epsilon$. For each $(\widehat{\boldsymbol{d}}, \widehat{\boldsymbol{c}}) \in \mathcal{N}(n, \widehat{\delta}, \widehat{\gamma}, \epsilon)$, we define $\mathcal{C}_{n,\widehat{\boldsymbol{d}},\widehat{\boldsymbol{c}}}$ to be the uniform probability space of all configurations with degree sequence $\widehat{\boldsymbol{d}}$ and clause-type sequence $\widehat{\boldsymbol{c}}$. In order to prove the lemma, it suffices to show that for any $(\widehat{\boldsymbol{d}}, \widehat{\boldsymbol{c}}) \in \mathcal{N}(n, \widehat{\delta}, \widehat{\gamma}, \epsilon)$ we have $\mathbf{Pr}_{\mathcal{C}_{n,\widehat{\boldsymbol{d}},\widehat{\boldsymbol{c}}}}(\textsc{Sat} \wedge \textsc{Simple}) = O(\beta^n)$. Hence, we consider $\widehat{\boldsymbol{d}}$, $\widehat{\boldsymbol{c}}$ and the probability space $\mathcal{C}_{n,\widehat{\boldsymbol{d}},\widehat{\boldsymbol{c}}}$ to be fixed throughout this section, and we try to find a suitable bound for $\mathbf{Pr}(\textsc{Sat} \wedge \textsc{Simple})$.

We need some definitions. Let us fix any given configuration $\varphi \in \mathcal{C}_{n,\widehat{\boldsymbol{d}},\widehat{\boldsymbol{c}}}$. A *light* variable of φ is a variable with $i \leq M$ positive occurrences and $j \leq M$ negative occurrences in φ (we use in the numerical calculations the value $M = 23$). The other variables are called *heavy*. We consider a weaker notion of satisfiability in which heavy variables are treated as jokers and are always satisfied regardless of their sign in the formula and their assigned value. Given a configuration $\varphi \in \mathcal{C}_{n,\widehat{\boldsymbol{d}},\widehat{\boldsymbol{c}}}$ and a truth assignment A, we say that $A \models^{\flat} \varphi$ iff each clause of φ contains at least one heavy variable or at least one satisfied occurrence of a light variable. Let $\textsc{Sat}^{\flat}$ be the set of configurations $\varphi \in \mathcal{C}_{n,\widehat{\boldsymbol{d}},\widehat{\boldsymbol{c}}}$ for which there exists at least one truth assignment A such that $A \models^{\flat} \varphi$. Clearly, if $A \models \varphi$, then also $A \models^{\flat} \varphi$, and hence $\textsc{Sat} \subset \textsc{Sat}^{\flat}$. We still introduce a further restriction to satisfiability in a way similar to [11] and [4], in order to decrease the number of satisfying truth assignments of each configuration without altering the set of satisfiable configurations (at least without alterating this set for simple configurations). Given a configuration $\varphi \in \mathcal{C}_{n,\widehat{\boldsymbol{d}},\widehat{\boldsymbol{c}}}$ and a truth assignment A, we say that

$A \models^{\flat\prime} \varphi$ iff $A \models^{\flat} \varphi$ and moreover each light variable which is assigned the value zero by A appears at least once as the only satisfied literal of a *blocking clause* (i.e. a clause with one satisfied negative literal and two unsatisfied ones). Let $\text{SAT}^{\flat\prime}$ be the set of configurations which are satisfiable according to this latter notion. Notice that if $\varphi \in \text{SIMPLE}$, then $\varphi \in \text{SAT}^{\flat\prime}$ iff $\varphi \in \text{SAT}^{\flat}$ (by an argument analogous to the one in [11] and [4]). Therefore, we have $\mathbf{Pr}(\text{SAT} \wedge \text{SIMPLE}) \leq \mathbf{Pr}(\text{SAT}^{\flat} \wedge \text{SIMPLE}) = \mathbf{Pr}(\text{SAT}^{\flat\prime} \wedge \text{SIMPLE}) \leq \mathbf{Pr}(\text{SAT}^{\flat\prime})$. Let X be the random variable counting the number of satisfying truth assignments of a randomly chosen configuration $\varphi \in \mathcal{C}_{n,\widehat{\boldsymbol{d}},\widehat{\boldsymbol{c}}}$ in the $\text{SAT}^{\flat\prime}$ sense. We need to bound

$$\mathbf{Pr}(\text{SAT}^{\flat\prime}) = \mathbf{Pr}(X > 0) \leq \mathbf{E}X = \frac{|\{(\varphi, A) \ : \ \varphi \in \mathcal{C}_{n,\widehat{\boldsymbol{d}},\widehat{\boldsymbol{c}}},\ A \models^{\flat\prime} \varphi\}|}{|\mathcal{C}_{n,\widehat{\boldsymbol{d}},\widehat{\boldsymbol{c}}}|}. \tag{8}$$

In the following subsection, we obtain an exact but complicated expression for $\mathbf{E}X$ by a counting argument, and then we give a simple asymptotic bound which depends on the maximization of a particular continuous function over a bounded polytope. The next subsection contains the maximization of that function.

3.1 Asymptotic bound on $\mathbf{E}X$

First, we compute the denominator of the rightmost member in (8).

$$\begin{aligned}
|\mathcal{C}_{n,\widehat{\boldsymbol{d}},\widehat{\boldsymbol{c}}}| &= \binom{n}{(\widehat{d}_{i,j}n)_{i,j}} \binom{\ell_{\mathtt{p}}n}{\widehat{c}_1 n, 2\widehat{c}_2 n, 3\widehat{c}_3 n} \binom{\ell_{\mathtt{n}}n}{3\widehat{c}_0 n, 2\widehat{c}_1 n, \widehat{c}_2 n} \frac{(3\widehat{c}_0 n)!}{(\widehat{c}_0 n)! 6^{\widehat{c}_0 n}} \frac{(2\widehat{c}_1 n)!}{2^{\widehat{c}_1 n}} \frac{(2\widehat{c}_2 n)!}{2^{\widehat{c}_2 n}} \frac{(3\widehat{c}_3 n)!}{(\widehat{c}_3 n)! 6^{\widehat{c}_3 n}} \\
&= \frac{n!}{\prod_{i,j}(\widehat{d}_{i,j}n)!} \frac{(\ell_{\mathtt{p}}n)!(\ell_{\mathtt{n}}n)!}{2^{\widehat{c}n} 3^{(\widehat{c}_0+\widehat{c}_3)n} (\widehat{c}_0 n)!(\widehat{c}_1 n)!(\widehat{c}_2 n)!(\widehat{c}_3 n)!}
\end{aligned}$$

In order to deal with the numerator in (8), we need some definitions. Let us consider any fixed $\varphi \in \mathcal{C}_{n,\widehat{\boldsymbol{d}},\widehat{\boldsymbol{c}}}$ and any assignment A such that $A \models^{\flat\prime} \varphi$. We will classify the variables, the clauses and the copies of literals in φ into several types, and define parameters counting the scaled number of items of each type. Variables are classified according to their degree. A variable is said to have degree (i, j) if it appears i times positively and j times negatively in φ. Let $\mathcal{L}$ and $\mathcal{H}$, respectively, be the set of possible degrees for light and heavy variables, i.e. $\mathcal{L} = \{(i, j) \in \mathbb{N}^2 \ : \ 0 \leq i, j \leq M\}$, $\mathcal{H} = \{(i, j) \in \mathbb{N}^2 \ : \ i > M \text{ or } j > M\}$. We also consider an extended notion of degree for light variables which are assigned 0 by A. One of such variables has extended degree (i, j, k) if it has degree (i, j) and among its j negative occurrences k appear in a blocking clause (being the only satisfied literal of the clause). Let $\mathcal{L}' = \{(i, j, k) \in \mathbb{N}^3 \ : \ 0 \leq i \leq M,\ 1 \leq k \leq j \leq M\}$, be the set of possible extended degrees for these light 0-variables. For each $(i, j) \in \mathcal{L}$, let $t_{i,j}$ be the scaled number of light variables assigned 1 by A with degree (i, j) in φ. For each $(i, j, k) \in \mathcal{L}'$, let $f_{i,j,k}$ be the scaled number of light variables assigned 0 by A with extended degree (i, j, k) in φ. We must have

$$t_{i,j} + \sum_{k=1}^{j} f_{i,j,k} = \widehat{d}_{i,j}, \qquad \forall (i, j) \in \mathcal{L}. \tag{9}$$

On the other hand, we classify the copies of literals occurring in φ into five different types depending on their sign in φ, their assignment by A and whether they belong or not to a

blocking clause. Each copy receives a label from the set $\mathcal{S} = \{\mathtt{ps}, \mathtt{ns1}, \mathtt{ns2}, \mathtt{pu}, \mathtt{nu}\}$, where the labels ps, pu, ns1, ns2 and nu denote positive-satisfied, positive-unsatisfied, negative-satisfied in a blocking clause, negative-satisfied in a non-blocking clause and negative-unsatisfied, respectively. It is useful to consider as well coarser classifications of the copies of literals in φ and thus we define the types p, n and ns which correspond to positive, negative and negative-satisfied copies, respectively. Also, let $\mathcal{S}' = \{\mathtt{ps}, \mathtt{ns}, \mathtt{pu}, \mathtt{nu}\}$ and $\mathcal{S}'' = \{\mathtt{p}, \mathtt{n}\}$. For each of the types $\sigma \in \mathcal{S} \cup \mathcal{S}' \cup \mathcal{S}''$ that we defined, let ℓ_σ be the scaled number of copies of type σ. Note that $\ell_\mathtt{p}$ and $\ell_\mathtt{n}$ were already defined (see (5) and (6)). Also, let h_σ be the scaled number of copies of type σ which come from heavy variables (recall that these copies are always satisfied by definition regardless of their sign). In view of $h_\mathtt{ps} = \sum_\mathcal{H} i\widehat{d}_{i,j}$, $h_\mathtt{ns} = \sum_\mathcal{H} j\widehat{d}_{i,j}$ and of (5) and (6), we observe that $\ell_\mathtt{p}$, $\ell_\mathtt{n}$, $h_\mathtt{ps}$ and $h_\mathtt{ns}$ are constants which do not depend on the particular choice of (φ, A). The parameters $h_\mathtt{ns1}$ and $h_\mathtt{ns2}$ depend on the particular (φ, A) and satisfy

$$h_\mathtt{ns1} + h_\mathtt{ns2} = h_\mathtt{ns}. \tag{10}$$

The parameters $\ell_\mathtt{ps}$, $\ell_\mathtt{pu}$, $\ell_\mathtt{ns1}$, $\ell_\mathtt{ns2}$ and $\ell_\mathtt{nu}$ also depend on (φ, A) and can be expressed as

$$\ell_\mathtt{ps} = \sum_\mathcal{L} i t_{i,j} + h_\mathtt{ps}, \qquad \ell_\mathtt{pu} = \sum_{\mathcal{L}'} i f_{i,j,k}, \qquad \ell_\mathtt{ns1} = \sum_{\mathcal{L}'} k f_{i,j,k} + h_\mathtt{ns1},$$
$$\ell_\mathtt{ns2} = \sum_{\mathcal{L}'} (j-k) f_{i,j,k} + h_\mathtt{ns2}, \qquad \ell_\mathtt{nu} = \sum_\mathcal{L} j t_{i,j}. \tag{11}$$

Finally, the clauses of φ are classified into 16 extended types (not to be mistaken with the four syntactic types defined immediately before (6)). Each type is represented by a 2×2 matrix from the set $\mathcal{A} = \left\{ \alpha = \begin{pmatrix} \mathtt{ps}(\alpha) & \mathtt{ns}(\alpha) \\ \mathtt{pu}(\alpha) & \mathtt{nu}(\alpha) \end{pmatrix} \in \mathbb{N}^4 \ : \ \sum_{\sigma \in \mathcal{S}'} \sigma(\alpha) = 3,\ \mathtt{ps}(\alpha) + \mathtt{ns}(\alpha) > 0 \right\}$. A clause is said to be of extended type $\alpha = \begin{pmatrix} \mathtt{ps}(\alpha) & \mathtt{ns}(\alpha) \\ \mathtt{pu}(\alpha) & \mathtt{nu}(\alpha) \end{pmatrix}$ if for each $\sigma \in \mathcal{S}'$ the clause contains $\sigma(\alpha)$ copies of literals of type σ. Notice that all clauses of extended type α also contain the same number of copies of type σ for all other $\sigma \in \mathcal{S} \cup \mathcal{S}''$ and thus we can define $\sigma(\alpha)$ to be this number. For each $\alpha \in \mathcal{A}$, let c_α be the scaled number of clauses of extended type α (while $\widehat{c}_k$, $0 \leq k \leq 3$ is the number of clauses of syntactic type k, i.e. with k positive literals). We have

$$\sum_{\substack{\alpha \in \mathcal{A} \\ \mathtt{p}(\alpha) = k}} c_\alpha = \widehat{c}_k. \tag{12}$$

The parameters $\ell_\mathtt{ps}$, $\ell_\mathtt{pu}$, $\ell_\mathtt{ns1}$, $\ell_\mathtt{ns2}$ and $\ell_\mathtt{nu}$ can also be expressed in terms of the c_α by

$$\ell_\sigma = \sum_{\alpha \in \mathcal{A}} \sigma(\alpha) c_\alpha, \quad \forall \sigma \in \mathcal{S}. \tag{13}$$

We now consider the following equations:

$$\ell_\mathtt{ps} + \ell_\mathtt{pu} = \ell_\mathtt{p} \qquad \ell_\mathtt{ns1} + \ell_\mathtt{ns2} + \ell_\mathtt{nu} = \ell_\mathtt{n} \tag{14}$$

$$\ell_\mathtt{ps} = \sum_\mathcal{L} i t_{i,j} + h_\mathtt{ps} \qquad \ell_\mathtt{ns1} = \sum_{\mathcal{L}'} k f_{i,j,k} + h_\mathtt{ns1} \qquad \ell_\mathtt{ns2} = \sum_{\mathcal{L}'} (j-k) f_{i,j,k} + h_\mathtt{ns2} \tag{15}$$

$$\ell_\mathtt{ps} = \sum_{\alpha \in \mathcal{A}} \mathtt{ps}(\alpha) c_\alpha \qquad \ell_\mathtt{ns1} = \sum_{\alpha \in \mathcal{A}} \mathtt{ns1}(\alpha) c_\alpha \qquad \ell_\mathtt{ns2} = \sum_{\alpha \in \mathcal{A}} \mathtt{ns2}(\alpha) c_\alpha \tag{16}$$

In view of (5) and (6), the system of equations {(9), (10), (11), (12), (13)} is equivalent to {(9), (10), (12), (14), (15), (16)}.

So far we verified that the constraints {(9), (10), (12), (14), (15), (16)} express necessary conditions for the parameters of any particular (φ, A), with $\varphi \in \mathcal{C}_{n,\widehat{\boldsymbol{d}},\widehat{\boldsymbol{c}}}$ and $A \models^{\flat\prime} \varphi$. Now we will see that they are also sufficient, in the sense that for each tuple of parameters satisfying the above-mentioned constraints we will be able to construct pairs (φ, A).

Let $\bar{t} = (t_{i,j})_{\mathcal{L}}$, $\bar{f} = (f_{i,j,k})_{\mathcal{L}'}$, $\bar{h} = (h_{\mathtt{ns}1}, h_{\mathtt{ns}2})$, $\bar{c} = (c_\alpha)_{\alpha\in\mathcal{A}}$, $\bar{\ell} = (\ell_\sigma)_{\sigma\in\mathcal{S}}$ and $K = |\mathcal{L}| + |\mathcal{L}'| + 2 + |\mathcal{A}| + |\mathcal{S}| = (M+1)^2(1+M/2) + 23$. We define the bounded polytope $\mathcal{P}(\widehat{\boldsymbol{d}},\widehat{\boldsymbol{c}}) \subset \mathbb{R}^K$ as the set of tuples $\bar{x} = (\bar{t},\bar{f},\bar{h},\bar{c},\bar{\ell})$ of non-negative reals satisfying {(9), (10), (12), (14), (15), (16)}, and consider the following set of lattice points in $\mathcal{P}(\widehat{\boldsymbol{d}},\widehat{\boldsymbol{c}})$: $\mathcal{I}(n,\widehat{\boldsymbol{d}},\widehat{\boldsymbol{c}}) = \mathcal{P}(\widehat{\boldsymbol{d}},\widehat{\boldsymbol{c}}) \cap \left(\frac{1}{n}\mathbb{N}\right)^K$. For any tuple of parameters $\bar{x} \in \mathcal{I}(n,\widehat{\boldsymbol{d}},\widehat{\boldsymbol{c}})$, we count the number of pairs (φ, A), with $\varphi \in \mathcal{C}_{n,\widehat{\boldsymbol{d}},\widehat{\boldsymbol{c}}}$ and $A \models^{\flat\prime} \varphi$, satisfying these parameters. We denote this number by $T(\bar{x},n,\widehat{\boldsymbol{d}},\widehat{\boldsymbol{c}})$. We obtain (see the full version for details)

$$T(\bar{x},n,\widehat{\boldsymbol{d}},\widehat{\boldsymbol{c}}) = 2^{\sum_{\mathcal{H}} \widehat{d}_{i,j} n} \binom{n}{(t_{i,j}n)_{\mathcal{L}}, (f_{i,j,k}n)_{\mathcal{L}'}, (\widehat{d}_{i,j}n)_{\mathcal{H}}} \left(\prod_{\mathcal{L}'} \binom{j}{k}^{f_{i,j,k}n}\right) \binom{h_{\mathtt{ns}}n}{h_{\mathtt{ns}1}n, h_{\mathtt{ns}2}n} \prod_{\sigma\in\mathcal{S}} \binom{\ell_\sigma n}{(\sigma(\alpha)c_\alpha n)_{\alpha\in\mathcal{A}}} \prod_{\alpha\in A} W(\alpha),$$

where $W(\alpha) = \frac{(w(\alpha)c_\alpha n)!(c_\alpha n)!^{2-w(\alpha)}}{(w(\alpha)!)^{c_\alpha n}}$, and $w(\alpha)$ is the number of 0's in the matrix α. Hence $\mathbf{E}X = \frac{1}{|\mathcal{C}_{n,\widehat{\boldsymbol{d}},\widehat{\boldsymbol{c}}}|} \sum_{\bar{x}\in\mathcal{I}(n,\widehat{\boldsymbol{d}},\widehat{\boldsymbol{c}})} T(\bar{x},n,\widehat{\boldsymbol{d}},\widehat{\boldsymbol{c}})$.

To characterize the asymptotic behaviour of $T(\bar{x},n,\widehat{\boldsymbol{d}},\widehat{\boldsymbol{c}})/|\mathcal{C}_{n,\widehat{\boldsymbol{d}},\widehat{\boldsymbol{c}}}|$ with respect to n, we define

$$F(\bar{x}) = \frac{\prod_{\sigma\in\mathcal{S}} \ell_\sigma{}^{\ell_\sigma}}{\prod_{\mathcal{L}} t_{i,j}{}^{t_{i,j}} \prod_{\mathcal{L}'} \left(f_{i,j,k}/\binom{j}{k}\right)^{f_{i,j,k}} h_{\mathtt{ns}1}{}^{h_{\mathtt{ns}1}} h_{\mathtt{ns}2}{}^{h_{\mathtt{ns}2}} \prod_{\alpha\in\mathcal{A}} \left((w(\alpha)!/2)c_\alpha\right)^{c_\alpha}}$$

and

$$B(\widehat{\boldsymbol{d}},\widehat{\boldsymbol{c}}) = 2^{\sum_{\mathcal{H}} \widehat{d}_{i,j}} h_{\mathtt{ns}}{}^{h_{\mathtt{ns}}} \prod_{\mathcal{L}} \widehat{d}_{i,j}^{\,\widehat{d}_{i,j}} \frac{3^{c_0+c_3} c_0{}^{c_0} c_1{}^{c_1} c_2{}^{c_2} c_3{}^{c_3}}{\ell_{\mathtt{p}}{}^{\ell_{\mathtt{p}}} \ell_{\mathtt{n}}{}^{\ell_{\mathtt{n}}}}.$$

By Stirling's inequality we obtain $\frac{T(\bar{x},n,\widehat{\boldsymbol{d}},\widehat{\boldsymbol{c}})}{|\mathcal{C}_{n,\widehat{\boldsymbol{d}},\widehat{\boldsymbol{c}}}|} \leq \mathrm{poly}_1(n)\big(B(\widehat{\boldsymbol{d}},\widehat{\boldsymbol{c}})F(\bar{x})\big)^n$, where $\mathrm{poly}_1(n)$ is some fixed polynomial in n which can be chosen to be independent of $\bar{x}$, $\widehat{\boldsymbol{d}}$ and $\widehat{\boldsymbol{c}}$ (as long as $\bar{x} \in \mathcal{I}(n,\widehat{\boldsymbol{d}},\widehat{\boldsymbol{c}})$ and $(\widehat{\boldsymbol{d}},\widehat{\boldsymbol{c}}) \in \mathcal{N}(n,\widehat{\boldsymbol{\delta}},\widehat{\boldsymbol{\gamma}},\epsilon)$). Moreover, since the size of $\mathcal{I}(n,\widehat{\boldsymbol{d}},\widehat{\boldsymbol{c}})$ is also polynomial in n, we can write

$$\mathbf{E}X \leq \mathrm{poly}_2(n) \left(B(\widehat{\boldsymbol{d}},\widehat{\boldsymbol{c}}) \max_{\bar{x}\in\mathcal{I}(n,\widehat{\boldsymbol{d}},\widehat{\boldsymbol{c}})} F(\bar{x})\right)^n \leq \mathrm{poly}_2(n) \left(B(\widehat{\boldsymbol{d}},\widehat{\boldsymbol{c}}) \max_{\bar{x}\in\mathcal{P}(n,\widehat{\boldsymbol{d}},\widehat{\boldsymbol{c}})} F(\bar{x})\right)^n,$$

for some other fixed polynomial $\mathrm{poly}_2(n)$. By continuity, if we choose ϵ to be small enough, we can guarantee that

$$\mathbf{E}X \leq \left((1+10^{-7})B \max_{\bar{x}\in\mathcal{P}(n,\widehat{\boldsymbol{\delta}},\widehat{\boldsymbol{\gamma}})} F(\bar{x})\right)^n, \tag{17}$$

where (recall the definition in (7))

$$B = B(\widehat{\delta},\widehat{\gamma}) = 2^{\sum_{\mathcal{H}} \widehat{\delta}_{i,j}} \left(\sum_{\mathcal{H}} j\widehat{\delta}_{i,j}\right)^{\sum_{\mathcal{H}} j\widehat{\delta}_{i,j}} \prod_{\mathcal{L}} \widehat{\delta}_{i,j}^{\widehat{\delta}_{i,j}} \frac{3^{\widehat{\gamma}_0+\widehat{\gamma}_3}\,\widehat{\gamma}_0^{\widehat{\gamma}_0}\widehat{\gamma}_1^{\widehat{\gamma}_1}\widehat{\gamma}_2^{\widehat{\gamma}_2}\widehat{\gamma}_3^{\widehat{\gamma}_3}}{\lambda_{\mathtt{p}}^{\lambda_{\mathtt{p}}}\lambda_{\mathtt{n}}^{\lambda_{\mathtt{n}}}}$$

$$= 2^{\sum_{\mathcal{H}} \widehat{\delta}_{i,j}} \left(\sum_{\mathcal{H}} j\widehat{\delta}_{i,j}\right)^{\sum_{\mathcal{H}} j\widehat{\delta}_{i,j}} \frac{\prod_{\mathcal{L}} \widehat{\delta}_{i,j}^{\widehat{\delta}_{i,j}}}{(3\widehat{\gamma})^{2\widehat{\gamma}}}. \tag{18}$$

3.2 Maximization of $F(\bar{x})$

We wish to maximize F or equivalently $\log F$ over the domain $\mathcal{P}(n,\widehat{\delta},\widehat{\gamma})$. We need the following lemma:

LEMMA 7. *$F(\bar{x})$ does not maximize on the boundary of $\mathcal{P}(n,\widehat{\delta},\widehat{\gamma})$.*

Since $\log F$ does not maximize on the boundary of its domain, the maximum must be attained at a critical point of $\log F$ in the interior of $\mathcal{P}(n,\widehat{\delta},\widehat{\gamma})$. We use the Lagrange multipliers technique and characterize each critical point of $\log F$ in terms of the solution of a 3×3 system. The system is numerically solved with the help of Maple, which finds just one solution. We express the maximum of F over $\mathcal{P}(n,\widehat{\delta},\widehat{\gamma})$ in terms of this solution, and multiply it by B given in (18), and from (17) we obtain the bound

$$\mathbf{E}X \le \left((1+10^{-7})0.9999998965\right)^n, \tag{19}$$

which concludes the proof of Lemma 6, since $(1+10^{-7})0.9999998965 < 1$.

Note that the validity of our approach relies on the assumption that the solution of the 3×3 system found by Maple is unique, which implies that the critical point of $\log F$ we found is indeed the global maximum (if an alternative solution exists it could happen that at the corresponding critical point the function F attains a value greater than the maximum obtained).

In order to be more certain about the correctness of (19) we performed the following alternative experiment: Let $\mathcal{P}_{\bar{\ell}}$ be the polytope obtained by restricting $\mathcal{P}(n,\widehat{\delta},\widehat{\gamma})$ to the coordinates $\ell_{\mathtt{ps}},\ell_{\mathtt{pu}},\ell_{\mathtt{ns1}},\ell_{\mathtt{ns2}},\ell_{\mathtt{nu}}$. Observe that this is a 3-dimensional polytope in $\mathbb{R}^5$, since its elements are determined by the values of the coordinates $\ell_{\mathtt{ps}},\ell_{\mathtt{ns1}},\ell_{\mathtt{ns2}}$. We performed a sweep over this polytope by considering a grid of 100 equispaced points in each of the three dimensions. For each of the 100^3 fixed tuples of $(\ell_{\mathtt{ps}},\ell_{\mathtt{ns1}},\ell_{\mathtt{ns2}})$ which correspond to the points on the grid, we determine the remaining two coordinates of $\mathcal{P}_{\bar{\ell}}$, and maximize $\log F$ restricted to those fixed values of $\bar{\ell}$. Observe that in this case $\log F$ is strictly concave and thus has a unique maximum which can be efficiently found by any iterative Newton-like algorithm. We checked, again using Maple, that the value obtained for each fixed tuple of $\bar{\ell}$ is below the maximum in (19).

References

[1] D. Achlioptas and F. Ricci-Tersenghi. On the solution-space geometry of random constraint satisfaction problems. In *38th Annual ACM Symposium on Theory of Computing (STOC)*, pages 130–139, 2006.

[2] D. Coppersmith, D. Gamarnik, M. T. Hajiaghayi, and G. Sorkin. Random Max SAT, Random Max Cut, and their phase transitions. *Random Structures and Algorithms*, 24:502–545, 2004.

[3] O. Dubois. Upper bound on the satisfiability threshold. *Theoretical Computer Science*, 265:187–197, 2001.

[4] O. Dubois and Y. Boufkhad. A general upper bound for the satisfiability threshold of random r-SAT formulae. *Journal of Algorithms*, 24:395–420, 1997.

[5] O. Dubois, Y. Boufkhad, and J. Mandler. Typical random 3-SAT formulae and the satisfiability threshold. In *Proceedings of the 11th Symposium on Discrete Algorithms (SODA)*, pages 126–127, 2000.

[6] J. Franco and M. Paull. Probabilistic analysis of the Davis-Putnam procedure for solving the satisfiability problem. *Discrete Applied Mathematics*, 5:77–87, 1983.

[7] E. Friedgut and J. Bourgain. Sharp thresholds of graph properties, and the *k*-SAT problem. *Journal of the American Mathematical Society*, 12:1017–1054, 1999.

[8] A. Kamath, R. Motwani, K. Palem, and P. Spirakis. Tail bounds for occupancy and the satisfiability threshold conjecture. *Random Structures and Algorithms*, 7:59–80, 1995.

[9] A. C. Kaporis, L. M. Kirousis, and E. G. Lalas. The probabilistic analysis of a greedy satisfiability algorithm. *Random Structures and Algorithms*, 28:444–480, 2006.

[10] S. Kirkpatrick and B. Selman. Critical behavior in the satisfiability of random Boolean expressions. *Science*, 264(5163):1297–1301, 1994.

[11] L. M. Kirousis, E. Kranakis, D. Krizanc, and Y. C. Stamatiou. Approximating the unsatisfiability threshold of random formulas. *Random Structures and Algorithms*, 12:253–269, 1998.

[12] L. M. Kirousis, Y. C. Stamatiou, and M. Zito. The satisfiability threshold conjecture: Techniques behind upper bound. In A. P. G. Istrate and C. Moore, editors, *Computational Complexity and Statistical Physics*. Oxford University Press, 2005.

[13] E. Maneva and A. Sinclair. On the satisfiability threshold and clustering of solutions of random 3-SAT formulas. *Theoretical Computer Science*, 2008.

[14] M. Mézard and R. Zecchina. Random k-satisfiability: from an analytic solution to a new efficient algorithm. *Physics Review*, E-66, 056126:1357–1361, 2002.

[15] R. Monasson and R. Zecchina. Statistical mechanics of the random *k*-SAT problem. *Physics Review*, E-56:1357–1361, 1997.

[16] N. C. Wormald. The differential equation method for random graph processes and greedy algorithms. In M. Karoński and H. J. Prömel, editors, *Lectures on approximation and randomized algorithms*, pages 73–155. PWN, Warsaw, 1999.

Foundations of Software Technology and Theoretical Computer Science (Bangalore) 2008.
Editors: R. Hariharan, M. Mukund, V. Vinay; pp 175-186

Abstraction Refinement for Games with Incomplete Information*

Rayna Dimitrova†, Bernd Finkbeiner
Universität des Saarlandes
{dimitrova,finkbeiner}@cs.uni-sb.de

ABSTRACT.Counterexample-guided abstraction refinement (CEGAR) is used in automated software analysis to find suitable finite-state abstractions of infinite-state systems. In this paper, we extend CEGAR to games with incomplete information, as they commonly occur in controller synthesis and modular verification. The challenge is that, under incomplete information, one must carefully account for the knowledge available to the player: the strategy must not depend on information the player cannot see. We propose an abstraction mechanism for games under incomplete information that incorporates the approximation of the players' moves into a knowledge-based subset construction on the abstract state space. This abstraction results in a perfect-information game over a finite graph. The concretizability of abstract strategies can be encoded as the satisfiability of strategy-tree formulas. Based on this encoding, we present an interpolation-based approach for selecting new predicates and provide sufficient conditions for the termination of the resulting refinement loop.

1 Introduction

Infinite games are a natural model of reactive systems as they capture the ongoing interaction between a system and its environment. Many problems in automated software analysis, including controller synthesis and modular verification, can be reduced to finding (or deciding the existence of) a winning strategy. The design of algorithms for solving such games is complicated by the following two challenges: First, games derived from software systems usually have an infinite (or finite, but very large) state space. Second, the games are usually played under incomplete information: it is unrealistic to assume that a system has full access to the global state, e.g., that a process can observe the private variables of the other processes.

The most successful approach to treat infinite state spaces in software verification is predicate abstraction with counterexample-guided abstraction refinement (CEGAR) [3, 1]. For games with complete information [7, 4], CEGAR builds abstractions that overapproximate the environment's moves and underapproximate the system's moves. If the system wins the abstract game it is guaranteed to also win the concrete game. If the environment wins the abstract game, one checks if the strategy is spurious in the sense that it contains an abstract state from which the strategy cannot be concretized. If such a state exists, the state is split to ensure that the strategy is eliminated from further consideration.

For games with incomplete information, the situation is more complicated, because the strategic capabilities of a player depend not only on the available moves, but also on the

*This work was partly supported by the German Research Foundation (DFG) as part of the Transregional Collaborative Research Center Automatic Verification and Analysis of Complex Systems (SFB/TR 14 AVACS).
†Supported by a Microsoft Research European PhD Scholarship and by an IMPRS-CS PhD Scholarship.

FSTTCS 2008
IARCS Annual Conference on
Foundations of Software Technology and Theoretical Computer Science
http://drops.dagstuhl.de/opus/volltexte/2008/1751

knowledge about the state of the game. If the abstract game provides less information to the system than the concrete game, then the environment may spuriously win the abstract game, because the abstract system may be unable to distinguish a certain pair of states and may therefore be forced to apply the *same* move in the two states where the concrete system can select *different* moves. An abstraction refinement approach for games with incomplete information must therefore carefully account for the information collected by the system. A first requirement is that the refinement should avoid predicates that mix variables that are observable to the system with those that are hidden. Such mixed predicates lead to the situation that the concrete system has partial information (the values of the observable variables), while the abstract system does not know the value of the predicate at all. Since the system may collect information over multiple steps of a play, however, just separating the variables alone is not enough. Consider, for example, a situation where, in order to win, the system has to react with output $x_o \approx 0$ if some hidden variable x_h has value $x_h \approx 0$ and with output $x_o \approx 1$ if $x_h \approx 1$. Now, suppose the system is able to deduce the value of x_h from the prefix that leads to the state, because an observable rational-valued variable x_i is either always positive or always negative if $x_h \approx 0$ and flips its sign otherwise. To rule out the spuriously winning strategy for the environment, it is necessary to refine the abstraction with the new predicate $x_i > 0$, even though the system wins for *any* value of x_i.

Contributions. In this paper, we propose the first CEGAR approach for games with incomplete information. We extend the abstraction of the game with a subset construction on the abstract state space that ensures that the system only uses information it can see. The result is a perfect-information game over a finite game graph that soundly abstracts the original game under incomplete information.

The refinement of the abstraction accounts for two cases: we *refine the abstract transition relations* by adding new predicates if the environment spuriously wins because it uses moves that are impossible in the concrete game or because moves of the system are impossible in the abstract game but possible in the concrete game; we *refine the observation equivalence* by adding new predicates if the environment spuriously wins because the abstract system has too little information. To ensure that the new predicates do not mix observable and unobservable variables, we develop a novel constraint-based interpolation technique which provides interpolants that meet arbitrary variable partitioning requirements.

The resulting refinement loop terminates for games for which a finite region algebra (that satisfies certain conditions related to the observation-equivalence) exists. This includes important infinite-state models such as timed games or games defined by bounded rectangular automata, given that the observation-equivalence meets the requirement.

In the following, due to space constraints, all proofs and some technical details have been omitted. We refer the reader to the full version of this paper [6].

Related work. The classic solution to games with incomplete information is the translation to perfect-information games with a *knowledge-based subset construction* due to Reif [9]. For games over infinite graphs, however, this construction is in general not effective. Our approach is symbolic and is therefore suited to the analysis of games over infinite state spaces. For incomplete-information games with finite state spaces, an alternative would be to first use the knowledge-based subset construction to obtain a perfect-information concrete game and then apply the CEGAR technique of [7] in the usual way. However, since the

subset construction leads to an exponential blow-up of the state space of the game, which for realistic systems will make the problem practically infeasible, it is imperative to first use predicate abstraction and obtain a much smaller state space and only then construct the subsets of observation-equivalent prefixes. *Symbolic fixed-point algorithms* based on antichains were proposed in [5, 2]. In the case of infinite game graphs, these algorithms are applied on a given finite region algebra for the infinite-state game. Our approach, on the other hand, automatically constructs a sufficiently precise finite abstraction. *Interpolation* was applied successfully in verification for the generation of refinement predicates. There one infers from an unconcretizable abstract counterexample-trace predicates, each of which refers only to variables that describe a single state on that trace. In our case we need to consider sets of traces each of which is concretizable and that are represented symbolically using sets of variables whose intersection contains observable variables only. The straightforward application of existing interpolation methods ([8, 10]) would produce refinement predicates that are either guaranteed to be observable or guaranteed not to relate two or more states. These approaches are incapable of meeting both guarantee requirements. To this end, we present our extension of the algorithm from [10] which provides interpolants that meet arbitrary variable partitioning requirements.

2 Preliminaries

Variables, predicates and formulas. We model the communication between a system and its environment with a finite set X of *variables*, which is partitioned into four pairwise disjoint sets: X_h, X_i, X_o and $\{t\}$. The environment updates (and can observe) the variables in X_h and X_i and the system updates (and can observe) the variables in X_o. The variables in X_i are the input variables for the system, i.e., it can read their value but not update them. The variables in X_h are private variables for the environment, i.e., the system cannot even observe them. The set X_o consists of the output variables of the system which can be only read by the environment. The value of the auxiliary variable t determines whether it is the system's or the environment's turn to make a transition, i.e., the two players take turns in making a transition. The set X' consists of the primed versions of the variables in X.

Sets of concrete and abstract states and transitions are represented as *formulas* over some possibly infinite set $\mathcal{AP}$ of *predicates* (*atomic formulas*) over the variables in $X \cup X'$. For a formula φ, we denote with $Vars(\varphi)$ and $Preds(\varphi)$ the sets of variables and predicates, respectively, that occur in φ. For a set $\mathcal{P}$ of predicates, the set $Obs(\mathcal{P})$ consists of the predicates in $\mathcal{P}$ that contain only observable variables, i.e., from $Obs(X \cup X') = (X \cup X') \setminus (X_h \cup X'_h)$.

Game structures. A *game structure with perfect information* $C = (S_s, S_e, s_0, R_s, R_e)$ consists of a set of states $S = S_s \cup S_e$, which is partitioned into a set S_s of *system states* and a set S_e of *environment states*, a distinguished initial state $s_0 \in S$, and a transition relation $R = R_s \cup R_e$, where $R_s \subseteq S_s \times S_e$ (when the system makes a transition, it always gives back the turn to the environment) and $R_e \subseteq S_e \times S$ are the transition relations for the system and the environment respectively. A *game structure with incomplete information* $(S_s, S_e, s_0, \equiv, R_s, R_e)$ additionally defines an *observation equivalence* $\equiv$ on S. The system has partial knowledge about the current state, i.e., it knows the equivalence class of the current state, but not the particular state in this class. We require that the relation $\equiv$ meets the following two require-

ments. The relation $\equiv$ respects the partitioning of S into S_s and S_e: If $v_1 \in S_s$ and $v_2 \in S_e$ then $v_1 \not\equiv v_2$. The system can distinguish between the different successors of a system state: For every $v \in S_s$ and $w_1, w_2 \in S_e$, if $(v, w_1) \in R_s$, $(v, w_2) \in R_s$ and $w_1 \neq w_2$, then $w_1 \not\equiv w_2$. The set of available transitions in a system state is the same for all observation-equivalent states: For every states $v_1, v_2 \in S_s$ and $w_1 \in S_e$ such that $v_1 \equiv v_2$ and $(v_1, w_1) \in R_s$, there exists a state $w_2 \in S_e$ such that $w_1 \equiv w_2$ and $(v_2, w_2) \in R_s$. A state v for which there is no $w \in S$ with $(v, w) \in R$ is called a *dead-end*.

We use a symbolic representation of game structures. A *symbolic game structure with incomplete information* $\mathcal{C} = (X, init, \mathcal{T}_s, \mathcal{T}_e)$ consists of a set of variables X (partitioned into X_h, X_i, X_o and $\{t\}$), a formula *init* over X and formulas $\mathcal{T}_s$ and $\mathcal{T}_e$ over $X \cup X'$. For simplicity, we assume that we have singleton sets $X_h = \{x_h\}$, $X_i = \{x_i\}$ and $X_o = \{x_o\}$ (the extension to the general case is trivial). The formulas are required to satisfy the following conditions: (1) $\mathcal{T}_e$ implies $t \approx 0$ and $x'_o \approx x_o$, (2) $\mathcal{T}_s$ implies $t \approx 1$, $t' \approx 0$, $x'_h \approx x_h$ and $x'_i \approx x_i$, (3) the formula $\mathcal{T}_s\{x_h \mapsto x_h^1\} \leftrightarrow \mathcal{T}_s\{x_h \mapsto x_h^2\}$ is valid.

Let H, I and O be the domains of x_h, x_i and x_o respectively. We assume that the set O of possible *outputs* for the system is finite. We denote with c_o the constant from the signature corresponding to an element $o \in O$ and with C_o the set of all constants for elements of O. The domain of t is $\{0, 1\}$. The set $Val(X)$ consists of all total functions that map each variable in X to its domain. For a formula φ over X, and $v \in Val(X)$ we denote with $\varphi[v]$ the truth value of the formula φ for the valuation v of the variables. We write $v \models \varphi$ iff $\varphi[v]$ is true. For a formula φ over $X \cup X'$, $v \in Val(X)$ and $w \in Val(X')$, $\varphi[v, w]$ is defined analogously.

A symbolic game structure $\mathcal{C} = (X, init, \mathcal{T}_s, \mathcal{T}_e)$ together with corresponding variable domains defines a game structure with incomplete information $C = (S_s, S_e, s_0, \equiv, R_s, R_e)$ in the following way. The sets S_s and S_e consist of the valuations in $Val(X)$ where t is mapped to 1 and 0 respectively. Since the variable x_h cannot be observed by the system, two states are observation-equivalent if they agree on the valuation of the variables in $Obs(X)$. We require that *init* is satisfied by a single initial state s_0. The formulas $\mathcal{T}_s$ and $\mathcal{T}_e$ define the transition relations, where $(v, w) \in R_s$ iff $\mathcal{T}_s[v, w]$ is true, and R_e is defined analogously.

For a formula φ and $c_o \in C_o$, $Pre_s(c_o, \varphi)$ is a formula such that $v \models Pre_s(c_o, \varphi)$ iff there exists $w \models \varphi \wedge x_o \approx c_o$ such that $(v, w) \in R_s$, $Pre_s(\varphi) = \bigvee_{c_o \in C_o} Pre_s(c_o, \varphi)$ and $Pre_e(\varphi)$ is a formula such that $v \models Pre_e(\varphi)$ iff there exists $w \models \varphi$ such that $(v, w) \in R_e$.

Safety games. We consider safety games defined by a set of *error states*, which we assume w.l.o.g. to be a subset of S_e. The objective for the system is to avoid the error states. Clearly, w.l.o.g. we can assume that S does not contain dead-ends and that for every $v \in S_s$ and $c_o \in C_o$, $v \models Pre_s(c_o, true)$. A *safety game* with perfect information (with incomplete information) $G = (C, E)$ consists of a game structure C with complete information (with incomplete information) and a set of error states E. A *symbolic safety game* $\mathcal{G} = (\mathcal{C}, err)$ consists of a symbolic game structure $\mathcal{C}$ and a formula *err* denoting the set of error states.

Strategies. Let G be a safety game. A *path* in G is a finite sequence $\pi = v_0 v_1 \dots v_n$ of states such that for all $0 \leq j < n$, we have $(v_j, v_{j+1}) \in R$. The length $|\pi|$ of π is $n + 1$. For $0 \leq j < |\pi|$, $\pi[j]$ is the j-th element of π and $\pi[0, j] = v_0 \dots v_j$. We define $last(\pi) = \pi[|\pi| - 1]$. A *prefix* in G is a path $\pi = v_0 v_1 \dots v_n$ such that $v_0 = s_0$. We call π a *system prefix* if $last(\pi) \in S_s$, and an *environment prefix* otherwise. We denote with $Prefs(G)$ the set of prefixes in G, and with $Prefs_s(G)$ and $Prefs_e(G)$ the sets of system and environment prefixes,

respectively. A *play* in G is either an infinite sequence $\omega = v_0 v_1 \dots v_j \dots$ with $v_0 = s_0$ and for all $j \geq 0$, $(v_j, v_{j+1}) \in R$ or a prefix π such that $last(\pi)$ is an error state. For an infinite play ω, $|\omega| = \infty$. The observation-equivalence $\equiv$ can be extended in a natural way to prefixes and plays. A *strategy for the system* is a function $f_s : Prefs_s(G) \to S_e$ such that if $f_s(\pi) = v$, then $(last(\pi), v) \in R_s$. Strategies for the environment are defined analogously. A strategy f_s for the system in an incomplete-information game is called *consistent* iff for all $\pi_1, \pi_2 \in Prefs_s(G)$ with $\pi_1 \equiv \pi_2$, it holds that $f_s(\pi_1) \equiv f_s(\pi_2)$. The outcome of two strategies f_s and f_e is a play $\omega = Outcome(f_s, f_e)$ such that for all $0 \leq j < |\omega|$ if $\omega[j] \in S_s$ then $\omega[j+1] = f_s(\omega[0, j])$ and if $\omega[j] \in S_e$ then $\omega[j+1] = f_e(\omega[0, j])$. A strategy f_s for the system is *winning* iff for every strategy f_e for the environment, if $\omega = Outcome(f_s, f_e)$ then for every $j \geq 0$, $\omega[j]$ is not an error state. A strategy f_e for the environment is winning iff for every strategy f_s for the system, if $\omega = Outcome(f_s, f_e)$ then for some j, $\omega[j]$ is an error state.

Strategy trees. A winning strategy f_e for the environment in a safety game G can be naturally represented as a finite tree $T(f_e)$, called *strategy tree*. Each node in $T(f_e)$ is labeled by a state in S, such that the following are satisfied: (1) the root is labeled by the initial state s_0, (2) if an internal node is labeled by a state v and a child of that node is labeled by a state w, then $(v, w) \in R$, (3) if an internal node π is labeled by $v \in S_s$, then for every $w \in S$ with $(v, w) \in R_s$, there exists exactly one child of π which is labeled by w, and $Children(\pi, T(f_e))$ is the set of all children of π in $T(f_e)$, (4) if an internal node π is labeled by $v \in S_e$, then that node has exactly one child, denoted by $Child(\pi, T(f_e))$, labeled by some $w \in S$ with $(v, w) \in R_e$, (5) a node is a leaf iff it is labeled by an error state. Thus, each node corresponds to a prefix in G, and a prefix in $Prefs(G)$ is represented by at most one node. We identify each node with the corresponding prefix and define $Prefs(f_e)$ as the set of prefixes in $T(f_e)$.

Knowledge-based subset construction. The *knowledge-based subset construction* of an incomplete-information game $G = ((S_s, S_e, s_0, \equiv, R_s, R_e), E)$ is a perfect-information game $G^k = ((S_s^k, S_e^k, s_0^k, R_s^k, R_e^k), E^k)$ defined as follows: $S_s^k = \{V \in 2^{S_s} \setminus \{\emptyset\} \mid \forall v_1, v_2 \in V.\ v_1 \equiv v_2\}$; $S_e^k = \{V \in 2^{S_e} \setminus \{\emptyset\} \mid \forall v_1, v_2 \in V.\ v_1 \equiv v_2\}$; $s_0^k = \{s_0\}$; $(V, W) \in R_s^k$ iff $V \in S_s^k$, $W \in S_e^k$ and (1) for every $v \in V$ there is a $w \in W$ such that $(v, w) \in R_s$, (2) for every $w \in W$ there is a $v \in V$ such that $(v, w) \in R$ and (3) if $w_1 \equiv w_2$, $w_1 \in W$ and there is a $v \in V$ with $(v, w_2) \in R$ then $w_2 \in W$; $(V, W) \in R_e^k$ iff $V \in S_e^k$, $W \in S_s^k \cup S_e^k$ and (1′), (2) and (3) are satisfied, where (1′) there exist $v \in V$ and $w \in W$ such that $(v, w) \in R_e$; $E^k = \{V \in S_e^k \mid V \cap E \neq \emptyset\}$.

The game solving problem. The *game solving problem* is to determine whether there exists a consistent winning strategy for the system player in a given safety game with incomplete information. The *strategy synthesis problem* is to find such a strategy if one exists.

3 Abstraction

We use two subset constructions to abstract infinite-state games with incomplete information into finite-state games with perfect information: first, we overapproximate the moves of the environment and underapproximate the moves of the system in the abstract domain defined by the predicate valuations. Then, we overapproximate the observation-equivalence based on the observable predicates to obtain a sound abstraction.

Let $\mathcal{G} = (\mathcal{S}, err)$ be a symbolic safety game. For a finite set of predicates $\mathcal{P}$ over X, $Vals(\mathcal{P})$ is the set of all valuations of the elements of $\mathcal{P}$. For $a \in Vals(\mathcal{P})$, and $p \in \mathcal{P}$, $[a]$ is

the corresponding formula over $\mathcal{P}$ and we write $a \models p$ iff the value of p in a is *true*. Similarly for a formula φ over $\mathcal{P}$. The concretization $\gamma_{\mathcal{P}}(a)$ of $a \in \mathit{Vals}(\mathcal{P})$ is the set of concrete states $\{s \in S \mid \forall p \in \mathcal{P} : s \models p \text{ iff } a \models p\}$. For $A \subseteq \mathit{Vals}(\mathcal{P})$, we define $\gamma_{\mathcal{P}}(A) = \bigcup_{a \in A} \gamma_{\mathcal{P}}(a)$. For a_1 and a_2 in $\mathit{Vals}(\mathcal{P})$, we define $a_1 \equiv^a_{\mathcal{P}} a_2$ iff for every $p \in \mathit{Obs}(\mathcal{P})$, $a_1 \models p$ iff $a_2 \models p$.

We abstract a concrete game w.r.t. a pair $\mathcal{P} = (\mathcal{P}_{se}, \mathcal{P}_s)$ of finite sets of predicates such that $\mathit{Preds}(\mathit{init}) \cup \mathit{Preds}(\mathit{err}) \cup \{t \approx 0\} \subseteq \mathcal{P}_{se}$. The states in S_e are abstracted w.r.t. $\mathcal{P}_{se}$ and the states in S_s are abstracted w.r.t. the full set $\mathcal{P} = \mathcal{P}_{se} \cup \mathcal{P}_s$. We require that $\mathit{PredsSyst}(\mathcal{P}_{se}) \subseteq \mathcal{P}_s$, where $\mathit{PredsSyst}(\mathcal{Q}) = \bigcup_{a \in \mathit{Vals}(\mathit{Obs}(\mathcal{Q}))} \mathit{Preds}(\mathit{Pre}_s([a]))$, to ensure the absence of dead-ends in the abstract game. By refining $\mathcal{P}_s$ with predicates that are used to split only abstract system states, we ensure the monotonicity of the abstraction of R_s. In the following, $\gamma(a)$ means $\gamma_{\mathcal{P}_{se}}(a)$ if $a \in \mathit{Vals}(\mathcal{P}_{se})$ and $\gamma_{\mathcal{P}}(a)$ if $a \in \mathit{Vals}(\mathcal{P})$. Similarly for $\equiv^a$.

For two pairs of sets of predicates $\mathcal{P} = (\mathcal{P}_{se}, \mathcal{P}_s)$ and $\mathcal{Q} = (\mathcal{Q}_{se}, \mathcal{Q}_s)$, we write $\mathcal{P} \subseteq \mathcal{Q}$ iff $\mathcal{P}_{se} \subseteq \mathcal{Q}_{se}$ and $\mathcal{P}_s \subseteq \mathcal{Q}_s$, and define $\mathcal{P} \cup \mathcal{Q} = (\mathcal{P}_{se} \cup \mathcal{Q}_{se}, \mathcal{P}_s \cup \mathcal{Q}_s)$.

The *abstraction* $\alpha(\mathcal{G}, \mathcal{P})$ of $\mathcal{G} = (\mathcal{S}, \mathit{err})$ w.r.t. a pair $\mathcal{P} = (\mathcal{P}_{se}, \mathcal{P}_s)$ of finite sets of predicates is the perfect-information safety game $G^a = ((S^a_s, S^a_e, s^a_0, R^a_s, R^a_e), E^a)$ defined below.

States. The set S^a of abstract states is the union of $S^a_s \subseteq 2^{\mathit{Vals}(\mathcal{P})} \setminus \{\emptyset\}$ and $S^a_e \subseteq 2^{\mathit{Vals}(\mathcal{P}_{se})} \setminus \{\emptyset\}$ which are defined as follows. An element A of $2^{\mathit{Vals}(\mathcal{P})} \setminus \{\emptyset\}$ belongs to S^a_s iff (1) for every $a \in A$, $a \not\models t \approx 0$ and $\gamma(a) \neq \emptyset$ and (2) for every $a_1, a_2 \in A$, $a_1 \equiv^a a_2$. Similarly, an element A of $2^{\mathit{Vals}(\mathcal{P}_{se})} \setminus \{\emptyset\}$ belongs to S^a_e iff (1) for every $a \in A$, $a \models t \approx 0$ and $\gamma(a) \neq \emptyset$ and (2) for every $a_1, a_2 \in A$, $a_1 \equiv^a a_2$. The initial abstract state s^a_0 consists of the single element a_0 of S^a such that $a_0 \models \mathit{init}$ and $\gamma(a_0) \neq \emptyset$.

May transitions. The abstract transition relation $R^a_e \subseteq S^a_e \times S^a$ for the environment is defined as: $(A, A') \in R^a_e$ iff the following are satisfied: (1 **may**) there exist $a \in A$, $v \in \gamma(a)$, $a' \in A'$ and $v' \in \gamma(a')$ with $(v, v') \in R_e$, (2) for every $a' \in A'$ there exist $a \in A$, $v \in \gamma(a)$ and $v' \in \gamma(a')$ such that $(v, v') \in R$ and (3) for every $a'_1 \in \mathit{Vals}(\mathcal{P})$ and $a'_2 \in \mathit{Vals}(\mathcal{P})$, if $a'_1 \in A'$, $a'_1 \equiv^a a'_2$ and there exist $a \in A$, $v \in \gamma(a)$ and $v' \in \gamma(a'_2)$ with $(v, v') \in R$, then $a'_2 \in A'$.

Must transitions. The abstract transition relation $R^a_s \subseteq S^a_s \times S^a_e$ for the system is defined as: $(A, A') \in R^a_s$ iff the conditions (1 **must**), (2) and (3) are satisfied, where: (1 **must**) for every $a \in A$ and every $v \in \gamma(a)$ there exist $a' \in A'$ and $v' \in \gamma(a')$ with $(v, v') \in R_s$.

Error states. An abstract state A is an element of E^a iff there exists an $a \in A$ with $a \models \mathit{err}$.

Concretization. The concretization $\gamma^k(f_e)$ of a winning strategy f_e for the environment in G^a is a set of winning environment strategies in the knowledge-based game G^k. For $\pi \in \mathit{Prefs}(G^a)$, we define $\gamma^k(\pi) = \{\pi^k \in \mathit{Prefs}(G^k) \mid |\pi^k| = |\pi|, \forall j : 0 \leq j < |\pi| \Rightarrow \pi^k[j] \subseteq \gamma(\pi[j])\}$ and $\gamma(\pi) = \{\pi^c \in \mathit{Prefs}(G) \mid |\pi^c| = |\pi|, \forall j : 0 \leq j < |\pi| \Rightarrow \pi^c[j] \in \gamma(\pi[j])\}$ (similarly for paths). Then $\gamma^k(f_e)$ is the set of all winning environment strategies f^k_e in G^k such that for every $\pi^k \in \mathit{Prefs}(f^k_e)$ there exists $\pi \in \mathit{Prefs}(f_e)$ with $\pi^k \in \gamma^k(\pi)$. Let $\mathcal{P}$ and $\mathcal{Q}$ be pairs of sets of predicates with $\mathcal{P} \subseteq \mathcal{Q}$. If π and π' are prefixes in $\alpha(\mathcal{G}, \mathcal{P})$ and $\alpha(\mathcal{G}, \mathcal{Q})$, respectively, we write $\pi' \leq \pi$ iff $|\pi| = |\pi'|$ and for every $0 \leq j < |\pi|$, $\gamma(\pi'[j]) \subseteq \gamma(\pi[j])$. If f_e and f'_e are winning strategies for the environment in $\alpha(\mathcal{G}, \mathcal{P})$ and $\alpha(\mathcal{G}, \mathcal{Q})$ respectively, then $f'_e \leq f_e$ iff for every $\pi' \in T(f'_e)$ there exists $\pi \in T(f_e)$ such that $\pi' \leq \pi$.

THEOREM 1. *[Soundness of the abstraction] If f_s is a winning strategy for the system in the perfect-information game $\alpha(\mathcal{G}, \mathcal{P})$, then there exists a consistent winning strategy f^c_s for the system in the symbolic game $\mathcal{G}$ with incomplete information.*

4 Abstract Counterexample Analysis

A winning strategy f_e for the environment in the game $\alpha(\mathcal{G}, \mathcal{P})$ is a *genuine* counterexample if it has a winning concretization in G^k. Otherwise it is called *spurious*. The analysis of the strategy-tree $T(f_e)$ constructs a *strategy-tree formula* $F(f_e)$ that is satisfiable iff f_e is genuine. The key idea is to symbolically simulate a perfect-information game over the equivalence classes of the prefixes of the concrete game structure G with incomplete information.

Traces and error paths. With each node π in $T(f_e)$, we associate a set $\mathit{Traces}(\pi)$ of traces, where a *trace* is a finite sequence $\tau \in C_o^*$ of system outputs, and define $\mathit{Traces}(f_e) = \mathit{Traces}(s_0^a)$. Each trace induces a set of concrete error paths in G. If the strategy f_e is genuine, then for each $\tau \in \mathit{Traces}(f_e)$, the concrete strategy in G^k should provide an error path ξ_τ in G. If π is a leaf node (i.e., an error node), then $\mathit{Traces}(\pi) = \{\epsilon\}$, otherwise, if π is a system node, then $\mathit{Traces}(\pi) = \{c_o\tau \mid c_o \in C_o, \rho \in \mathit{Children}(\pi, T(f_e)), \tau \in \mathit{Traces}(\rho)\}$, and, if π is an environment node, then $\mathit{Traces}(\pi) = \mathit{Traces}(\mathit{Child}(\pi, T(f_e)))$. A path ξ in the concrete game structure G is an *error path of a trace* τ if one of the following three conditions is satisfied: (i) $\xi[0] \models \mathit{err}$, (ii) $\xi[0] \in S_e$ and $\xi[1, |\xi|]$ is an error path for τ or (iii) $\xi[0] \in S_s$, $\tau = c_o\sigma$, $\xi[1]$ is a c_o-successor of $\xi[0]$ and $\xi[1, |\xi|]$ is an error path for σ.

Trace formulas. For each $\tau \in \mathit{Traces}(f_e)$ we define a formula $F(f_e, \tau)$ which is satisfiable iff there is a node $\rho \in T(f_e)$ such that there is an error path for τ in $\gamma(\rho)$. Here, unlike in the perfect-information case, in the concrete strategy the error paths ξ_{τ_1} and ξ_{τ_2} for two different traces $\tau_1, \tau_2 \in \mathit{Traces}(f_e)$ may differ even before the first position in which τ_1 and τ_2 are different, as long as their prefixes up to that position are equivalent. We encode this constraint by indexing the variables in the trace formulas as explained below.

Consider a node π and a trace $\tau \in \mathit{Traces}(f_e)$ such that $\tau = \sigma_1\sigma_2$, σ_1 corresponds to the outputs on the prefix π and $\sigma_2 \in \mathit{Traces}(\pi)$. The variables in $F(f_e, \tau)$ are indexed as follows. The variables that represent a concrete state in $\gamma(\mathit{last}(\pi))$ are indexed with the node π, so that there are different variables in the formula for different nodes. They are indexed also with the part σ_1 of τ, so that there are different variables in different trace formulas after the first difference in the outputs. The unobservable variables have to be indexed additionally with the remaining part σ_2 of τ, in order to have different unobservable variables for corresponding states in different trace formulas even before the first difference in the outputs. To this end, with each node $\pi \in T(f_e)$ and $\sigma_1, \sigma_2 \in C_o^*$ we associate a set $X^{(\pi,\sigma_1,\sigma_2)} = \{x_h^{(\pi,\sigma_1,\sigma_2)}, x_i^{(\pi,\sigma_1)}, x_o^{(\pi,\sigma_1)}, t^{(\pi,\sigma_1)}\}$ of variables and define substitutions which map variables from the original set $X \cup X'$ to variables in the sets $X^{(\pi,\sigma_1,\sigma_2)}$ and vice versa.

We define recursively a *trace formula* $F(\pi, \tau)$ for every node $\pi \in T(f_e)$ and trace $\tau \in \mathit{Traces}(\pi)$. We consider three cases that correspond to the three cases in the definition of error paths: the auxiliary formulas *ErrorState*, *EnvTrans* and *SystTrans* account for cases (i), (ii) and (iii) respectively. If π is a leaf node, then $\tau = \epsilon$ and $F(\pi, \epsilon) = \mathit{ErrorState}(\pi)$. If π is an internal environment node we define $F(\pi, \tau) = \mathit{EnvTrans}(\pi, \pi', \tau)$, where $\pi' = \mathit{Child}(\pi, T(f_e))$. If π is an internal system node, then $\tau = c_o\sigma$ for some c_o and σ and we define $F(\pi, c_o\sigma) = \bigvee_{\pi' \in \mathit{Children}(\pi, T(f_e))} \mathit{SystTrans}(\pi, \pi', c_o\sigma)$. By the definition, the trace formula $F(\pi, \tau)$ is satisfied by a sequence ξ of concrete states iff ξ is an error path for τ and there exists a node ρ in the subtree of $T(f_e)$ below π such that $\xi \in \gamma(\rho)$.

Strategy-tree formula. We define $F(f_e, \tau) = F(s_0^a, \tau)$ for every $\tau \in \mathit{Traces}(f_e)$ and finally, the

strategy-tree formula is $F(f_e) = \bigwedge_{\tau \in Traces(f_e)} F(f_e, \tau)$. It can be constructed by annotating in a bottom-up manner the nodes in $T(f_e)$ with the corresponding sets of traces and formulas.

THEOREM 2. *Let f_e be a winning strategy for the environment in the game $\alpha(\mathcal{G}, \mathcal{P})$. The formula $F(f_e)$ is satisfiable iff the strategy f_e is genuine, i.e., iff $\gamma^k(f_e) \neq \emptyset$.*

5 Counterexample-Guided Refinement

If f_e is a spurious winning strategy for the environment in $\alpha(\mathcal{G}, \mathcal{P})$, we enhance $\mathcal{P} = (\mathcal{P}_{se}, \mathcal{P}_s)$ with sets of refinement predicates $R_{se}(f_e)$ and $R_s(f_e)$, such that in the refined game $\alpha(\mathcal{G}, (\mathcal{P}_{se} \cup R_{se}(f_e), \mathcal{P}_s \cup R_s(f_e)))$ the environment has no winning strategy f'_e with $f'_e \leq f_e$.

5.1 Refining the Abstract Transition Relations

If for some $\tau \in Traces(f_e)$ the formula $F(f_e, \tau)$ is unsatisfiable, then the occurrence of the spurious abstract strategy is due to the approximations of the transition relations. Therefore we compute refinement predicates for eliminating the approximations that cause the existence of f_e. Such predicates can be determined by a bottom-up analysis of the strategy tree $T(f_e)$ that annotates each node π in the tree with a formula $\tilde{F}(\pi, \tau)$ for each trace $\tau \in Traces(\pi)$. The formula $\tilde{F}(\pi, \tau)$ denotes the subset of $\gamma(\pi)$ that consist of those concrete states from which there exists a concrete path that satisfies $F(f_e, \tau)$. We denote with $\texttt{RPGG}(f_e)$ (Refinement Predicates for the Game Graph) the pair $(\texttt{RPGG}_{se}(f_e), \texttt{RPGG}_s(f_e))$ of sets of predicates computed at this step and used to enhance $\mathcal{P}_{se}$ and $\mathcal{P}_s$, respectively.

State formulas. For $\pi \in T(f_e)$ and $\tau \in Traces(\pi)$, we define $\tilde{F}(\pi, \tau)$ as follows. If π is a leaf node, then $\tau = \epsilon$ and $\tilde{F}(\pi, \epsilon) = \bigvee_{a \in last(\pi), a \models err} [a]$. Otherwise, $\tau = c_o \sigma$ and $\tilde{F}(\pi, \tau) = [last(\pi)] \wedge Pre_s(c_o, \bigvee_{\pi' \in Children(\pi, T(f_e))} \tilde{F}(\pi', \sigma))$ if π is a system node, and $\tilde{F}(\pi, \tau) = [last(\pi)] \wedge Pre_e(\tilde{F}(Child(\pi, T(f_e)), \tau))$ otherwise. If $\tau \notin Traces(\pi)$, then $\tilde{F}(\pi, \tau)$ is $\tilde{F}(\pi, \sigma)$, where σ is the maximal prefix of τ such that $\sigma \in Traces(\pi)$ if such exists, and *false* otherwise.

Refinement predicates. The set $\texttt{RPGG}_{se}(f_e)$ of predicates with which we enhance $\mathcal{P}_{se}$ contains all predicates that occur in the annotation formulas $\tilde{F}(\pi, \tau)$. We ensure that the refined abstraction is precise w.r.t. the outputs from some trace $\tau \in Traces(f_e)$ for which $\tilde{F}(f_e, \tau)$ is unsatisfiable, by adding the elements of $OutPreds(\{\tau\})$ for one such τ to $\texttt{RPGG}_{se}(f_e)$, where for a set T of traces, we have defined $OutPreds(T) = \{x_o \approx \tau[j] \mid \tau \in T, 0 \leq j < |\tau|\}$. The set $\texttt{RPGG}_s(f_e)$ is equal to the set $PredsSyst(\mathcal{P}_{se} \cup \texttt{RPGG}_{se}(f_e))$. By refining with these predicates we ensure the monotonicity of the abstraction of the system's transition relation.

5.2 Refining the Abstract Observation Equivalence

If for every $\tau \in Traces(f_e)$ the formula $F(f_e, \tau)$ is satisfiable, the predicates from $\texttt{RPGG}(f_e)$ might not suffice to eliminate the counterexample, because the reason for its existence is the coarseness of the abstract observation-equivalence. We propose an algorithm RPOE (Refinement Predicates for the Observation Equivalence) for computing a set of observable refinement predicates that allow us to distinguish the concrete error paths for different traces. The predicates are obtained from interpolants for unsatisfiable conjunctions of trace formulas.

According to the construction of these formulas, they share only observable variables and hence the computed interpolants contain only observable predicates. The key challenge for the interpolation computation in our case is to ensure that these predicates are *localized*, i.e., that the variables which occur in an atom correspond to a single concrete state and not to a sequence of concrete states. We extend the algorithm from [10], which reduces the computation of interpolants for linear arithmetic to linear programming problems, in order to handle this additional condition on the variable occurrences. Our more general algorithm LILA (Linear Interpolation with Localized Atoms) receives in addition a partitioning of the variables which occur in the input systems of inequalities and as a result, each atom in the generated interpolant is guaranteed to contain variables from exactly one partition. We first present the algorithm RPOE and then describe the procedure LILA.

Algorithm:RPOE

Input: symbolic game $\mathcal{G} = (\mathcal{S}, err)$, pair $\mathcal{P} = (\mathcal{P}_{se}, \mathcal{P}_s)$ of finite sets of predicates,
strategy tree $T(f_e)$ of an abstract winning environment strategy in $\alpha(\mathcal{G}, \mathcal{P})$
Output: pair of sets of refinement predicates (R_{se}, R_s)
$\Phi := \{F(f_e, \tau) \mid \tau \in \mathit{Traces}(f_e)\}$; $R_{se} := \emptyset$;
while *all elements of* Φ *are satisfiable* **do**
  pick $\Psi \subseteq \Phi$, $\varphi \in \Phi \setminus \Psi$ **such that** $\psi := \bigwedge_{\phi \in \Psi} \phi$ **is satisfiable and** $\varphi \wedge \psi$ **is unsatisfiable;**
  $n := \mathit{max}(\mathit{MaxIx}(\varphi), \mathit{MaxIx}(\psi))$**;**
  if $R_{se} = \emptyset$ **then** $R_{se} := \mathit{OutPreds}(\{\tau \mid F(f_e, \tau) \in \{\varphi\} \cup \Psi\})$**;**
  $\theta := \mathtt{LILA}(\varphi, \psi, (\mathit{Vars}^0(\varphi) \cup \mathit{Vars}^0(\psi), \ldots, \mathit{Vars}^n(\varphi) \cup \mathit{Vars}^n(\psi)))$**;**
  $R_{se} := R_{se} \cup (\mathit{Preds}(\theta))\ \mathit{subst}_X$**;** $\Phi := \{\theta \wedge \phi \mid \phi \in \Psi\}$**;**
return $(R_{se}, \mathit{PredsSyst}(\mathcal{P}_{se} \cup R_{se}))$;

Distinguishing abstract prefixes. Let $\Phi = \{F(f_e, \tau) \mid \tau \in \mathit{Traces}(f_e)\}$. As all formulas $F(f_e, \tau)$ are satisfiable and the formula $F(f_e)$ is not, there exists a subset Ψ of Φ such that $\psi = \bigwedge_{\phi \in \Psi} \phi$ is satisfiable and there exists a formula $\varphi \in \Phi \setminus \Psi$ such that $\varphi \wedge \psi$ is unsatisfiable.

The variables in $\bigcup_{\pi \in T(f_e), \sigma_1, \sigma_2 \in C_o^*} X^{(\pi, \sigma_1, \sigma_2)}$ (and hence the variables in φ and in ψ) are partitioned according to the length of π: For $j \in \mathbb{N}$, X^j is the union of all sets $X^{(\pi, \sigma_1, \sigma_2)}$ with $|\pi| = j$. For a formula ϕ, $\mathit{MaxIx}(\phi)$ is the maximal j with $\mathit{Vars}^j(\phi) = \mathit{Vars}(\phi) \cap X^j \neq \emptyset$.

When φ and ψ are (disjunctions of) mixed systems of linear inequalities, we apply algorithm LILA described in the next paragraph to compute an interpolant θ such that each literal which occurs in θ is of the form $ix \lhd \delta$ where $\lhd \in \{\leq, <\}$ and the only variables which occur in such an inequality are in the set $\{x_i^{(\pi,\sigma)}, x_o^{(\pi,\sigma)}, t^{(\pi,\sigma)}\}$ for some $\pi \in T(f_e)$ and $\sigma \in C_o^*$, i.e. the coefficients in front of all other variables are 0. By applying the substitution subst_X to the atoms in θ, we obtain a set of predicates over observable variables from the original set of variables X. Then, the set Φ is updated to be the set of conjunctions $\theta \wedge \phi$, where $\phi \in \Psi$ and the process is repeated while all elements of the current set Φ are satisfiable. The predicates in $\mathtt{RPOE}_{se}(f_e)$ are the atoms from all computed interpolants, plus the set of output predicates for the traces corresponding to the formulas in the initial set $\Psi \cup \{\varphi\}$. The predicates in $\mathtt{RPOE}_s(f_e)$ ensure the monotonicity of the abstraction.

Computing interpolants with localized atoms. We now present the algorithm LILA for computing localized interpolants. A mixed system, denoted $Ax \leqslant a$, consists of strict and non-strict linear inequalities. The input of algorithm LI from [10] consists of two mixed

systems of inequalities $Ax \leqslant a$ and $Bx \leqslant b$ such that the conjunction $Ax \leqslant a \wedge Bx \leqslant b$ is not satisfiable. The output is a linear interpolant $ix \lhd \delta$ where $\lhd \in \{\leq, <\}$. Algorithm LILA receives in addition a partitioning $(V^0, V^1, \ldots, V^n)$ of the variables in the vector x. The output is an interpolant for $Ax \leqslant a$ and $Bx \leqslant b$ which is of the form $\bigwedge_{j=0}^{n} i_j x \lhd_j \delta_j$, where $\lhd_j \in \{\leq, <\}$ and for each $0 \leq j \leq n$, only variables from V^j occur in $i_j x \lhd_j \delta_j$. If such interpolant is not found, the element $\bot$ is returned. The variables $\lambda, \lambda_0, \lambda_1, \ldots, \lambda_n$ denote vectors which define linear combinations of inequalities in $Ax \leqslant a$. The subvectors $\lambda^{\mathsf{lt}}, \lambda^{\mathsf{le}}, \lambda_j^{\mathsf{lt}}, \lambda_j^{\mathsf{le}}$ for $j = 0, 1, \ldots, n$ define linear combinations of strict and non-strict inequalities in $Ax \leqslant a$, respectively. Similarly for $\mu, \mu^{\mathsf{lt}}, \mu^{\mathsf{le}}$. For each $0 \leq j \leq n$, the set of variables V^j defines a set $Ix(j)$ of indices: $Ix(j) = \{k \mid k \in \{1, \ldots, m_A\}, x_k \in V^j\}$, where m_A is the number of columns in A. Its complement $\{1, \ldots, m_A\} \setminus Ix(j)$ is denoted with $\overline{Ix(j)}$. For $1 \leq k \leq m_A$, the k-th column of the matrix A is denoted with $A_{|k}$. For disjunctions of mixed systems, i.e., for formulas $\bigvee_k A_k x \leqslant a_k$ and $\bigvee_l B_l x \leqslant b_l$ in DNF, we proceed as in [10]: compute an interpolant θ_{kl} for each pair of disjuncts and then take $\bigvee_k \bigwedge_l \theta_{kl}$.

THEOREM 3. *Algorithm* LILA *is* **sound***: If it returns a conjunction $\theta = \bigwedge_{j=0}^{n} \theta_j$, then θ is an interpolant for the pair of mixed systems $Ax \leqslant a$ and $Bx \leqslant b$ with the following properties: (1) for each j, θ_j is of the form $i_j x \lhd_j \delta_j$ where $\lhd_j \in \{\leq, <\}$; (2) there exist row vectors $\lambda_0, \ldots, \lambda_n$ such that for every $0 \leq j \leq n$, $\lambda_j \geq 0$, $i_j = \lambda_j A$ and $\delta_j = \lambda_j a$; (3) for each $0 \leq j \leq n$, only variables from V^j occur in θ_j. Algorithm* LILA *is* **complete***: if an interpolant $\theta = \bigwedge_{j=0}^{n} \theta_j$ with the properties (1),(2) and (3) exists, then the algorithm will find one.*

Algorithm:LILA
Input: $Ax \leqslant a$ and $Bx \leqslant b$: mixed systems, $Ax \leqslant a \wedge Bx \leqslant b$ is unsatisfiable,
partitioning $(V^0, V^1, \ldots, V^n)$ of the variables in x
Output: interpolant $\bigwedge_{j=0}^{n} i_j x \lhd_j \delta_j$ where $\lhd_j \in \{\leq, <\}$ and
only variables from V^j occur in $i_j x \lhd_j \delta_j$
$\chi_1 := \lambda \geq 0 \wedge \mu \geq 0 \wedge \lambda A + \mu B = 0$;
$\chi_2 := \lambda = \sum_{j=0}^{n} \lambda_j \wedge \bigwedge_{j=0}^{n} (\lambda_j \geq 0 \wedge i_j = \lambda_j A \wedge \delta_j = \lambda_j a \wedge \bigwedge_{k \in \overline{Ix(j)}} \lambda_j A_{|k} = 0)$;
if exist $\lambda, \mu, \lambda_j, i_j, \delta_j$, for $0 \leq j \leq n$ **satisfying** $\chi_1 \wedge \chi_2 \wedge \lambda a + \mu b \leq -1$
then return $\bigwedge_{j=0}^{n} i_j x \leq \delta_j$;
elif exist $\lambda, \mu, \lambda_j, i_j, \delta_j$, for $0 \leq j \leq n$ **satisfying** $\chi_1 \wedge \chi_2 \wedge \lambda a + \mu b \leq 0 \wedge \lambda^{\mathsf{lt}} \neq 0$
then return $\bigwedge_{0 \leq j \leq n, \lambda_j^{\mathsf{lt}} \neq 0} i_j x < \delta_j \wedge \bigwedge_{0 \leq j \leq n, \lambda_j^{\mathsf{lt}} = 0} i_j x \leq \delta_j$;
elif exist $\lambda, \mu, \lambda_j, i_j, \delta_j$, for $0 \leq j \leq n$ **satisfying** $\chi_1 \wedge \chi_2 \wedge \lambda a + \mu b \leq 0 \wedge \mu^{\mathsf{lt}} \neq 0$
then return $\bigwedge_{j=0}^{n} i_j x \leq \delta_j$;
else return $\bot$

5.3 Refinement Loop

In each iteration of the refinement loop, an abstract perfect-information game is solved. If it is won by the system player, the algorithm terminates returning an abstract winning strategy for the system. Otherwise, the abstraction is refined with the predicates $R(f_e)$, computed for some abstract winning strategy f_e for the environment. There are two cases. If refining the transition relations suffices to eliminate f_e, the abstraction is refined with the predicates in RPGG(f_e). Otherwise, the predicates in RPOE(f_e) are used for refinement. In

the second case, it is possible that in the game $\alpha(\mathcal{G}, \mathcal{P} \cup \texttt{RPOE}(f_e))$, the environment has a winning strategy f'_e with $f'_e \leq f_e$. Then, we also refine with the predicates in $\texttt{RPGG}(f'_e)$ for every such f'_e. The set $\mathit{Refine}(f_e, \mathcal{P}')$ consists of all winning strategies for the environment in $\alpha(\mathcal{G}, \mathcal{P}')$ subsumed by f_e. It can be computed from the strategy f_e and the predicates in $\mathcal{P}'$.

Algorithm:ARGII
Input: symbolic safety game $\mathcal{G} = (\mathcal{S}, \mathit{err})$ **Output:** pair $(\mathit{winner}, \mathit{abstract\ strategy})$
$\mathcal{P} := (\mathcal{P}_{se}, \mathcal{P}_s)$, **where** $\mathcal{P}_{se} := \mathit{Preds}(\mathit{init}) \cup \mathit{Preds}(\mathit{err}) \cup \{t \approx 0\}$ **and** $\mathcal{P}_s := \mathit{PredsSyst}(\mathcal{P}_{se})$;
solve $\alpha(\mathcal{G}, \mathcal{P})$ **and determine:** *winner* **and** *strategy*;
while *winner* = *env* **do**
  if $F(\mathit{strategy})$ **is satisfiable then return** $(\mathit{winner}, \mathit{strategy})$;
  $f_e := \mathit{strategy}$;
  if $\exists \tau \in \mathit{Traces}(f_e) : F(f_e, \tau)$ *is unsatisfiable* **then compute** $R := \texttt{RPGG}(f_e)$;
  else
    $R := \texttt{RPOE}(f_e)$; **compute** $S := \mathit{Refine}(f_e, R)$;
    forall $f'_e \in S$ **do** $R := R \cup \texttt{RPGG}(f'_e)$;
  $\mathcal{P} := \mathcal{P} \cup R$; **solve** $\alpha(\mathcal{G}, \mathcal{P})$ **and determine** *winner* **and** *strategy*;
return $(\mathit{winner}, \mathit{strategy})$

THEOREM 4. *[Soundness of algorithm* ARGII*] The algorithm* ARGII *is sound: if it returns* (sys, f^a_s)*, then the concrete symbolic game* $(\mathcal{S}, \mathit{err})$ *is won by the system and* f^a_s *is a concretizable abstract winning strategy for the system; if it returns* (env, f^a_e) *then* $(\mathcal{S}, \mathit{err})$ *is won by the environment and* f^a_e *is a concretizable abstract winning strategy for the environment.*

THEOREM 5. *[Progress property of the refinement] Let* f_e *be a spurious wining strategy for the environment in the game* $\alpha((\mathcal{S}, \mathit{err}), \mathcal{P})$*. In* $\alpha((\mathcal{S}, \mathit{err}), \mathcal{P} \cup R(f_e))$*, the environment does not have a winning strategy* f'_e *with* $f'_e \leq f_e$*.*

6 Termination of the Abstraction Refinement Loop

In this section we provide sufficient conditions for termination of the refinement loop. In order to guarantee that only finitely many different abstract states are generated during the execution of the algorithm, we make standard assumptions about the concrete game graph, which we extend with conditions related to the presence of incomplete information. As we also have to account for the refinement predicates obtained from interpolants, we apply the standard technique (e.g, [8]) of restricting the interpolants computed at each step to some finite language $\mathcal{L}_b$ and maintaining completeness by gradually enlarging the restriction language when this is needed. We make use of the fact that our algorithm reduces interpolant computation to constraint solving, in order to achieve the restriction of the language by imposing additional constraints on the generated inequalities.

Computing restricted linear interpolants. We restrict the language of the computed interpolants to the set of rectangular predicates over the variables in $\mathit{Obs}(X)$. A *rectangular predicate* over $\mathit{Obs}(X)$ is a conjunction of *rectangular inequalities* of the form $ax \lhd c$, where $x \in \mathit{Obs}(X)$, $a \in \{-1, 1\}$, $\lhd \in \{<, \leq\}$ and c is an integer constant. For $m \in \mathbb{N}$, a rectangular predicate φ is called *m-bounded* if for each conjunct $ax \lhd c$ of φ, $|c| \leq m$. Let $\mathcal{L}_m$ be the set of all m-bounded rectangular predicates over $\mathit{Obs}(X)$. The modified algorithm $\texttt{LILA}_\texttt{r}$ gets

as input also a bound $b \in \mathbb{N}$ and ensures that every conjunct in the computed interpolant is in $\mathcal{L}_b$. If such an interpolant does not exist, then the bound b is increased. The modified algorithm partitions the variables into singleton sets and uses in conjunction with χ_1 and χ_2 the additional constraints: (1) χ_3 defined as $\chi_3 = \bigwedge_{j=0}^{n}(i_j \leq 1 \wedge i_j \geq -1 \wedge \delta_j \leq b \wedge \delta_j \geq -b)$ and (2) the variables δ_j and the variables in the vector i_j assume integer values.

Region algebra for an incomplete-information game. A *region algebra for a symbolic safety game* $(\mathcal{S}, err)$ is a pair (R, Obs) of possibly infinite sets $R \subseteq 2^S$ and $Obs \subseteq R$ of regions with the following properties: (1) for every $r_1, r_2 \in R$, we have $r_1 \cup r_2, r_1 \cap r_2, S \setminus r_1 \in R$; (2) for every $r_1, r_2 \in Obs$, we have $r_1 \cup r_2, r_1 \cap r_2, S \setminus r_1 \in Obs$; (3) the sets $\{v \in S \mid v \models t \approx 0\}$ and $\{v \in S \mid v \models t \approx 1\}$ are in R; (4) for every $r \in R$ and $c_o \in C_o$, and every $p \in Preds(Pre_e(r)) \cup Preds(Pre_s(c_o, r))$ it holds that for every $r' \in R$, either for every $v \in r'$, $v \models p$ or for every $v \in r'$, $v \models \neg p$; (5) for every $c_o \in C_o$, the set $\{v \in S \mid v \models x_o \approx c_o\}$ is in Obs; (6) for every $\pi_1, \pi_2 \in Prefs(G)$, if each of $last(\pi_1)$ and $last(\pi_2)$ is an error state and there exists an index j such that $\pi_1[j]$ and $\pi_2[j]$ are system states and $\pi_1[j] \not\equiv \pi_2[j]$, then there exist $0 \leq k \leq j$ and $r \in Obs$ such that $\pi_1[k] \in r$ and $\pi_2[k] \notin r$.

THEOREM 6. *[Termination] Consider a symbolic safety game $(\mathcal{S}, err)$ for which there exists a finite region algebra (R, Obs) with $Obs = \mathcal{L}_m$ for some $m \in \mathbb{N}$. If algorithm* `ARGII` *using the modified algorithm* $\texttt{LILA}_\texttt{r}$ *is called with argument $(\mathcal{S}, err)$, then it terminates.*

References

[1] T. Ball, B. Cook, S. Das, and S. K. Rajamani. Refining approximations in software predicate abstraction. In *TACAS*, volume 2988 of *LNCS*, pages 388–403. Springer, 2004.

[2] K. Chatterjee, L. Doyen, T. A. Henzinger, and J.-F. Raskin. Algorithms for omega-regular games of incomplete information. In *Proc. CSL*, volume 4207 of *LNCS*. 2006.

[3] S. Das and D. L. Dill. Counter-example based predicate discovery in predicate abstraction. In *Proc. FMCAD*, pages 19–32, London, UK, 2002. Springer-Verlag.

[4] L. de Alfaro and P. Roy. Solving games via three-valued abstraction refinement. In *Proc. CONCUR*, volume 4703, pages 74–89. Springer-Verlag, 2007.

[5] M. De Wulf, L. Doyen, and J.-F. Raskin. A lattice theory for solving games of imperfect information. In *Proc. HSCC*, LNCS, pages 153–168. Springer-Verlag, 2006.

[6] R. Dimitrova and B. Finkbeiner. Abstraction refinement for games with incomplete information. Reports of SFB/TR 14 AVACS 43, SFB/TR 14 AVACS, October 2008.

[7] T. Henzinger, R. Jhala, and R. Majumdar. Counterexample-guided control. In *Proc. ICALP'03*, volume 2719 of *LNCS*, pages 886–902. Springer-Verlag, 2003.

[8] R. Jhala and K. L. McMillan. A practical and complete approach to predicate refinement. In *Proc. TACAS*, volume 3920, pages 459–473. Springer-Verlag, 2006.

[9] J. H. Reif. The complexity of two-player games of incomplete information. *J. Comput. Syst. Sci.*, 29(2):274–301, 1984.

[10] A. Rybalchenko and V. Sofronie-Stokkermans. Constraint solving for interpolation. In *Proc. VMCAI*, volume 4349 of *LNCS*, pages 346–362. Springer-Verlag, 2007.

Foundations of Software Technology and Theoretical Computer Science (Bangalore) 2008.
Editors: R. Hariharan, M. Mukund, V. Vinay; pp 187-198

A new approach to the planted clique problem

Alan Frieze[1]*,**Ravi Kannan**[2]

[1] Department of Mathematical Sciences,
Carnegie Mellon University.
Pittsburgh PA15213
USA
alan@random.math.cmu.edu

[2] Microsoft Research Laboratories,
India
kannan100@gmail.com

1 Introduction

It is well known that finding the largest clique in a graph is NP-hard, [8]. Indeed, Hastad [5] has shown that it is NP-hard to approximate the size of the largest clique in an n vertex graph to within a factor $n^{1-\epsilon}$ for any $\epsilon > 0$. Not surprisingly, this has directed some researchers attention to finding the largest clique in a random graph. Let $G_{n,1/2}$ be the random graph with vertex set $[n]$ in which each possible edge is included/excluded independently with probability $1/2$. It is known that **whp** the size of the largest clique is $(2 + o(1)) \log_2 n$, but no known polymomial time algorithm has been proven to find a clique of size more than $(1 + o(1)) \log_2 n$. Karp [9] has even suggested that finding a clique of size $(1 + \epsilon) \log_2 n$ is computationally difficult for any constant $\epsilon > 0$.

Significant attention has also been directed to the problem of finding a *hidden* clique, but with only limited success. Thus let G be the union of $G_{n,1/2}$ and an unknown clique on vertex set P, where $p = |P|$ is given. The problem is to recover P. If $p \geq c(n \log n)^{1/2}$ then, as observed by Kucera [10], with high probability, it is easy to recover P as the p vertices of largest degree. Alon, Krivelevich and Sudakov [1], using spectral analysis, were able to improve this to $p = \Omega(n^{1/2})$. McSherry [11] gives some refinements of this method. In conjunction with a negative result of Jerrum [6] that one possible Markov chain approach fails for $p = o(n^{1/2})$, $p = \Omega(n^{1/2})$ seems like a natural barrier for solving this problem. Feige and Krauthgamer [4] considered finding a planted clique in the context of the semi-random model. Juels and Peinado [7] considered the application of this problem to Cryptographic Security.

Let A_G denote the adjacency matrix of G. The spectral approach of [1] essentially maximizes $x^T A_G x$ over vectors x with $|x| = 1$, expecting that the optimal solution is close to u, defined by $u_i = p^{-1/2} 1_{i \in P}$, ($u$ is the scaled characteristic vector of P) so that we may recover P from the optimal solution.

*Supported in part by NSF grant ccr0200945

FSTTCS 2008
IARCS Annual Conference on
Foundations of Software Technology and Theoretical Computer Science
http://drops.dagstuhl.de/opus/volltexte/2008/1752

In this paper, we define a natural 3-dimensional array A related to the given graph : $A_{i,j,k}$ will be ± 1 depending on whether the parity of the number of edges among the vertices i, j, k is odd or even respectively. Our main result here (Section 2) shows that as long as $p = \Omega(n^{1/3}(\log n)^4)$, the maximum of the cubic form or *tensor* $A(x,x,x) = \sum_{i,j,k} A_{i,j,k}x_ix_jx_k$, $x \in B_n = \{x \in R^n : |x| = 1\}$ is attained close to u. Thus if we can find this maximimum, then we can recover the clique. However, unlike the case of the quadratic form, where the maximization is an eigenvalue computation which is well-known to be solvable in polynomial time, there are in general no known polynomial time algorithms for maxmizing cubic forms. So, our existential result does not automatically lead to an algorithm and this is left as an open question. We make the following conjecture which would yield an algorithm if proved.

Conjecture *Suppose that an $n \times n \times n$ array A is constructed as above from $G_{n,1/2}$ plus a planted clique of size $p \in \Omega(n^{1/3}(\log n)^c)$. Then the function $A(x,x,x)$ has a unique local maximum as x varies over B_n.*

2 The cubic form and the main result

We define the 3-dimensional array :

$$A_{i,j,k} = \begin{cases} 1 & \text{if } i,j,k \text{ are distinct and } G \text{ contains 1 or 3 edges of the triangle } i,j,k. \\ -1 & \text{if } i,j,k \text{ are distinct and } G \text{ contains 0 or 2 edges of the triangle } i,j,k. \\ 0 & \text{if } i,j,k \text{ are not distinct.} \end{cases}$$

We assume that

$$p = C_1 n^{1/3}(\log n)^4.$$

Here $C_1, C_2, \ldots,$ are unspecified positive absolute constants.

For vectors x, y, z, we define

$$A(x,y,z) = \sum_{i,j,k} A_{i,j,k}x_iy_jz_k.$$

x, y, z will denote vectors of length 1 throughout. We will reserve u for the scaled characteristic vector of P defined earlier. The following Theorem (which is the Main Theorem of the paper) will imply (see Corollary 2 below) that if at least one of x, y, z is orthogonal to u, then we have $|A(x,y,z)| \le C_2 n^{1/2}(\log n)^4$. In which case,

$$A(u,u,u) = \frac{p(p-1)(p-2)}{p^{3/2}} \sim p^{3/2} = \omega(A(x,y,z))$$

for all such x, y, z. (We use the notation $a_n = \omega(b_n)$ to mean that $a_n/b_n \to \infty$ as $n \to \infty$).

Let

$$P^{3*} = \{(i,j,k) \in P^3 : i,j,k \text{ are distinct}\}$$

Define the 3-dimensional matrix D by

$$D_{i,j,k} = \begin{cases} 1 & (i,j,k) \in P^{3*}, \\ 0 & \text{otherwise} \end{cases}$$

and let $B = A - D$.

$$B(x,y,z) = A(x,y,z) - \sum_{i,j,k \in P^{3*}} x_i y_j z_k. \tag{1}$$

The entries of A in $P \times P \times P$ contribute $\sum_{(i,j,k)\in P^{3*}} x_i y_j z_k$ to the tensor $A(x,y,z)$; so $B(x,y,z)$ is the contribution due to the random graph alone. The proof of Theorem 1 occupies all of Section 3. We defer the proofs of the corollaries following it to Section 4.

THEOREM 1. *There exists C_3 such that*

$$\mathbf{Pr}\left(\exists x,y,z:\ |B(x,y,z)| \geq C_3 n^{1/2}(\log n)^4\right) = o(1).$$

Let

$$U^* = \{(x,y,z):\ x.u = 0 \text{ or } y.u = 0 \text{ or } z.u = 0\}.$$

COROLLARY 2. *If $(x,y,z) \in U^*$ then*

$$|A(x,y,z)| \leq 2C_3 n^{1/2}(\log n)^4. \tag{2}$$

So, **whp** , we have that

$$A(u,u,u) = \omega\left(\max_{(x,y,z)\in U^*} A(x,y,z)\right). \tag{3}$$

COROLLARY 3. *Suppose the maximum of the multilinear form $A(x,y,z)$ as x,y,z vary over the unit ball is attained at x^*,y^*,z^*. Then,* $\min\{x^*\cdot u, y^*\cdot u, z^*\cdot u\} = 1 - o(1)$.

The above corollary ensures that from x^*,y^*,z^*, we can find the clique P using the Theorem below. (See Section 4.)

THEOREM 4. *There is a polynomial time algorithm which given as input a unit vector v, returns a set P' of cardinality p satisfying the following: If $v\cdot u \geq \frac{C_4 \log n}{p^{1/2}}$, for sufficiently large C_4 then $P' = P$.*

Observe that it is trivial to get a vector v satisfying $v\cdot u \geq 1/p^{1/2}$ by trying out all n unit vectors. Getting a vector v satisfying the hypothesis of the Theorem in polynomial time, however, seems to be non-trivial.

Remarks: We can assume that $x^* = y^* = z^*$ in Corollary 3. Indeed, for a fixed x, the problem of maximising $A(x,y,z)$ over the unit ball B_n amounts to maximizing $y^T A_x z$ for $y,z \in B_n$. Here A_x is the $n\times n$ matrix defined by $A_x(i,j) = \sum_k A_{i,j,k}x_k$. A_x is a symmetric matrix and so for each x there is a maximum in which $y = z$. Now define a sequence of vector triples $x_k,y_k,z_k, k = 0,1,2,\ldots$, where $x_0,y_0,z_0 = x^*,y^*,z^*$ and $x_1 = x_0$ and $y_1 = z_1$ maximise $y^T A_{x_1} z$ over B_n. Now to obtain $x_2,y_2 = y_1,z_2$ we find $x = z$ to maximise $A(x,y_1,z$ and so on. Any limit point of this sequence $\hat{x},\hat{y},\hat{z}$ must maximise $A(x,y,z)$ and must have $\hat{x} = \hat{y} = \hat{z}$. If for example, $\hat{x} \neq \hat{y}$ then we have the contradiction that there are points of the form ξ,ξ,η arbitrarily close $\hat{x},\hat{y},\hat{z}$.

Remarks: By switching from 2-dimensional matrices to 3-dimensional matrices we have reduced the necessary size of P from $\tilde{O}(n^{1/2})$ to $\tilde{O}(n^{1/3})$. An interesting open question is whether using the natural k-dimensional matrices (whose entries are ± 1 depending on the parity of the number of edges of G in the induced sub-graphs on k vertices) will allow us to go down to $\tilde{O}(n^{1/k})$, for any fixed positive integer k.

Remarks: We note that x^* is a local maximum of the function $A(x,x,x)$ (with respect to first and second order moves) over the unit ball iff

1. x^* is the eigenvector corresponding to the highest eigenvalue of the matrix $A(x^*)$ and
2. the second highest eigenvalue of $A(x^*)$ is at most half the highest.

We can assume that $|x| = 1$. Let $F(x) = A(x,x,x)$ and let h be small and let $x \cdot h = 0$. Then we write $F\left(\frac{x+h}{|x+h|}\right) \leq F(x)$ as

$$F(x) + 3A(x,x,h) + 3A(x,h,h) + O(|h|^3) \leq F(x)(1 + 3|h|^2/2 + O(|h|^4).$$

Then we will need $x \cdot h = 0$ implies $A(x,x,h) = 0$ and $\max_h A(x,h,h) = \lambda_2(A_x)|h|^2$.)

3 Proof of Theorem 1

We will have to make a series of technical modfications. These modifications reduce proving Theorem 1 to Lemma 6 below. In the next Section 3.1, we carry out the central part, namely the proof of Lemma 6.

The first modification is that it is easy to see that if we set to zero all the x_i for which $|x_i| \leq 1/n^2$, as well as similarly for y, z, then the RHS of (1) changes by at most 1. So we will assume that either $x_i = 0$ or $|x_i| \geq 1/n^2$, and similarly for y, z.

Now, here is our second technical modification: Let V_1, V_2, V_3 form an arbitrary partition of V into three subsets, each of size $m = n/3$. Noting that by symmetry, each triangle i, j, k appears in the same number of $V_1 \times V_2 \times V_3$, one can see that

$$\sum_{(i,j,k)} B_{i,j,k} x_i y_j z_k \leq \frac{27}{\binom{n}{m,m,m}} \sum_{V_1,V_2,V_3} \sum_{(i,j,k) \in V_1 \times V_2 \times V_3} B_{i,j,k} x_i y_j z_k$$

So,

$$\left| \sum_{(i,j,k)} B_{i,j,k} x_i y_j z_k \right| \leq \frac{27}{\binom{n}{m,m,m}} \sum_{V_1,V_2,V_3} \left| \sum_{(i,j,k) \in V_1 \times V_2 \times V_3} B_{i,j,k} x_i y_j z_k \right| \tag{4}$$

Now for any x, y, z we have

$$|\sum_{(i,j,k) \in V_1 \times V_2 \times V_3} B_{i,j,k} x_i y_j z_k| \leq (\sum_i |x_i|)(\sum_j |y_j|)(\sum_k |z_k|) \leq n^{3/2}. \tag{5}$$

We will prove below that for each **fixed** partition of V into three equal sized subsets - V_1, V_2, V_3, we have,

$$\mathbf{Pr}\left(\max_{x,y,z} \left| \sum_{(i,j,k) \in V_1 \times V_2 \times V_3} B_{i,j,k} x_i y_j z_k \right| \geq C_5 n^{1/2} (\log n)^4 \right) \leq \frac{1}{n^6}. \tag{6}$$

One can derive Theorem 1 from (4), (5) and (6) by the following simple argument: Say that a partition V_1, V_2, V_3 is bad for A, if $\max_{x,y,z}\left|\sum_{(i,j,k)\in V_1\times V_2\times V_3} B_{i,j,k}x_iy_jz_k\right| \geq C_5n^{1/2}(\log n)^4$ and we let $\mathcal{P}_B$ denote the set of bad partitions. Let

$$g(A) = \frac{|\mathcal{P}_B|}{\binom{n}{m,m,m}}.$$

Then, we know that $\mathbf{E}_A(g(A)) \leq 1/n^6$ from which it follows by Markov inequality that

$$\mathbf{Pr}_A\left(g(A) \geq \frac{100}{n^4}\right) \leq \frac{1}{100n^2}.$$

For any A with $g(A) \leq 100/n^4$, we have from (5)

$$\sum_{V_1,V_2,V_3}\max_{x,y,z}\left|\sum_{(i,j,k)\in V_1\times V_2\times V_3} B_{i,j,k}x_iy_jz_k\right| \leq \left(C_5n^{1/2}(\log n)^4 + \frac{100}{n^4}n^{3/2}\right)\binom{n}{m,m,m}$$

and Theorem 1 follows.

To prove (6), we fix attention from now on on one particular V_1, V_2, V_3. We let

$$X(x,y,z) = \sum_{(i,j,k)\in V_1\times V_2\times V_3} B_{i,j,k}x_iy_jz_k$$

and

$$(x^*,y^*,z^*) = \text{argmax}_{x,y,z}|X(x,y,z)|$$

and suppose that

$$|X(x^*,y^*,z^*)| \geq C_5n^{1/2}(\log n)^4. \tag{7}$$

For sets $R \subseteq V_1, S \subseteq V_2, T \subseteq V_3$ of vertices, we let $\mathbf{B}(R,S,T)$ be the set of triples of vectors (x,y,z) satisfying

$$\begin{aligned}
&|x|,|y|,|z| \leq 1.\\
&R = \{i : x_i \neq 0\}, \quad S = \{j : y_j \neq 0\}, \quad T = \{k : z_k \neq 0\}.\\
&|x_i/x_j| \leq 2,\ \forall i,j \in R,\ |y_i/y_j| \leq 2,\ \forall i,j \in S,\ |z_i/z_j| \leq 2,\ \forall i,j \in T.
\end{aligned}$$

Note that this implies

$$|x_i| \leq \frac{2}{|R|^{1/2}},\ |y_i| \leq \frac{2}{|S|^{1/2}},\ |z_i| \leq \frac{2}{|T|^{1/2}},\ \forall i. \tag{8}$$

Since $\frac{1}{n^2} \leq |x_i^*|, |y_j^*|, |z_k^*| \leq 1$, we can write each of x^*, y^*, z^* as the sum of $\log_2(n^2)$ vectors, each of which has the property that its non-zero components are within a factor of 2 of each other. Thus, (7) implies that there exist R, S, T such that

$$\max_{(x,y,z)\in\mathbf{B}(R,S,T)}|X(x,y,z)| \geq C_6n^{1/2}\log n.$$

So, we see that (7) would lead to the non-occurrence of the event $\mathcal{A}$ in the following Lemma.

LEMMA 5. *For every fixed partition of V into three equal sized sets V_1, V_2, V_3, we have that with probability at least $1 - \frac{1}{n^6}$, the following event $\mathcal{A}$ holds:*

$\mathcal{A}$: For all R, S, T, $R \subseteq V_1, S \subseteq V_2, T \subseteq V_3$,

$$\max_{(x,y,z)\in \mathbf{B}(R,S,T)} |X(x,y,z)| < C_6 n^{1/2} \log n.$$

This in turn will follow from the next lemma:

LEMMA 6. *Suppose R, S, T are fixed pair-wise disjoint subsets of vertices, with $|R| = r, |S| = s, |T| = t$. Then with probability at least $1 - n^{-6(r+s+t)}$, the following event which we will call $\mathcal{A}_{R,S,T}$ happens:*

$$\max_{(x,y,z)\in \mathbf{B}(R,S,T)} |X(x,y,z)| \geq C_6 n^{1/2} \log n.$$

Lemma 5 follows from Lemma 6 by the following argument: For each set of integers r, s, t, the number of subsets (R, S, T) of $\{1, 2, \ldots n\}$ with $|R| = r, |S| = s, |T| = t$ is at most n^{r+s+t}. Thus we will concentrate on proving Lemma 6.

3.1 Proof of Lemma 6

Note that R can be partitioned into two parts - $R \cap P$ and $R \setminus P$, similarly also S, T. So, it suffices to prove that for any fixed R, S, T, each either contained in P or disjoint from P, the following event $\mathcal{B}_{R,S,T}$ happens with probability at least $1 - n^{-6(r+s+t)}$:

$$\mathcal{B}_{R,S,T} : \max_{x,y,z\in \mathbf{B}(R,S,T)} |X(x,y,z)| \leq C_7 n^{1/2} \log n.$$

If $R, S, T \subseteq P$, then $X(x, y, z) = 0$. So, we may assume in what follows that

$$(R \subseteq P \text{ or } R \cap P = \emptyset),\ (S \subseteq P \text{ or } S \cap P = \emptyset),\ (T \subseteq P \text{ or } T \cap P = \emptyset),\ (R \cup S \cup T \not\subseteq P)$$

We consider the following cases, which up to re-naming of R, S, T are exhaustive:

Case 1: $S, T \subseteq P$ and $R \cap P = \emptyset$ and $|R| \leq \max\{|S|, |T|\} \leq |P|$.

In this case we use the Azuma-Hoeffding martingale tail inequality, see for example [3]. We have $\mathbf{E}(X) = 0$ and $X = X(x, y, z)$ is determined by $r(s + t)$ independent random variables (the edges in $R \times (S \cup T)$). Now adding or removing an edge in $R \times S$ (resp. $R \times T$) can change X by at most $\frac{8t}{(rst)^{1/2}}$ (resp. $\frac{8s}{(rst)^{1/2}}$) (recall (8)). Applying the inequality we see that

$$\mathbf{Pr}(|X| \geq C_6 n^{1/2} \log n) \leq 2 \exp\left\{-\frac{C_7 n (\log n)^2}{s+t}\right\} \leq n^{-20(r+s+t)}. \tag{9}$$

(Remember that $r, s, t \leq p = n^{1/3+o(1)}$).

The above deals with one particular $x, y, z \in \mathbf{B}(R, S, T)$.

Note next that there is a $1/(r + s + t)^2$-net $\mathcal{L}$ of $\mathbf{B}(R, S, T)$ of size at most $O((r + s + t)^{6(r+s+t)})$. (I.e., there is a set $\mathcal{L}$ of $O((r + s + t)^{6(r+s+t)})$ elements of $\mathbf{B}(R, S, T)$ so that for

each element (x,y,z) of $\mathbf{B}(R,S,T)$, there is some element (x',y',z') of $\mathcal{L}$ such that $|(x-x',y-y',z-z')| \le 1/(r+s+t)^2)$. Now, (9) implies that

$$\Pr\left(\exists (x',y',z') \in \mathcal{L} : |X(x',y',z')| \ge C_6 n^{1/2}\log n\right) \le n^{-12(r+s+t)}.$$

Lemma 6 follows from this and

$$\begin{aligned}|A(x,y,z)-A(x',y',z')| \le \\ |A(x,y,z)-A(x',y,z)|+|A(x',y,z)-A(x',y',z)|+|A(x',y',z)-A(x',y',z')| \\ \le \frac{4rst}{(r+s+t)^2}\left(\frac{1}{(st)^{1/2}}+\frac{1}{(rt)^{1/2}}+\frac{1}{(rs)^{1/2}}\right).\end{aligned}$$

Case 2 $|R| \ge |S|,|T|$ and either (i) $R \subseteq P$ and $S \cap P = T \cap P = \emptyset$ or (ii) $R \cap P = \emptyset$.

In either of the two sub-cases (i) and (ii), all the edges in G from $R \times (S \cup T)$ are from the random graph, not from the planted clique. Also, fix attention on one particular $(x,y,z) \in \mathbf{B}(R,S,T)$.

In this case, to prove an upper bound on $|X(x,y,z)|$, we bound its ℓth moment, where ℓ is an even integer to be chosen later.

Let I be the set of triples (i,j,k), where i,j,k are distinct and at most 2 of them are in P. Let Ω_ℓ denote the set of ordered sequences of ℓ triangles $T_1,T_2,\ldots,T_\ell$ where $T_i \in I \cap (R \times S \times T)$ for $i = 1,2,\ldots,\ell$. Let $X = X(x,y,z)$. We have

$$\mathbf{E}(X^\ell) = \sum_{\mathcal{T}\in\Omega_\ell} \mathbf{E}\left(\prod_{i=1}^{\ell} A(T_i)\right)\prod_{i=1}^{\ell} Z(T_i). \tag{10}$$

where if $T_i = (\alpha,\beta,\gamma)$ then $A(T_i) = A_{\alpha,\beta,\gamma}$ and $Z(T_i) = x_\alpha y_\beta z_\gamma$.

Consider an edge $e \in R \times (S \cup T)$ such that e appears in an odd number of triangles in $\mathcal{T}$. If we consider the measure preserving map f_e which deletes e if it appears in G and adds it otherwise then we see that

$$\prod_{i=1}^{\ell} A(f_e(T_i)) = -\prod_{i=1}^{\ell} A(T_i)$$

and so $\mathbf{E}\left(\prod_{r=1}^{\ell} A(T_r)\right) = 0$. This implies that it is sufficient to sum over those $\mathcal{T}$ in which each edge of $R \times (S \cup T)$ appears an even number of times. Let $\Omega^*_\ell(R,S,T)$ denote the set of ordered sequences $(i_1,j_1,k_1),\ldots,(i_\ell,j_\ell,k_\ell) \in (I \cap (R \times S \times T))^\ell$ such that each pair $(i,j) \in R \times S$ and each pair $(i,k) \in R \times T$ appears an even number of times.

LEMMA 7.

$$|\Omega^*_\ell(R,S,T)| \le \ell!\binom{\ell+r-1}{r-1}(4st)^{\ell/2}.$$

Proof Fix $d_i \ge 0, i \in R$ and let us first count the sequences in $\Omega'_\ell(R,S,T)$ in which $i \in R$ appears d_i times. Note that $\sum_{i\in R} d_i = \ell$. Now fix $i \in R$ and consider the d_i triangles $(i,s_1,t_1),\ldots(i,s_{d_i},t_{d_i})$ which contain i. Then consider the bipartite multigraph Γ on $S \cup T$

with edges $(s_1, t_1), \ldots, (s_{d_i}, t_{d_i})$. By assumption, each vertex of Γ is of even degree and so by Lemma 8 (below) there are at most $(4st)^{d_i/2}$ choices for Γ. Multiplying over i we see that there are at most $(4st)^{\ell/2}$ choices for any given sequence $d_1, \ldots, d_r$. The number of choices for $d_1, \ldots, d_r$ is at most $\binom{\ell+r-1}{r-1}$ and the lemma follows by multiplying by $\ell!$ to get an ordered sequence. □

Let $N(s, t, \mu)$ denote the number of bipartite multigraphs with vertex sets S, T on the two sides, with μ edges and such that each vertex has even degree.

LEMMA 8.

$$N(s, t, \mu) \leq (4st)^{\mu/2}.$$

Proof First note that for $f \geq 1$

$$\frac{2^{2f}}{2f^{1/2}} \leq \frac{(2f)!}{(f!)^2} \leq 2^{2f}.$$

Let $2e_1, 2e_2, \ldots, 2e_s$ and $2f_1, 2f_2, \ldots, 2f_t$ denote the degrees of vertices in S, T respectively. Then

$$\begin{aligned}
N(s,t,\mu) &\leq \sum_{\substack{2e_1+\cdots+2e_s=\mu \\ 2f_1+\cdots+2f_t=\mu}} \mu! \min\left\{\prod_{i\in S}\frac{1}{(2e_i)!}, \prod_{j\in T}\frac{1}{(2f_j)!}\right\} \\
&\leq \sum_{\substack{2e_1+\cdots+2e_s=\mu \\ 2f_1+\cdots+2f_t=\mu}} \mu! \left(\prod_{i\in S}\frac{1}{(2e_i)!}\prod_{j\in T}\frac{1}{(2f_j)!}\right)^{1/2} \\
&\leq \sum_{\substack{2e_1+\cdots+2e_s=\mu \\ 2f_1+\cdots+2f_t=\mu}} (\mu/2)!^2 2^\mu \prod_{i\in S}\frac{2^{1/2}e_i^{1/4}}{2^{e_i}e_i!}\prod_{j\in T}\frac{2^{1/2}f_j^{1/4}}{2^{f_j}f_j!} \\
&\leq 2^\mu \left(\sum_{e_1+\cdots+e_s=\mu/2}(\mu/2)!\prod_{i\in S}\frac{1}{e_i!}\right)\left(\sum_{f_1+\cdots+f_t=\mu/2}(\mu/2)!\prod_{j\in T}\frac{1}{f_j!}\right) \\
&= 2^\mu s^{\mu/2}t^{\mu/2},
\end{aligned}$$

the last because $\left(\sum_{e_1+\cdots+e_t=\mu/2}(\mu/2)!\prod_{j\in T}\frac{1}{e_j!}\right)$ is the number of ways of parititioning the set $\{1, 2, \ldots \mu/2\}$ into t subsets and this number also equals $t^{\mu/2}$. □

Thus,

$$\begin{aligned}
\mathbf{E}(X^\ell) &= \sum_{\mathcal{T}\in\Omega_\ell^*}\mathbf{E}\left(\prod_{r=1}^{\ell}A(T_r)\right)\prod_{r=1}^{\ell}Z(T_r) \\
&\leq |\Omega_\ell^*|\cdot\frac{8}{(rst)^{\ell/2}} \\
&\leq \binom{\ell+r-1}{r-1}\cdot\frac{2^{\ell+3}\ell!}{r^{\ell/2}} \\
&\leq \frac{2^{\ell+4}\ell^{\ell+1/2}e^r}{r^{\ell/2}}.
\end{aligned}$$

Now ℓ even implies that $X^\ell \geq 0$ and so applying the Markov inequality, we see that for any $\xi > 0$,

$$\mathbf{Pr}(X > \xi) \leq \frac{2^{\ell+4}\ell^{\ell+1/2}e^r}{\xi^\ell r^{\ell/2}}.$$

Putting $\xi = C_6 n^{1/2}\log n$ and $\ell = (r+s+t)\log n$, we see that

$$\mathbf{Pr}(X(x,y,z) \geq C_6 n^{1/2}\log n) \leq n^{-20(r+s+t)}. \tag{11}$$

This completes the proof of Lemma 6.

4 Proof of the Corollaries

Corollary 2 follows from Theorem 1 and the following:

$$\left|\sum_{i,j,k\in P^{3*}} x_i y_j z_k\right| \leq \left|\left(\sum_{i\in P} x_i\right)\left(\sum_{j\in P} y_j\right)\left(\sum_{k\in P} z_k\right)\right| + |y\cdot z|\left|\sum_P x_i\right| + |x\cdot z|\left|\sum_P y_j\right| + |x\cdot y|\left|\sum_P z_k\right| + \left|\sum_{i\in P} x_i y_i z_i|\right| \leq 3p^{1/2}.$$

□

For Corollary 3 we write $x^* = (x^*\cdot u)u + x'$, where x' is orthogonal to u, similarly for y^*, z^*. This splits $A(x^*,y^*,z^*)$ into the sum of 8 parts. Using (3), we get

$$A(u,u,u) \leq A(x^*,y^*,z^*) \leq o(A(u,u,u)) + (x^*\cdot u)(y^*\cdot u)(z^*\cdot u)A(u,u,u),$$

and the corollary follows. □

5 Proof of Theorem 4

Now, we prove Theorem 4. Let v with $|v| = 1$ be the given vector. Define a vector w by: $w_i = \max(v_i, 0)$. Clearly, $\sum_{i\in P} w_i \geq \sum_P v_i$. For ease of notation, we re-number the indices of coordinates so that $w_1 \geq w_2 \geq \ldots w_n$. Since v is given, we can explicitly do this reordering. Also for convenience, we let $w_{n+1} = 0$. After this renumbering, we let

$$S_k = \{1,2,\ldots k\}, \qquad T_k = S_k \cap P, \qquad t_k = |T_k| \qquad k = 1,2,\ldots,n. \tag{12}$$

LEMMA 9. *If $\sum_{i\in P} v_i \geq C_8 \log n$, then for some integer k,*

$$t_k \geq C_8\sqrt{k\log n}/3.$$

Proof Assume for the sake of contradiction that $\sum_{i\in P} v_i \geq C_8 \log n$ and that for all k, $t_k < C_8\sqrt{k\log n}/3$.

$$\begin{aligned}\sum_{i\in P} w_i &= \sum_{k=1}^{n} t_k\,(w_k - w_{k+1}) \leq \frac{1}{3}C_8\sqrt{\log n}\sum_{k=1}^{n}\sqrt{k}(w_k - w_{k+1}) \\ &= \frac{1}{3}C_8\sqrt{\log n}\sum_{k=1}^{n} w_k(\sqrt{k}-\sqrt{k-1}) \leq \frac{2}{3}C_8\sqrt{\log n}\sum_{k=1}^{n}\frac{w_k}{\sqrt{k}} \\ &\leq \frac{2}{3}C_8\sqrt{\log n}|w|\left(\sum_{k=1}^{n}\frac{1}{k}\right)^{1/2} \leq \frac{3}{4}C_8\log n,\end{aligned}$$

using $\frac{2}{\sqrt{k}} \geq \sqrt{k}-\sqrt{k-1}$ and also the Cauchy-Scwartz inequality. This contradiction proves the Lemma. □

Let G be the graph we are given (the random graph plus the planted clique.) Let M be its adjacency matrix, where we put a +1 for an edge and -1 for a non-edge. For a subset S of V, let G^S denote the induced subgraph on S and M^S the $|S|\times|S|$ adjacency matrix of G^S. (In our definition of adjacency matrix, we have 1's on the diagonal). We may write

$$M = puu^T + \hat{M} - \tilde{M}, \tag{13}$$

where $\hat{M}$ is the adjacency matrix of the random graph and $\tilde{M}$ is the adjacency matrix of the sub-graph induced on P of the random graph. [$\tilde{M}$ has 0 entries outside $P\times P$.] We may similary write for any $S\subseteq V$,

$$M^S = tu^S u^{S^T} + \hat{M}^S - \tilde{M}^S, \tag{14}$$

where $|S\cap P| = t$ and u^S denotes the vector with $1/\sqrt{t}$ in the $S\cap P$ positions and 0 elsewhere.

LEMMA 10. *With probability at least $1-n^{-3}$, we have that for all $S\subseteq V$,*

$$\max\{\lambda_1(\hat{M}^S), \lambda_1(\tilde{M}^S)\} \leq 100\sqrt{|S|\log n}$$

where λ_1 denotes the largest absolute value of an eigenvalue.

Proof For each fixed S, the matrix $\hat{M}^S$ is a random symmetric matrix. It is known [2] that with probability at least $1-4e^{-10|S|\log n}$, we have that $|\lambda_1(\hat{M}^S)| \leq 100\sqrt{|S|\log n}$. For each $s\in\{1,2,\ldots n\}$, there are at most n^s subsets S of V with $|S|=s$. So the probability that the assertion of the Lemma does not hold is at most $\sum_{s=1}^{n} n^s e^{-10s\log n} \leq 1/(2n^3)$. $\tilde{M}^S$ is dealt with similarly.

□

For notational convenience, we let M^k denote M^{S_k} (see (12)) and similarly for $\hat{M}^k, \tilde{M}^k$. The first step of our algorithm is to run through $k=1,2,\ldots n$, find $\lambda_1(M^k)$ and stop when for the first time, we find a k such that

$$\lambda_1(M^k) \geq 1000\sqrt{k\log n}. \tag{15}$$

LEMMA 11.
(i) *If $C_8 \geq 3000$ then the algorithm will find a k satisfying (15).*
(ii) *For any k satisfying (15), we have:*
(a) *if a is the top eigenvector of M^k, then $\left|\sum_{i\in T_k} a_i\right| \geq 0.8\sqrt{t_k}$ and*
(b) $t_k \geq 800\sqrt{k\log n}$.

Proof Let u^k be a vector defined by $u^k_i = 1/\sqrt{t_k}$ for $i \in T_k$ and 0 elsewhere. Then, $u^{k^T} M^k u^k = t_k$; this implies that $\lambda_1(M^k) \geq t_k$. Now (i) follows from Lemma 9.

(ii) Suppose now k satisfies (15) and a is the top eigenvector of M^k. Then, we have (recalling (14) and using Lemma 10),

$$1000\sqrt{k\log n} \leq a^T M^k a = t_k(u^k\cdot a)^2 + a^T\hat{M}^k a - a^T\tilde{M}^k a \leq t_k + 200\sqrt{k\log n}.$$

Thus,

$$t_k \geq 800\sqrt{k\log n}.$$

Also,

$$t_k \leq \lambda_1(M^k) \leq t_k(u^k\cdot a)^2 + 200\sqrt{k\log n} \leq t_k\left((u^k\cdot a)^2 + \frac{1}{4}\right)$$

which implies $(u^k\cdot a)^2 \geq 3/4$. This proves (ii).

□

LEMMA 12. *There is a polynomial time algorithm which given $S \subseteq V$ and a unit length vector a with support S, finds a $P' \subseteq V$ with the following property:*
If $|S\cap P| \geq 800\sqrt{|S|\log n}$ and $\sum_{i\in S\cap P} a_i \geq 0.8\sqrt{|S\cap P|}$, then $P' = P$.

Proof Re-number the coordinates, so that $a_1 \geq a_2 \geq \ldots \geq a_n$. In particular this implies that if $\ell \leq |S|$ then $[\ell] \subseteq S$. We wish to prove that there is an integer ℓ such that

$$|[\ell]\cap P| \geq \max\{\ell/100, 10\log n\} \tag{16}$$

First, if $|S\cap P| \geq |S|/10$, then we can take $\ell = |S|$. So assume that $t = |S\cap P| < |S|/10$ and let $\ell = 4t$. Now

$$\sum_{i\leq\ell; i\in P} a_i \leq \sqrt{|[\ell]\cap P|}$$

and so

$$\sum_{i\geq\ell+1; i\in P} a_i \geq 0.8\sqrt{|S\cap P|} - \sqrt{|[\ell]\cap P|} \text{ and } \sum_{i\leq\ell} a_i \geq \frac{\ell}{t}\left(0.8\sqrt{|S\cap P|} - \sqrt{|[\ell]\cap P|}\right).$$

But,

$$\sum_{i\leq\ell} a_i \leq \sqrt{\ell}.$$

This implies

$$\sqrt{|[\ell]\cap P|} \geq 0.8\sqrt{|S\cap P|} - 0.25\sqrt{\ell} = .15\sqrt{\ell}. \tag{17}$$

Also, we have $|S\cap P|^2 \geq 640000|S|\log n$ and so $|S\cap P| \geq 640000\log n$ and then (16) follows from (17) and $|[\ell]\cap P| \geq 4(.15)^2|S\cap P|$.

Now to construct P we try all values of ℓ. For each value of ℓ, we pick a random set Q_1 of $10\log n$ from $[\ell]$. For ℓ satisfying (16) there is at least a $10^{-20\log n}$ chance that $Q_1 \subseteq P$. Now **whp** no set of $10\log n$ vertices in P have more than $2\log n$ common neighbours outside P. Indeed the probability of the contrary event is at most

$$\binom{p}{10\log n}\binom{n}{2\log n}2^{-20(\log n)^2} = o(1).$$

So let Q_2 be the set of common neighbours of Q_1. By assumption we have $P \subseteq Q_2$ and $|Q_2 \setminus P| \leq 2\log n$. Also, **whp** for every $10\log n$-subset Q of P, no common neighbour outside P has $3p/4$ neighbours in P. Indeed the probability of the contrary event is at most

$$n\binom{p}{10\log n}\binom{n}{2\log n}2^{-p/12} = o(1).$$

Thus P is the set of vertices of degree at least $7p/8$ in the subgraph of G induced by Q_2. □

Acknowledgement We thank Santosh Vempala for interesting discussions on this problem.

References

[1] N. Alon, M. Krivelevich and B. Sudakov, Finding a large hidden clique in a random graph, *Random Structures and Algorithms* 13 (1998) 457-466.

[2] N. Alon, M. Krivelevich and V. H. Vu, On the concentration of eigenvalues of random symmetric matrices", Israel Journal of Mathematics 131 (2002) 259-267.

[3] N. Alon and J.H. Spencer, *The Probabilistic Method,* Wiley, (second edition) 2000.

[4] U. Feige and R.Krauthgamer, Finding and certifying a large hidden clique in a semi-random graph, *Random Structures and Algorithms*, 13 (1998) 457-466.

[5] J. Hastad, Clique is hard to approximate within $n^{1-\epsilon}$, *Proceedings of the 37th Annual IEEE Symposium on Foundations of Computing*, (1997) 627-636.

[6] M. Jerrum, Large cliques elude the Metropolis process, *Random Structures and Algorithms* 3 (1992) 347-359.

[7] A. Juels and M. Peinado, Hiding Cliques for Cryptographic Security, Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms, (1998) 678-684.

[8] R. Karp, Reducibility among combinatorial problems, in *The complexity of computer computations*, R. Miller and J. Thatcher (eds.) Plenum Press, New York (1972) 85-103.

[9] R. Karp, The probabilistic analysis of some combinatorial search algorithms, in *Algorithms and Complexity: New Directions and Recent Results*, J.F. Traub, ed., Academic Press (1976) 1-19.

[10] L. Kucera, Expected complexity of graph partitioning problems, *Discrete Applied Mathematics* 57 (1995) 193-212.

[11] F. McSherry, Spectral Partitioning of random graphs, FOCS 2001, 529-537.

Foundations of Software Technology and Theoretical Computer Science (Bangalore) 2008.
Editors: R. Hariharan, M. Mukund, V. Vinay; pp 199-210

All-Norms and All-L_p-Norms Approximation Algorithms

Daniel Golovin[1]*, **Anupam Gupta**[1]†, **Amit Kumar**[2]‡, **Kanat Tangwongsan**[1]†

[1] Computer Science Department
Carnegie Mellon University, Pittsburgh PA, USA 15213.

[2] Department of Computer Science & Engineering
Indian Institute of Technology, Hauz Khas, New Delhi, India 110016.

ABSTRACT. In many optimization problems, a solution can be viewed as ascribing a "cost" to each client, and the goal is to optimize some aggregation of the per-client costs. We often optimize some L_p-norm (or some other symmetric convex function or norm) of the vector of costs—though different applications may suggest different norms to use. Ideally, we could obtain a solution that optimizes several norms simultaneously. In this paper, we examine approximation algorithms that simultaneously perform well on all norms, or on all L_p norms.

A natural problem in this framework is the L_p Set Cover problem, which generalizes SET COVER and MIN-SUM SET COVER. We show that the greedy algorithm *simultaneously gives a* $(p + \ln p + O(1))$*-approximation for all* p*, and show that this approximation ratio is optimal up to constants* under reasonable complexity-theoretic assumptions.

We additionally show how to use our analysis techniques to give similar results for the more general *submodular set cover*, and prove some results for the so-called *pipelined set cover* problem. We then go on to examine approximation algorithms in the "all-norms" and the "all-L_p-norms" frameworks more broadly, and present algorithms and structural results for other problems such as k-facility-location, TSP, and average flow-time minimization, extending and unifying previously known results.

1 Introduction

When the solution to an optimization problem affects multiple people or organizations, there is often a trade-off between various efficiency and fairness measures. Typically, there is an abstract "cost" associated with each participant and the objective function is some aggregation of the individual costs. The method of aggregation represents our relative priorities concerning efficiency and fairness. E.g., in *k-median*, given demand points $D \subseteq V$ in a metric space (V, d), we must select k facilities to open: the cost associated with each participant $d \in D$ is its distance to the nearest open facility. Each solution thus induces a cost vector $\mathbf{C} \in \mathbb{R}_+^{|D|}$, and the objective is to minimize $\|\mathbf{C}\|_1 = \sum_{d \in D} \mathbf{C}_d$, the sum of the participant costs: hence, this method of aggregation favors global efficiency over fairness. Another extreme is

*Supported in part by NSF ITR grants CCR-0122581 (The Aladdin Center) and IIS-0121678

†Supported in part by an NSF CAREER awards CCF-0448095 and CCF-0729022, and by an Alfred P. Sloan Fellowship.

‡Part of this work done while visiting Max-Planck-Institut für Informatik, Saarbrücken, Germany.

FSTTCS 2008
IARCS Annual Conference on
Foundations of Software Technology and Theoretical Computer Science
http://drops.dagstuhl.de/opus/volltexte/2008/1753

k-center, where we minimize the fairer objective function $\|\mathbf{C}\|_\infty$, the maximum participant cost. Other examples where such trade-offs appear include:

- *Sequencing problems*: $\mathbf{C}$ measures the "time" of service for each participant, for example the cover times of the elements in a set cover instance, or the times to reach the vertices in a TSP instance.
- *Scheduling problems*: $\mathbf{C}$ could be the load of the machines or the flow-times of the individual jobs.
- *Allocation problems*: $\mathbf{C}$ measures the quality of service of each participant, for example congestion or dilation in routing problems, and distances in facility location problems.

In general, there are many aggregation functions we might wish to consider. However, if we are feeling particularly ambitious, we might ask if we can efficiently find solutions that *simultaneously* approximate the optimal solutions for each member of a large class of aggregation functions. Formally, we are given a minimization problem and a class of aggregation functions $\mathcal{F}$. For each $f \in \mathcal{F}$, let $\mathbf{C}_f^*$ be the feasible vector minimizing $f(\cdot)$. Then for as small an α as possible, we want to find a feasible cost vector $\mathbf{C}$ such that $f(\mathbf{C}) \leq \alpha \cdot f(\mathbf{C}_f^*)$ for all $f \in \mathcal{F}$. Such a vector $\mathbf{C}$ is a *simultaneous α approximation* for $\mathcal{F}$.

In this paper, we will consider two classes of aggregation functions: the class of *Minkowski L_p norms* $\{L_p \mid p \in \mathbb{R}_{\geq 1}\} \cup \{L_\infty\}$ (i.e., All L_p Norm results), and the class of *all symmetric norms* (i.e., AllNorm results). The L_p norm of $\mathbf{C}$, which is $\|\mathbf{C}\|_p := (\sum_i \mathbf{C}_i^p)^{1/p}$ for a real value $1 \leq p < \infty$ and $\max_i \mathbf{C}_i$ for $p = \infty$, provides a nicely parameterized way of quantifying the efficiency/fairness trade-off.

The question of all-norm minimization was investigated by Kleinberg et al. [KRT01] in their study of *fair* resource allocation algorithms for routing and load balancing, and the problem of all L_p-norms minimization was considered by Azar et al. [AERW04] for machine scheduling. Subsequent work on these topics was done in the papers [KK00, GMP01, GM06]—the concepts studied here are closely linked to *submajorization* of vectors [HLP88], which is even stronger than simultaneously approximating all symmetric norms (and hence all Minkowski norms), see [GM06] for details and many interesting results derived therefrom. For the comprehensive treatment of submajorization and AllNorm approximation, see books by Hardy et al. [HLP88] or Steele [Ste04].

1.1 All L_p-norms Set Cover

The classical set cover problem wants to pick a small number of sets one-by-one to cover the elements *early in the worst-case*, whereas the *min-sum* set cover problem tries to pick the sets to cover the elements *early "on average"*. In this paper, the first question we consider is how to pick sets so that the second (or higher) moments are small: this is just the L_p-Set Cover (L_p SC) problem. We show that the greedy algorithm is, in fact, a $(p + \ln p + 3)$-approximation for all L_p norms *simultaneously!* Moreover, for any fixed p, we cannot hope to do much better using any other algorithm, and hence greedy is essentially the best.

Formally, a set cover instance consists of a ground set $\mathcal{U}$ of n elements, a collection $\mathcal{F}$ of subsets of $\mathcal{U}$, and a cost function $c : \mathcal{F} \to \mathbb{R}_+$. An algorithm picks sets $S_1, S_2, \ldots, S_t$ (in that order) so that their union $\cup_i S_i$ is $\mathcal{U}$. On this ordering, let c_i be the cost of the set S_i; i.e., $c_i = c(S_i)$. Informally, we may think of S_i as corresponding to an action a_i that covers the elements of S_i, and c_i is the time required to execute a_i. Let the *cover index* of an element

$e \in \mathcal{U}$ be defined as $index(e) = \min\{i : e \in S_i\}$; i.e., the position of the first set that contains e. The *cover time* of an element $e \in \mathcal{U}$ is defined to be the time required to cover e if we execute actions in this order: i.e., $time(e) = \sum_{i=1}^{index(e)} c_i$. Note that for the case of unit costs, the cover index and cover time are the same. Given the sequence of sets that the algorithm picks, we obtain a *cover time vector* $\mathbf{C} \in \mathbb{R}_+^n$, where $\mathbf{C}_e$ is the cover time of the element $e \in \mathcal{U}$. The L_p set cover problem is then to find the ordering that minimizes $\|\mathbf{C}\|_p$. It is easy to see that using the L_1 norm and unit costs we obtain the MIN-SUM SET COVER problem [FLT04], whereas using the L_∞ norm we obtain the classical set cover problem [Chv79, Lov75, Joh74].

We prove the greedy algorithm achieves an approximation ratio of $(1 + o(1)) \min\{p, \ln n\}$ for L_p set cover (which is simultaneously optimal for all L_p norms), and also an $O(\log n)$-approximation in the AllNorm model. Moreover, even if we focus on any fixed value of p, we show that it is impossible to approximate the L_p set cover problem better than $\Omega(p)$ unless $\mathbf{NP} \subseteq \mathbf{DTIME}(n^{O(\log \log n)})$. This lower bound holds for all functions $p(n)$ such that $1 \le p(n) \le \frac{1-\epsilon}{2} \ln(n)$ for all n. We also show that the greedy algorithm achieves an $(p + \ln p + 3)$-approximation in the L_p *Submodular Set Cover problem*, which is a generalization of the L_p set-cover problem to arbitrary submodular functions.

To the best of our knowledge, there has not been any prior work on All L_p Norm approximation for Set Covering problems seeking to minimize all $\|\mathbf{C}\|_p$; of course, there is much work for special values of p. For the classical MINIMUM SET COVER problem (minimize $\|\mathbf{C}\|_\infty$), an $(1 + o(1)) \ln n$-approximation is known both by greedy and by LP rounding [Joh74, Lov75, Chv79, Sla97, Sri99]. Moreover, one cannot get an $(1 - \epsilon) \ln n$-approximation unless $\mathbf{NP} \subseteq \mathbf{DTIME}(n^{O(\log \log n)})$ [Fei98]. For the MIN-SUM SET COVER problem (minimize $\|\mathbf{C}\|_1$), we know that greedy is an optimal 4-approximation [FLT04] (see also [BNBH+98, CFK03]).

1.2 Overview of our Other Results and Related Work

Pipelined Set Cover: This problem was studied in the All L_p Norm framework by Munagala et al. [MBMW05], and seeks to minimize $\|\mathbf{R}\|_p$ where $\mathbf{R}_i$ is the number of *uncovered* elements before the i^{th} set is chosen. To put this in context, the L_1 norm for this problem is the MIN-SUM SET COVER problem, and the L_∞ norm is just $|\mathcal{U}|$. Munagala et al. show that the output of the greedy algorithm is simultaneously a $9^{1/p}$-approximation for the L_p norm, and also give a local-search algorithm that is a $4^{1/p}$ approximation. We show how our proof ideas from MIN-SUM SET COVER give an $(1 + \frac{\ln p}{p} + \frac{3}{p})$-approximation guarantee for the greedy algorithm for this problem; while slightly worse than the previous known guarantee (note $1 + \frac{\ln(4)}{p} \le 4^{1/p} \le 1 + \frac{3}{p}$ for all $p \ge 1$), it extends to the case of *non-uniform costs* where no guarantee was known for the greedy algorithm.

Norm Sampling: We consider the problem of finding a good representative set for the class of all L_p norms with $p \in \mathbb{R}_{\ge 1} \cup \{\infty\}$—namely a set $S \subset \mathbb{R}_{\ge 1} \cup \{\infty\}$ such that an simultaneous α-approximation for all L_p norms with $p \in S$ implies a simultaneous $O(\alpha)$-approximation for all L_p norms with $p \in \mathbb{R}_{\ge 1} \cup \{\infty\}$. This leads us to a notion of *norm sampling*, and we give tight bounds for the size of S necessary and sufficient to well represent (various subsets of) the L_p norms, as well as explicit constructions of such sets.

Facility Location Problems: We return to the example at the beginning of the introduction,

where we seek to open k facilities to minimize $\|\mathbf{C}\|_p$, where $\mathbf{C}$ is the vector of assignment costs of demands. It is known that one can get $O(1)$-approximation algorithm for all norms provided we open $O(k \log n)$ facilities [KK00, GM06], and such a $O(\log n)$ blow-up in the number of open facilities cannot be avoided [KK00]. In contrast, we use the above norm-sampling ideas to give an $O(1)$-approximation algorithm for all L_p norms with *integer values of* p provided we open $O(k\sqrt{\log n})$ facilities, and show that opening $\Omega(k \cdot (\log_k n)^{1/3})$ facilities may be necessary in some instances.

Results via Partial Covering: For sequencing problems such as TSP, where the cost vector is the time to reach each of the n vertices in some graph, or sequencing versions of covering problems (of which L_p set cover is a good example), we show how to use partial covering results to generate AllNorm approximations. For example, we give an AllNorm 16-approximation result for the TSP by drawing on the elegant techniques of Blum et al. [BCC+94] and the large body of subsequent and related work. To extend the result to other problems (like vertex cover and Multicut on trees), we use results from the well-studied area of *partial* covering problems, and the papers of [GKS04, KPS06] in particular.

Flow-Time Scheduling: Some scheduling problems naturally lend themselves to a job-centric perspective. We consider scheduling jobs on parallel machines and look at the vector of flow times for each job: given ε-factor extra speed for each machine, we get an $O(1/\varepsilon^{O(1)})$- approximation algorithms for all norms. This extends previous work of Chekuri et al. [CGKK04] (who proved the result for all L_p norms), Bansal and Pruhs [BP03] (who gave an All L_p Norm result for a single machine). Related work includes results in the machine-centric model (see, e.g., [AERW04, GM06, AT04, AE05]).

1.3 Preliminaries and Notations

A *norm* $\|\cdot\|$ on vectors of length n is a function from $\mathbb{R}^n \to \mathbb{R}$ that satisfies the following: $\|\alpha X\| = |\alpha| \|X\|$ for any $\alpha \in \mathbb{R}$ and $X \in \mathbb{R}^n$, and secondly $\|X + Y\| \leq \|X\| + \|Y\|$ for $X, Y \in \mathbb{R}^n$. The Minkowski L_p norm of X is $\|X\|_p = (\sum_i X_i^p)^{1/p}$ for a real value $1 \leq p < \infty$; the L_∞ norm is just $\|X\|_\infty = \max_i X_i$. It is well-known that for all $X \in \mathbb{R}^n$ and $p < q$, $\|X\|_p \geq \|X\|_q$ [HLP88].

All of the problems we consider in this paper have the property that a solution to the problem induces a vector of length n; thus, for each instance $\mathcal{I}$ of such a problem, we have a set $V(\mathcal{I})$ consisting of all vectors that are induced by some feasible solution to the instance. For a norm $\|\cdot\|$, let $\|X\|$ denote the norm of the vector X. We state two well-known facts for easy reference: the latter follows directly from the convexity of x^p.

Fact 1 (Generalized AM-GM [Ste04]) $\frac{1}{p}A + \frac{p-1}{p}B \geq A^{1/p}B^{(p-1)/p}$

Fact 2 (The Discrete Differential) *Let $p \geq 1$. If the real numbers a, b, and c satisfy $c = a - b \geq 0$, then $a^p - b^p \leq c \cdot p \cdot a^{p-1}$.*

2 The L_p Set Cover Problem

We show that the greedy algorithm simultaneously gives an $(p + \ln p + 3)$-approximation for the L_p Set Cover problem for all p, hence generalizing the fact that it is an $O(\log n)$-approximation for MIN SET COVER (i.e., the $L_\infty \approx L_{\log n}$ case) and 4-approximation for the L_1 case. We then show that for any p, we give a hardness of approximation result of $\Omega(p)$.

2.1 An Upper Bound for the Greedy Algorithm

Consider the familiar setup. We have a universe $\mathcal{U}$ of n elements and a family $\mathcal{F}$ of subsets of $\mathcal{U}$. The greedy algorithm picks sets $S_1, S_2, \ldots, S_t$ from $\mathcal{F}$ until $\cup_i S_i = \mathcal{U}$, such that each S_i satisfies $|S_i \setminus (\cup_{j<i} S_j)| = \max_{S \in \mathcal{F}} \{|S \setminus (\cup_{j<i} S_j)|\}$.

Let c_i be the cost of the set S_i. Let s_i be the cumulative cost of the first i sets picked by the greedy algorithm. That is, $s_0 = 0$ and $s_{i+1} = s_i + c_{i+1}$. Let $X_i = S_i \setminus (\cup_{j<i} S_j)$ be the set of elements with cover index i. Let $R_i = \mathcal{U} - \bigcup_{j=1}^{i-1} X_j$ be the elements uncovered just before the i^{th} set is picked. We use S_i^*, c_i^*, s_i^*, X_i^* and R_i^* to denote the analogous quantities for the optimal algorithm.

For a fixed value of p, the cost of the greedy algorithm (denoted by $\underline{\text{greedy}}$) can be written in terms of the values X_i and R_i as follows:

$$\underline{\text{greedy}} = \left(\textstyle\sum_{i>0} s_i^p \, |X_i|\right)^{1/p} \tag{1}$$

$$= \left(\textstyle\sum_{i>0} (s_i^p - s_{i-1}^p) |R_i|\right)^{1/p}, \tag{2}$$

where the second expression follows from the fact that $|R_{i+1}| = |R_i| - |X_i|$. The cost of the optimal algorithm can be expressed in a similar fashion.

The following lemma upper bounds the cost of greedy by a somewhat exotic expression, which will later turn out to be crucial to our analysis.

Lemma 3 (Upper-bound on Greedy)

$$\underline{\text{greedy}}^p \;\le\; (\underline{\text{greedy}}')^p \;\stackrel{def}{=}\; \sum_{i>0} \left(p \cdot c_i \frac{|R_i|}{|X_i|}\right)^p \cdot |X_i|$$

PROOF. Let $A_i = \left(p \cdot c_i \frac{|R_i|}{|X_i|}\right)^p \cdot |X_i|$ be the i^{th} term in the summation above. Taking the i^{th} terms in the expressions (1) and (2) measuring the cost of the greedy algorithm, and raising them to the p^{th} powers, define $B_i = (s_i^p - s_{i-1}^p)\,|R_i|$ and $C_i = s_i^p \, |X_i|$. It follows from Fact 1 that $\frac{1}{p} A_i + \frac{p-1}{p} C_i \ge A_i^{1/p} C_i^{(p-1)/p} = p \cdot c_i \cdot s_i^{p-1} |R_i| \ge B_i$. The last inequality follows from Fact 2 and the observation that $c_i = s_i - s_{i-1}$. Now, rearranging terms, we have that $A_i \ge p\, B_i - (p-1)\, C_i$; summing this over all i and noting that $\sum_i B_i = \sum_i C_i = \underline{\text{greedy}}^p$, we get that $\sum_i \left(p \cdot c_i \frac{|R_i|}{|X_i|}\right)^p \cdot |X_i| = \sum_i A_i \ge p \sum_i B_i - (p-1) \sum_i C_i = \underline{\text{greedy}}^p$, which completes the proof. ∎

Given this upper bound on the cost of the greedy algorithm, we now compare this to the optimal L_p set cover cost. While the structure of the remainder of the proof follows that by Feige et al. [FLT04] for the L_1 case, we need a few new ingredients, most notably obtaining the correct "price" function.

Theorem 4 (L_p Approximation Guarantee) *The greedy algorithm gives a $(1+p)^{1+1/p} \le (p + \ln p + 3)$-approximation for the L_p set cover problem.*

PROOF. Recall that $\underline{\text{greedy}}$ and $\underline{\text{opt}}$ denote the cost of the greedy algorithm and the optimal algorithm, respectively. We show $\underline{\text{opt}}$ graphically as in Figure 1 (left). The horizontal axis is divided into n equal columns, corresponding to the elements of the universe $\mathcal{U}$. The elements are arranged from left to right in the order that the optimal algorithm covers them.

The column corresponding to the element x has height $(s^*_{index^*(x)})^p$. Thus the area under the curve is $\underline{\text{opt}}^p$.

As Lemma 3 shows, $\underline{\text{greedy}}^p$ can be upper-bounded by the expression $(\underline{\text{greedy}}')^p$. The right panel of Figure 1 models the quantity $(\underline{\text{greedy}}')^p$. The diagram has n columns corresponding to the elements of $\mathcal{U}$ appearing from left to right in the order that the greedy algorithm covers them. For each element of X_i, its corresponding column has height $[p \cdot c_i|R_i|/|X_i|]^p$.

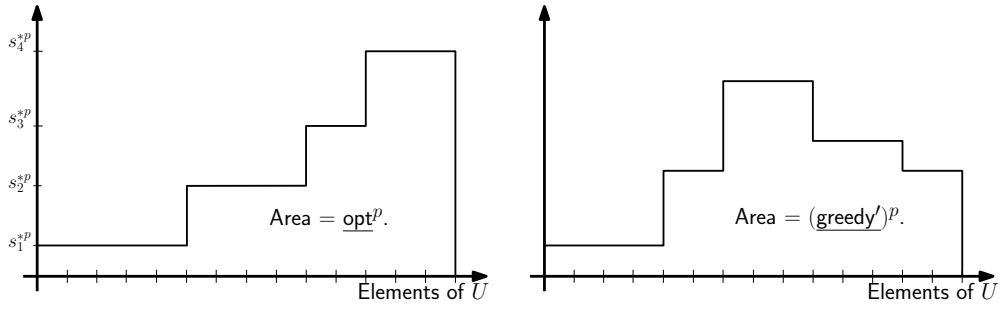


Figure 1: Graphical representations of the cost of the optimal algorithm (left) and an upper bound of the cost of the greedy algorithm (right).

We will now show that the area of the $(\underline{\text{greedy}}')^p$ curve is at most $p^p(1+p)(1+1/p)^p$ times the area of the $\underline{\text{opt}}^p$ curve. To prove this, we scale the $(\underline{\text{greedy}}')^p$ curve down by $[p(1+1/p)]^p$ vertically and by $1+p$ horizontally, and place this scaled curve so that its bottom-right is aligned with the bottom-right of the $\underline{\text{opt}}^p$ curve. Now consider a point $q = (x, y)$ on the original $(\underline{\text{greedy}}')^p$ curve. Suppose the point q corresponds to an element of X_i, so $y \le [p \cdot c_i|R_i|/|X_i|]^p$. Also the distance to q from the right side is at most $|R_i|$. Therefore, the height of the point q after scaling, which we denote by h, is at most $\left(\frac{1}{1+1/p} \cdot \frac{|R_i|}{|X_i|/c_i}\right)^p$, and the distance from the right (after scaling), denoted by r, is at most $|R_i|/(1+p)$.

In order to show that the point q (after scaling) lies within the $\underline{\text{opt}}^p$ curve, it suffices to show that when the optimal algorithm's cover time is $h^{1/p}$, at least r points remain uncovered. Consider the set R_i. Within this set, the greedy algorithm covers the most elements per unit increase in cover time. Therefore, the number of elements from R_i that the optimal algorithm can cover in time $h^{1/p}$ is at most $\left(\frac{1}{1+1/p} \cdot \frac{|R_i|}{|X_i|/c_i}\right)\frac{|X_i|}{c_i} \le \frac{1}{1+1/p}|R_i|$, and so at least $\frac{1}{1+p}|R_i|$ elements remain uncovered at time $h^{1/p}$. Since $|R_i|/(1+p) \ge r$, this implies that q (after scaling) lies within the $\underline{\text{opt}}^p$ curve, and hence the scaled-down version of the $(\underline{\text{greedy}}')^p$ curve is completely contained within the $\underline{\text{opt}}^p$ curve. Quantitatively, this implies that $\underline{\text{greedy}}^p \le (1+p)[p(1+1/p)]^p\ \underline{\text{opt}}^p = (1+p)^{p+1}\ \underline{\text{opt}}^p$. It can be shown that $(1+p)^{1+1/p} \le p + \ln p + 3$ for $p \ge 1$, which completes the proof. ■

Having shown that the greedy algorithm gives an $O(p)$ approximation for any fixed p, in the full version we give an example for which the greedy algorithm is an $\Omega(p)$ approximation.

Theorem 5 (Tight Example for Greedy) *There is a set system on which greedy yields an $\Omega(p)$ approximation.*

2.2 A Matching Hardness Result for L_p Set Cover

In this section, we show that the greedy algorithm achieves the best possible approximation factor up to constant factors; indeed, we show that even if we fix a value of p, there is no polynomial-time algorithm approximating L_p set cover problem better than $\Omega(p)$ unless $\mathbf{NP} \subseteq \mathbf{DTIME}(n^{O(\log\log n)})$. We first prove a technical lemma.

Lemma 6 *Let $\#OPT(I)$ denote the number of sets an optimal algorithm (for the classical min set cover) needs to cover the set-cover instance I. Let $\epsilon > e^2$. Let $t : \mathbb{N} \to \mathbb{R}_+$ be a non-decreasing function such that $1 \le t(n) \le \log_\epsilon n$ for all n. If there exists an efficient algorithm A such that for all $n > 0$, for all instance I with n elements, A covers at least $n \cdot (1 - \epsilon^{-t(n)})$ elements with $t(n) \cdot \#OPT(I)$ sets, then $\mathbf{NP} \subseteq \mathbf{DTIME}(n^{O(\log\log n)})$.*

The proof is standard and can be found in the full version [GGKT07].

Lemma 7 *Suppose $\delta > 0$, and $p(n) = \omega(1)$ is non-decreasing and $1 \le p(n) \le (\frac{1}{2} - \delta)\ln n$ for all n. Then the L_p set-cover problem is $\Omega(p)$-hard to approximate unless $\mathbf{NP} \subseteq \mathbf{DTIME}(n^{O(\log\log n)})$.*

PROOF. Assume $\mathbf{NP} \not\subseteq \mathbf{DTIME}(n^{O(\log\log n)})$. Let p (the norm parameter), $\epsilon > e^2$ be given, and let $t(n) = p(n)$. (Note that since $t(n)$ must be less than $\log_\epsilon n$, and $\epsilon > e^2$, we need the upper bound of $(\frac{1}{2} - \delta)\ln n$ on $p(n)$.) As a direct consequence of Lemma 6 and our complexity assumption, we know that for all efficient algorithm A, there is $n > 0$ such that there is an instance I of size n such that using $t(n) \cdot \#OPT(I)$ sets, A has at least $n \cdot \epsilon^{-t(n)}$ elements remaining.

Let A be any polynomial-time algorithm for solving L_p set cover. Fix n and such an instance I. Let $\underline{\mathsf{opt}}$ denote the L_p cost of any optimal algorithm on the instance I, and let $\underline{\mathsf{alg}}$ denote the L_p cost of the algorithm A. As before, let X_i denote the elements with cover index i and let R_i denote the elements with cover index i or greater A's solution, and let X_i^* and R_i^* denote the analogous sets for the optimal solution. We know that $\underline{\mathsf{opt}}^p = \sum_{i=1}^k i^p |X_i^*| \le n \cdot [\#OPT(I)]^p$, because the classical solution is also a solution of the L_p version. On the other hand, $\underline{\mathsf{alg}}^p \ge s^p \cdot |R_s|$ for all $s > 0$. In particular, with $s = p \cdot \#OPT(I)$ and our lower bound on $|R_s|$ from Lemma 6, we conclude $\underline{\mathsf{alg}}^p \ge (\#OPT(I) \cdot p)^p \cdot \frac{n}{\epsilon^p}$ Therefore, $\underline{\mathsf{alg}}/\underline{\mathsf{opt}} \ge \left((p/\epsilon)^p\right)^{1/p} = p/\epsilon = \Omega(p)$. ∎

Lemma 8 *For $p(n) = O(1)$, it is impossible to approximate L_p set cover better than $\Omega(p)$ unless $\mathbf{P} = \mathbf{NP}$.*

PROOF. Feige et al. [FLT04] shows that, for all $c_0, \epsilon > 0$, there are set cover instances such that it is **NP**-hard to distinguish between the following two cases: (**1**) There is a set cover of size t, or (**2**) For all integers x such that $1 \le x \le c_0 t$, every collection of x sets leaves at least a fraction of $(1 - 1/t)^x - \epsilon$ of the elements uncovered.

It follows that if we guess t, any algorithm leaving fewer than $((1 - 1/t)^x - \epsilon)\, n$ elements uncovered after buying x sets, for any $x \in [1, c_0 t]$, allows us to solve an **NP**-Complete problem. Thus unless $\mathbf{P} = \mathbf{NP}$, every polynomial time algorithm run on these instances has at least $((1 - 1/t)^x - \epsilon)\, n$ elements uncovered after buying x sets, for any $x \in [1, c_0 t]$.

Now fix p and a polynomial time algorithm A and let $\underline{\mathsf{alg}}^p$ be the p^{th} power its cost for the L_p set-cover problem. Let $\underline{\mathsf{opt}}^p$ denote the corresponding quantity for the optimal solution. Let $g(x) := x^p - (x-1)^p$. Recall $\underline{\mathsf{alg}}^p = \sum_x |R_x| \cdot g(x)$, where R_x is the set of elements with cover index at least x. Suppose that there is a set cover of size t. In that case

it is not too hard to show that $\underline{\text{opt}}^p \leq \sum_{x=1}^{t} \left(\frac{n}{t}\right) x^p$, since after buying x sets the optimal solution covers at least $\frac{n}{t}x$ elements. Thus $\underline{\text{opt}}^p \leq n \cdot t^p$. On the other hand:

$$\underline{\text{alg}}^p = \sum_{x \geq 1} |R_x| g(x) \geq \sum_{x=1}^{c_0 \cdot t} \left((1 - \frac{1}{t})^x - \epsilon \right) n \cdot g(x) \approx n \int_{x=1}^{c_0 \cdot t} \left(e^{-\frac{x}{t}} - \epsilon \right) g(x) dx$$

Note that $t = \omega(1)$, so $(1 - 1/t)^x \approx e^{-x/t}$ is an arbitrarily accurate approximation. If we can set $c_0 > (p+1)$ and $\epsilon \leq e^{-(p+1)}/2$ it is not too hard to show $\underline{\text{alg}}^p = \Omega(nt^p \left(\frac{p}{e}\right)^p)$, simply by considering the contribution of $\sum_{x=pt}^{(p+1)\cdot t} \left(e^{-x/t} - \epsilon\right) n \cdot g(x)$ to $\underline{\text{alg}}^p$. Thus $\underline{\text{alg}}^p / \underline{\text{opt}}^p = \Omega(\left(\frac{p}{e}\right)^p)$, and we obtain a gap of $\underline{\text{alg}}/\underline{\text{opt}} = \Omega(p)$ for all constant p. ∎

Combining Lemma 7 and Lemma 8 immediately yields the following theorem.

Theorem 9 *Unless* **NP** $\subseteq$ **DTIME**$(n^{O(\log \log n)})$, *for all* $\delta > 0$ *and* $p = p(n)$ *such that* $1 \leq p(n) \leq (\frac{1}{2} - \delta) \ln(n)$, *it is impossible to approximate* L_p *set cover better than* $\Omega(p)$.

2.3 Submodular Set Cover

We now consider a generalization of the L_p set cover problem. Our setting now assumes a (monotone) submodular function $f : 2^V \to \mathbb{R}_+$. Using techniques similar to those above, we can analyze the greedy algorithm's performance on this generalization, and obtain the same approximation guarantee. Thus, if action x_i takes c_i time to perform, and we perform actions $x_1, x_2, \ldots, x_k$ in that order, the total cost will be

$$\left(\sum_{i=1}^{k} \left(f(S_i) - f(S_{i-1}) \right) \cdot \left(\sum_{j=1}^{i} c_j \right)^p \right)^{1/p}$$

where $S_i := \{x_1, x_2, \ldots, x_i\}$. The objective is to select the permutation that minimizes this cost. The proof of the following theorem appears in the full version [GGKT07].

Theorem 10 (Submodular L_p Approximation Guarantee) *The greedy algorithm gives a* $(1 + p)^{1+1/p} \leq p + \ln p + 3$*-approximation for the submodular* L_p *set cover problem.*

2.4 The Pipelined Set Cover Problem

Closely related to the L_p set cover problem is the L_p pipelined set cover problem. In L_p-pipelined set cover, the cost function is given by:

$$\text{cost} = \left(\sum_{i \geq 0} c_i |R_i|^p \right)^{1/p}$$

This formulation follows [MBMW05] but incorporates the notion of cost for each set. [§] When $p = 1$, this cost function is the same that for the L_p case (and the min sum set cover problem). For this problem, we use the technique in the proof of Theorem 4 to argue that the greedy algorithm achieves the following approximation ratio; previous work [MBMW05] gave no approximation guarantee the general costs case. The proof is given in the full version.

Theorem 11 (Pipelined Set-Cover Approximation Guarantee) *The standard greedy algorithm gives a* $(1 + \frac{\ln p}{p} + \frac{3}{p})$*-approximation for the* L_p *pipelined set-cover problem.*

[§]This expression, in fact, differs from that defined by Munagala et al. [MBMW05]: their objective raises c_i to the p^{th} power. However, this only changes the quantity minimized in the greedy step, and hence we use this expression for convenience.

3 All L_p Norm Approximations via Sampling Norms

We now ask the following question: *Is there a small "basis" set of L_p norms that "approximate" all other L_p norms?* Formally, given two vectors X and Y of length n each, is there a set S of indices such that if $\|X\|_p \leq \|Y\|_p$ for all $p \in S$, then the same inequality holds (up to a constant approximation) for *all* L_p norms? Given such a set S, we can imagine finding a solution for each L_p with $p \in S$, and then "composing" them together to get solution that is good for *all* L_p norms. In this section, we will show that there is indeed such a set S of size $O(\log n)$; if we are interested in maintaining L_p norms only for *integer* p, then we can get a set of size $O(\sqrt{\log n})$. Moreover, we show that both these bounds are tight. Proofs omitted from this section appear in the full version [GGKT07].

Definition 12 (α-Sampling) *For a domain $D \subseteq \mathbb{R}_{\geq 1} \cup \{\infty\}$, a set $S \subseteq D$ is an* α-sampling of D of order n *if for all pairs of non-negative vectors $X, Y \in \mathbb{R}^n_{\geq 0}$*

$$\|X\|_p \leq \|Y\|_p \text{ for all } p \in S \quad \Rightarrow \quad \|X\|_p \leq \alpha \cdot \|Y\|_p \text{ for all } p \in D.$$

Such samplings prove useful in the All L_p Norm framework in the following way.

Theorem 13 *Given a minimization problem whose objective function is the L_p norm of some cost vector, and an α-sampling S of $D \subseteq \mathbb{R}_{\geq 1} \cup \{\infty\}$, then a cost vector $\mathbf{C}$ that is a simultaneous β-approximation for the class $\{L_p \mid p \in S\}$ is a simultaneous $\alpha\beta$-approximation for the class $\{L_p \mid p \in D\}$.*

We prove the following tight bounds on the size of $O(1)$-samplings.

Theorem 14 (Tight Bounds on $O(1)$-Samplings) *There exists an $O(1)$-sampling of the domain $D_{reals} = \mathbb{R}_{\geq 1} \cup \{\infty\}$ of order n with size $|S| = O(\log n)$, and an $O(1)$-sampling of the domain $D_{ints} = \mathbb{Z}_{\geq 1} \cup \{\infty\}$ of order n with size $O(\sqrt{\log n})$. Moreover, one cannot obtain smaller $O(1)$-samplings for either of these domains.*

3.1 All L_p Norm Approximations for Facility Location Problems

In this section, we show how the $O(1)$-samplings immediately give algorithms for the All L_p Norm k-facility location problems. As mentioned in the introduction, we can imagine an abstract facility location problem where given a metric space (V, d) with demand points $D \subseteq V$, we open a set of at most k facilities $F \subseteq V$ and assign each demand to a facility. This naturally gives a vector $\mathbf{C}$ of *assignment costs* for the demands with each solution: the k-median problem now minimizes $\|\mathbf{C}\|_1$, the k-means problem looks at $\|\mathbf{C}\|_2$, and the k-center problem at $\|\mathbf{C}\|_\infty$, etc. Let $\underline{\text{opt}}_p(k)$ denote a solution opening k facilities that minimizes the L_p norm of the vector of assignment costs. For any set of open facilities F, let $\mathsf{Cost}_p(F)$ denote the ℓ_p norm of the resulting vector of assignment costs. The following theorem shows how to get an All L_p Norm approximation to such problems.

Theorem 15 *There exists a set F of $O(k \log n)$ facilities F such that $\mathsf{Cost}_p(F) \leq O(1) \cdot \mathsf{Cost}_p(\underline{\text{opt}}_p(k))$ for all $p \geq 1$. If we want this to hold for all L_p norms for integer values of p only, then we need only $O(k\sqrt{\log n})$ facilities. Moreover, we can find these facilities in polynomial time in both cases.*

The proof is immediate from Theorems 13 and 14, and the fact that for any $1 \leq p < \infty$, one can use existing techniques to get an $O(1)$-approximation algorithm for minimizing the ℓ_p norm $\|\mathbf{C}\|_p$. Indeed, all the approximation algorithms for the k-median problem cited

above have the following additional property—if the underlying space only satisfies a λ-relaxed triangle-inequality (i.e., the distances satisfy $d(x,y) \leq \lambda \cdot (d(x,z) + d(y,z)$ for the parameter $\lambda \geq 1$), then these algorithms give an $O(\lambda)$-approximation algorithm for the k-median problem. The problem of minimizing the (p^{th} power of) the ℓ_p norm of assignment cost can be thought of as the k-median problem where distance between two points x and y is given by $d(x,y)^p$. Now these distances satisfy the $\lambda = 2^p$-relaxed triangle-inequality, and hence we get an $[O(2^p)]^{1/p}$-approximation algorithm for the ℓ_p norm.

Kumar and Kleinberg showed that we need to open $\Omega(k \log n)$ facilities to get an $O(1)$-AllNorm-approximation. That proof does not work for the All L_p Norm case; however, we can show the following result.

Theorem 16 *Given a parameter α, there exists a metric space over n demand points such that for a set of facilities F satisfying $\mathsf{Cost}_p(F) \leq \alpha \cdot \underline{\mathsf{opt}}_p(k)$ for all integer $p \geq 1$, $|F| \geq \Omega\big(k\big(\frac{\log n}{\log(\alpha k)}\big)^{\frac{1}{3}}\big)$. In fact, the lower bound holds even for L_p norms with integer p.*

It is an interesting open problem if we can open $o(k \log n)$ facilities and still be $O(1)$-competitive against all L_p norms.

4 AllNorm Approximation Algorithms

In the previous sections, we were interested in All L_p Norm approximations, and situations where focusing on L_p norms (instead of all symmetric norms) would give more nuanced results. In this section, we give results for the AllNorm case; complete proofs of the theorems in this section appear in the full version [GGKT07].

For a vector X, define $\overleftarrow{X}$ as the vector obtained by sorting the coordinates of X in descending order. Given vectors X and Y of length n each, we say that X is *α-submajorized* by Y (written as $X \prec_\alpha Y$) if for all $i \in [n]$, $\sum_{j \leq i} \overleftarrow{X}_j \leq \alpha \sum_{j \leq i} \overleftarrow{Y}_j$ (i.e., the partial sums of $\overleftarrow{X}$ are at most α times the partial sums in $\overleftarrow{Y}$). Intuitively, this means that the k unhappiest elements in X are together at most α times worse off than the k unhappiest elements of Y: we will want to find solutions X which are α-submajorized by *any other* solution Y (for small α). The following result is well-known (see, e.g., [Ste04]).

Theorem 17 *Let X and Y be two vectors of equal length, such that X is α-submajorized by Y. Then $f(X) \leq f(\alpha \cdot Y)$ for all real symmetric convex functions. In particular, if f is a symmetric norm, then $f(X) \leq \alpha f(Y)$.*

4.1 AllNorm Approximation from Partial Covering Algorithms

We now show how solutions for "partial covering" problems can be used to prove submajorization results; these submajorization results immediately lead to AllNorm approximations for these problems by Theorem 17. *Partial covering* problems include the k-MST problem (find a tree of minimum cost spanning at least k nodes), or the k-vertex cover problem (find a set of nodes of minimum size/cost that covers at least k edges). In this paper, we show how an $O(1)$-approximation to the k-MST problem implies an $O(1)$-submajorization result, and how these ideas extend to other partial cover problems.

Theorem 18 *For a TSP instance on a graph $G = (V, E)$, given a tour π, let t_i be the time at which the salesperson reaches vertex v_i, and let $T_\pi = (t_1, t_2, \ldots, t_n)$ be the vector of these* arrival times *sorted in ascending order. Then there is a solution where the arrival time vector is α-submajorized by the corresponding vector in any other solution, where $\alpha \leq 16$.*

The ideas behind this theorem can be used to show that Set Cover problem admits an $O(\log n)$-AllNorm approximation, Vertex Cover an 8-AllNorm approximation, etc. Let us sketch the idea for Vertex Cover: first use the fact that k-vertex cover admits a 2-approximation [BB98, Hoc98, BY01, GKS04]. This gives us an algorithm that given a budget B, finds a solution of cost $2B$ in poly-time which covers at least as many edges as any other solution of cost B. Setting the value of B to be successive powers of 2, we can argue that if any other algorithm covers k elements with cost at most 2^{i-1}, then we would have covered at least k elements with cost at most $4 \cdot 2^i$; this gives us an 8-submajorization. See the papers [GKS04, KPS06] for results on partial covering problems (all of which can be similarly extended).

4.2 AllNorm Algorithms for Flow Time on Parallel Machines

Finally, we consider the problem of scheduling jobs on parallel machines: given a schedule $\mathcal{A}$, the vector of interest is the vector $F^{\mathcal{A}}$ of *flow times*, where the flow time is the difference between its completion time and release date—hence, the ℓ_1 norm of this vector is the problem of minimizing the average flow time on parallel machines: see, [CKZ01] and the references therein for several polynomial-time logarithmic-approximation algorithms.

It is known that for any schedule $\mathcal{A}$, the All L_p Norm-guarantee $\alpha_{ALN}(F^{\mathcal{A}})$ is unbounded even if there is only one machine [BP04]: hence results have been given in the *resource augmentation* framework by giving our machines $(1+\varepsilon)$-speed. In particular, Bansal and Pruhs [BP04], and Chekuri et al. [CGKK04] gave results showing that given any constant $\varepsilon > 0$, we can get an $O(\frac{1}{\varepsilon^{O(1)}})$-approximation algorithm for all ℓ_p norms. In this paper, we show that one can extend their results to a submajorization, and hence AllNorm result.

Theorem 19 *There exists a schedule $\mathcal{A}$ such that $F^{\mathcal{A}}$ β-submajorizes $F^{\mathcal{B}}$ for all schedules $\mathcal{B}$, where β is a constant (depending only on ε).*

References

[AE05] Yossi Azar and Amir Epstein. Convex programming for scheduling unrelated parallel machines. In *STOC'05: Proceedings of the 37th Annual ACM Symposium on Theory of Computing*, pages 331–337, New York, 2005. ACM.

[AERW04] Yossi Azar, Leah Epstein, Yossi Richter, and Gerhard J. Woeginger. All-norm approximation algorithms. *J. Algorithms*, 52(2):120–133, 2004.

[AT04] Yossi Azar and Shai Taub. All-norm approximation for scheduling on identical machines. In *Algorithm theory—SWAT 2004*, volume 3111 of *Lecture Notes in Comput. Sci.*, pages 298–310. Springer, Berlin, 2004.

[BB98] Nader H. Bshouty and Lynn Burroughs. Massaging a linear programming solution to give a 2-approximation for a generalization of the vertex cover problem. In *STACS 98 (Paris, 1998)*, volume 1373 of *Lecture Notes in Comput. Sci.*, pages 298–308. Springer, Berlin, 1998.

[BCC+94] Avrim Blum, Prasad Chalasani, Don Coppersmith, Bill Pulleyblank, Prabhakar Raghavan, and Madhu Sudan. The minimum latency problem. In *Proceedings of the twenty-sixth annual ACM symposium on Theory of computing*, pages 163–171. ACM Press, 1994.

[BNBH+98] Amotz Bar-Noy, Mihir Bellare, Magnús M. Halldórsson, Hadas Shachnai, and Tami Tamir. On chromatic sums and distributed resource allocation. *Inform. and Comput.*, 140(2):183–202, 1998.

[BP03] Nikhil Bansal and Kirk Pruhs. Server scheduling in the L_p norm: a rising tide lifts all boat. In *Proceedings of the Thirty-Fifth Annual ACM Symposium on Theory of Computing*, pages 242–250, New York, 2003. ACM.

[BP04] Nikhil Bansal and Kirk Pruhs. Server scheduling in the weighted l_p norm. In *LATIN 2004: Theoretical informatics*, volume 2976 of *Lecture Notes in Comput. Sci.*, pages 434–443. Springer, Berlin, 2004.

[BY01] Reuven Bar-Yehuda. Using homogeneous weights for approximating the partial cover problem. *J. Algorithms*, 39(2):137–144, 2001.

[CFK03] Edith Cohen, Amos Fiat, and Haim Kaplan. Efficient sequences of trials. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms (Baltimore, MD, 2003)*, pages 737–746, New York, 2003. ACM.

[CGKK04] Chandra Chekuri, Ashish Goel, Sanjeev Khanna, and Amit Kumar. Multi-processor scheduling to minimize flow time with ϵ resource augmentation. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing*, pages 363–372, New York, 2004. ACM.

[Chv79] V. Chvátal. A greedy heuristic for the set-covering problem. *Math. Oper. Res.*, 4(3):233–235, 1979.

[CKZ01] Chandra Chekuri, Sanjeev Khanna, and An Zhu. Algorithms for minimizing weighted flow time. In *Proceedings of the Thirty-Third Annual ACM Symposium on Theory of Computing*, pages 84–93 (electronic), New York, 2001. ACM.

[Fei98] U. Feige. A threshold of $\ln n$ for approximating set cover. *J. ACM*, 45(4):634–652, 1998.

[FLT04] Uriel Feige, László Lovász, and Prasad Tetali. Approximating min sum set cover. *Algorithmica*, 40(4):219–234, 2004.

[GGKT07] Daniel Golovin, Anupam Gupta, Amit Kumar, and Kanat Tangwongsan. All-Norms and All-L_p-Norms approximation algorithms. Technical Report CMU-CS-07-153, School of Computer Science, Carnegie Mellon University, September 2007.

[GKS04] Rajiv Gandhi, Samir Khuller, and Aravind Srinivasan. Approximation algorithms for partial covering problems. *J. Algorithms*, 53(1):55–84, 2004.

[GM06] Ashish Goel and Adam Meyerson. Simultaneous optimization via approximate majorization for concave profits or convex costs. *Algorithmica*, 44(4):301–323, 2006.

[GMP01] Ashish Goel, Adam Meyerson, and Serge Plotkin. Combining fairness with throughput: online routing with multiple objectives. *J. Comput. System Sci.*, 63(1):62–79, 2001.

[HLP88] G. H. Hardy, J. E. Littlewood, and G. Pólya. *Inequalities*. Cambridge Mathematical Library. Cambridge University Press, Cambridge, 1988. Reprint of the 1952 edition.

[Hoc98] Dorit S. Hochbaum. The t-vertex cover problem: extending the half integrality framework with budget constraints. In *Approximation algorithms for combinatorial optimization (Aalborg, 1998)*, volume 1444 of *Lecture Notes in Comput. Sci.*, pages 111–122. Springer, Berlin, 1998.

[Joh74] David S. Johnson. Approximation algorithms for combinatorial problems. *J. Comput. System Sci.*, 9:256–278, 1974.

[KK00] Amit Kumar and Jon Kleinberg. Fairness measures for resource allocation. In *41st Annual Symposium on Foundations of Computer Science (Redondo Beach, CA, 2000)*, pages 75–85. IEEE Comput. Soc. Press, Los Alamitos, CA, 2000.

[KPS06] Jochen Könemann, Ojas Parekh, and Danny Segev. A unified approach to approximating partial covering problems. In *ESA'06: Proceedings of the 14th conference on Annual European Symposium*, pages 468–479, London, UK, 2006. Springer-Verlag.

[KRT01] Jon Kleinberg, Yuval Rabani, and Éva Tardos. Fairness in routing and load balancing. *J. Comput. System Sci.*, 63(1):2–20, 2001. Special issue on internet algorithms.

[Lov75] L. Lovász. On the ratio of optimal integral and fractional covers. *Discrete Math.*, 13(4):383–390, 1975.

[MBMW05] Kamesh Munagala, Shivnath Babu, Rajeev Motwani, and Jennifer Widom. The pipelined set cover problem. In *Database theory—ICDT 2005*, volume 3363 of *Lecture Notes in Comput. Sci.*, pages 83–98. Springer, Berlin, 2005.

[Sla97] Petr Slavík. A tight analysis of the greedy algorithm for set cover. *J. Algorithms*, 25(2):237–254, 1997.

[Sri99] Aravind Srinivasan. Improved approximation guarantees for packing and covering integer programs. *SIAM J. Comput.*, 29(2):648–670, 1999.

[Ste04] J. Michael Steele. *The Cauchy-Schwarz master class*. MAA Problem Books Series. Mathematical Association of America, Washington, DC, 2004. An introduction to the art of mathematical inequalities.

Foundations of Software Technology and Theoretical Computer Science (Bangalore) 2008.
Editors: R. Hariharan, M. Mukund, V. Vinay; pp 211-222

An Optimal Construction of Finite Automata from Regular Expressions

Stefan Gulan, Henning Fernau
Universität Trier
{gulan,fernau}@uni-trier.de

ABSTRACT. We consider the construction of finite automata from their corresponding regular expressions by a series of digraph-transformations along the expression's structure. Each intermediate graph represents an extended finite automaton accepting the same language. The character of our construction allows a fine-grained analysis of the emerging automaton's size, eventually leading to an optimality result.

1 Introduction

Regular expressions provide a description of regular languages in a manner convenient for the human reader. On the machine level, however, the most appropriate representation is arguably that of finite automata. Thus, considerable effort has been put into ways of constructing automata describing the same language as a given expression. All algorithms known to the authors work by either incorporating the expression's syntactic structure into the state graph of the emerging automaton [OF61, Kle65, Tho68, SSS88, IY03] or by looking for first-time occurrences of symbols in subexpressions [Glu61, MY60, BS86]. The first kind of construction generally results in an NFA with ϵ-transitions (ϵNFA, for short), the latter produces no such transitions and may even provide a DFA. An exhaustive overview is given in [Wat94].

Our construction yields an ϵNFA. No tight bound for the size of such an automaton representing a given expression has been published yet. Ilie & Yu [IY03] came pretty close, proving a lower bound of $\frac{4}{3}$ times the size of a given expression while constructing an ϵNFA smaller than $\frac{3}{2}$ times the expression length. We close this gap by raising the lower bound and giving a construction reaching that bound in the worst case. Note, however, that plenty of definitions of the sizes of automata and regular expressions are afloat, some of which are compared in [EKSW05]. For comparability, we stick by the definition given in [IY03].

The algorithm presented in this paper is basically an extension to the one given in [OF61], which is, together with a variation of Thompson's algorithm in [Wat94], the only top-down algorithm among a variety of bottom-up procedures. It turns out that the top-down character is very helpful in the analysis, since it allows systematic construction of an expression yielding the worst ratio of automaton-to-expression sizes. This construction relies on extremal combinatorial arguments for inferring structural properties of a worst-case input. To our knowledge this is a novel approach to this kind of problem.

FSTTCS 2008
IARCS Annual Conference on
Foundations of Software Technology and Theoretical Computer Science
http://drops.dagstuhl.de/opus/volltexte/2008/1754

2 Preliminaries

Enclosing braces for singleton sets will be omitted. Let $\mathcal{A}$ be a finite set of symbols, called *alphabet*, the elements of $\mathcal{A}\cup\epsilon$ will be called *literals*. The set of regular expressions over $\mathcal{A}$, denoted $Reg(\mathcal{A})$, is the closure of $\mathcal{A}\cup\epsilon$ under product $\bullet$, sum $+$ and Kleene-star *. Operator precedence is $*, \bullet, +$. We will casually speak of *expressions* only. In the following, α and β will always be expressions. The regular language expressed by α is denoted $L(\alpha)$. We will call α and β *equivalent*, denoted $\alpha\equiv\beta$, if $L(\alpha)=L(\beta)$. The number of products (sums, stars) in α will be denoted $|\alpha|_\bullet$ ($|\alpha|_+$, $|\alpha|_*$). Likewise, the number of literals in α, counted with multiplicity, will be denoted $|\alpha|_\mathcal{A}$. The *size* of an expression is defined as $|\alpha|:=|\alpha|_\bullet+|\alpha|_+|\alpha|_*+|\alpha|_\mathcal{A}$. We call α *complex*, if $|\alpha|\geq 2$. The set of subexpressions of α will be denoted $sub(\alpha)$.

Both iterated products and sums will be denoted as is common in arithmetic, defining

$$\prod_{i=1}^{n}\alpha_i := \alpha_1\bullet\alpha_2\bullet\ldots\bullet\alpha_n \quad \text{and} \quad \sum_{i=1}^{n}\alpha_i := \alpha_1+\alpha_2+\ldots+\alpha_n$$

Each α_i as above will be called an *operand* to the product or sum. An iterated product (sum) which is not operand to a product (sum) itself, will be called *maximal*. If all operands in a maximal product (sum) are starred, it will be called *star-maximal*.

An *extended finite automaton*, short *EFA*, is a 5-tuple $E=(Q,\mathcal{A},\delta,q_0,F)$, where $q_0\in Q$, $F\subseteq Q$, and $\delta\subset Q\times Reg(\mathcal{A})\times Q$. This renders conventional FAs a special case of EFAs. An EFA is called *normalized*, if $|F|=1$. A pair $(q,w)\in Q\times\mathcal{A}^*$ is called *configuration* of E, valid changes in E's configuration are denoted by $\vdash$, writing $(q,vw)\vdash(q',w)$ if $(q,\alpha,q')\in\delta$ and $v\in L(\alpha)$. The language accepted by E is $L(E)=\{w|(q_0,w)\vdash^*(q_f,\epsilon),q_f\in F\}$, where $\vdash^*$ is the reflexive-transitive closure of $\vdash$.

The class of regular languages is not extended by allowing regular expressions as labels in automata, see [Woo87] for a proper introduction. The size of an EFA E is $|E|:=|Q|+|\delta|$. The sets of transitions leaving and reaching some $q\in Q$ are given by $q^+:=\delta\cap(q\times Reg(\mathcal{A})\times Q)$ and $q^-:=\delta\cap(Q\times Reg(\mathcal{A})\times q)$, respectively. A set of transitions $\gamma=\{(q_i,\alpha_i,q_{i+1})|1\leq i\leq n-1\}\cup(q_n,\alpha_n,q_1)$ is called *cycle*.

Let A be a FA generated from α by some algorithm $\mathcal{C}$. We call $\frac{|A|}{|\alpha|}$ the *conversion-ratio* of $\mathcal{C}$ with respect to α. The maximal conversion-ratio of $\mathcal{C}$ with respect to any expression, will simply be called conversion-ratio of $\mathcal{C}$. An expression reaching this bound is said to be *worst-case*.

3 A Lower Bound

First we improve on a lower bound for *any* construction of FAs from expressions, given by Ilie & Yu in [IY03], by a slight variation of their argument. To this end, a property of digraphs is shown, in which we refer to both vertices and arcs as *elements*.

PROPOSITION 1. *Consider a digraph (V,A). Let L,R be nonempty, disjoint subsets of V such that*

1. *there is a path from each $l \in L$ to each $r \in R$,*
2. *there is no path connecting any two vertices $l, l' \in L$ or any $r, r' \in R$.*

Then at least $\min\{|L||R|, |L|+|R|+1\}$ *elements are necessary to realize these paths.*

PROOF. Two cases need to be considered:

1. There is no vertex on any path connecting l with r. This can only be realized with $|L||R|$ arcs, by pairwise connections.
2. There is at least one vertex b on a path connecting $l_b \in L$ with $r_b \in R$, this path contains at least 3 elements. To connect l_b with the vertices of $R \backslash r_b$ at least $|R|-1$ further arcs are necessary. An additional $|L|-1$ arcs are leaving the vertices of $L \backslash l_b$. These numbers total to $|L|+|R|+1$.

Next we show the actual lower bound. Both states and transitions of an FA A will be called elements, the number of elements is simply $|A|$.

THEOREM 2. *Let $x_{i,j}$ be distinct literals, consider the expression*

$$\begin{aligned} \alpha &= \prod_{i=1}^{n} (x^*_{2i-1,1} + x^*_{2i-1,2})(x^*_{2i,1} + x^*_{2i,2} + x^*_{2i,3}) \\ &= (x^*_{1,1}+x^*_{1,2})(x^*_{2,1}+x^*_{2,2}+x^*_{2,3}) \dots (x^*_{2n-1,1}+x^*_{2n-1,2})(x^*_{2n,1}+x^*_{2n,2}+x^*_{2n,3}) \end{aligned}$$

Any normalized automaton A satisfying $L(A) = L(\alpha)$ has at least size $22n+1$.

PROOF. In A, each $x_{i,j}$ is read on some cycle $\gamma_{i,j}$ comprising at least one transition incident to a state $q_{i,j}$, i.e., 2 elements. The $\gamma_{i,j}$ are disjoint, since literals of the same factor occur mutually exclusive and literals of different factors are ordered by α. Thus $5n$ cycles, accounting for at least $10n$ elements, are required. As for the connectivity of cycles, no path may lead from $\gamma_{i,j}$ to $\gamma_{i,k}$, if $j \neq k$, however, there need to be paths from $\gamma_{i,j}$ to $\gamma_{i+1,k}$. This carries over to the connectivity of the $q_{i,j}$, thus each two sets of states $q_{i,j}$ and $q_{i+1,j'}$ satisfy the conditions given in Prop. 1. Since one of the sets contains 2, the other one 3 states, by Prop. 1 at least 6 Elements are needed to ensure connectivity. As there are $2n-1$ such pairs, $12n-6$ elements are needed to connect them. This totals to $22n-6$ elements, additionally, 2 states and 5 transitions are necessary to ensure a normalized FA.
For the following, note that α from Thm. 2 has size $15n-1$.

COROLLARY 3. *The conversion-ratio of any algorithm converting expressions to normalized FAs is bounded from below by*

$$\frac{|A|}{|\alpha|} \geq \frac{22n+1}{15n-1} > \frac{22}{15} + \frac{1}{|\alpha|} = 1.4\bar{6} + \frac{1}{|\alpha|}$$

4 Construction

The idea is to expand an initial EFA according to the structure of the expression, by introducing as few states and transitions as possible, while decomposing transition labels. Certain substructures in the expanded automata will be replaced by smaller equivalents. This is done until an ϵNFA emerges, i.e., there are no more complex labels.

DEFINITION 4.*[Expansion] Let $E = (Q, \mathcal{A}, \delta, q_0, F)$ be an EFA with a complex labeled transition t. We call an EFA $E' = (Q', \mathcal{A}, \delta', q_0, F)$ the* expansion *of E, if it is derived from E according to the label of t as follows:*

- *if $t = (p, \alpha\beta, q)$ then $Q' = Q \dot\cup p'$, $\delta' = \delta \setminus t \cup \{(p, \alpha, p'), (p', \beta, q)\}$*
- *if $t = (p, \alpha + \beta, q)$ then $Q' = Q$, $\delta' = \delta \setminus t \cup \{(p, \alpha, q), (p, \beta, q)\}$*
- *if $t = (p, \alpha^*, q)$, we distinguish several cases*
  - $*0$*: if $p = q$, replace α^* with α,*
    let $Q' = Q$, $\delta' = \delta \setminus t \cup (q, \alpha, q)$
  - $*1$*: if $|p^+| = |q^-| = 1$, merge q into p:*
    let $Q' = Q \setminus q$, $\delta' = \delta \setminus (q^+ \cup q^-) \cup \{(p, \gamma, r) | (q, \gamma, r) \in \delta\} \cup (p, \alpha, p)$
  - $*2$*: if $|p^+| > 1$, $|q^-| = 1$, introduce a loop in q:*
    let $Q' = Q$, $\delta' = \delta \setminus t \cup \{(p, \epsilon, q), (q, \alpha, q)\}$
  - $*3$*: if $|p^+| = 1$, $|q^-| > 1$, introduce a loop in p:*
    let $Q' = Q$, $\delta' = \delta \setminus t \cup \{(p, \alpha, p), (p, \epsilon, q)\}$
  - $*4$*: if $|p^+| > 1$, $|q^-| > 1$, introduce a new state p':*
    let $Q' = Q \dot\cup p'$, $\delta' = \delta \setminus t \cup \{(p, \epsilon, p'), (p', \alpha, p'), (p', \epsilon, q)\}$

Cases are sketched in Fig. 1. Expansions will be denoted relational, writing $E \lhd_t E'$ if E' results from expansion of t in E. Occasionally we write $\lhd_\bullet, \lhd_+, \lhd_{*i}$ to indicate which case of Def. 4 is applied, or simply $E \lhd E'$, if both t and the case are irrelevant. The latter might be formalized as $\lhd = \lhd_\bullet \cup \lhd_+ \cup \bigcup_{0 \le i \le 4} \lhd_{*i}$. The n-fold iteration of $\lhd$ will be denoted $\lhd^n$, thus if $E \lhd^n E'$ there is a series of EFAs E_i, $0 \le i \le n$, such that $E = E_0$, $E_i \lhd E_{i+1}$, $E_n = E'$. Usually we refer to $\lhd_{(q,\alpha,q')}$ by mentioning α's operator, e.g, '$\bullet$-expansion'. Distinct $*$-expansions will be referred to as '$*0$-expansion' to '$*4$-expansion' according to Def. 4.

DEFINITION 5.*[Primal EFA] Let $\mathcal{A}$ be the least alphabet satisfying $\alpha \in Reg(\mathcal{A})$. The EFA $A^0_\alpha = (\{q_0, q_f\}, \mathcal{A}, (q_0, \alpha, q_f), q_0, q_f\})$ is called the* primal EFA *representing α. We denote by A^i_α any automaton satisfying $A^0_\alpha \lhd^i A^i_\alpha$.*

Thus, A^i_α denotes any EFA derived from the primal automaton representing α in a series of i expansions. Note that generally, A^i_α is not unique. However, a most useful property of $\lhd$ is that the order of expansion is irrelevant, or formally:

LEMMA 6. *$\lhd$ is locally confluent, i.e., if $A \lhd A'$ and $A \lhd A''$, then $\exists A''' : A' \lhd A'''$ and $A'' \lhd A'''$.*

PROOF. Given in the appendix.

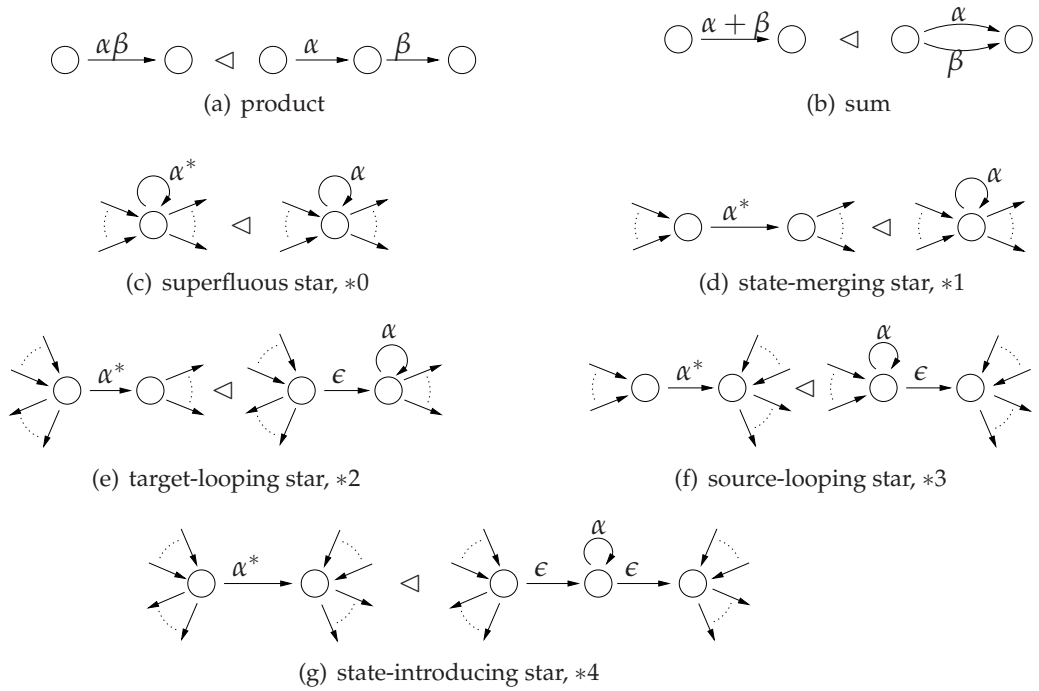
COROLLARY 7. *$\lhd$ is confluent.*

PROOF. Since $\lhd$ is terminating, the claim follows from Lem. 6. Detailed proof of this argument can be found, e.g., in [Hue80].

We introduce two further conversions of different nature, altering EFAs with respect to ϵ-labeled substructures.

DEFINITION 8.*[State-Elimination] Let $E{=}(Q, \mathcal{A}, \delta, q_0, F)$ be an EFA, $q \in Q \setminus F$. We consider two types of state-elimination, based on q^+ and q^-:*

- *Y-Type : $q^-{=}(p, \epsilon, q)$, $q^+{=}\{(q, \alpha_1, r_1), \ldots, (q, \alpha_n, r_n)\}$.*
  Then, let $\delta' = \delta \setminus (q^+ \cup q^-) \cup \{(p, \alpha_1, r_1), \ldots, (p, \alpha_n, r_n)\}$



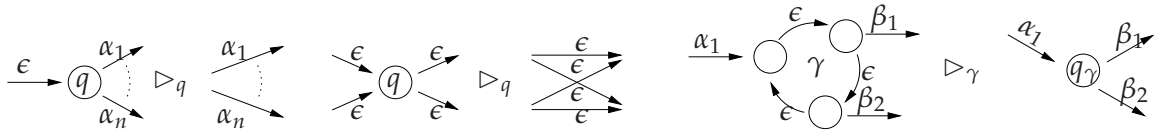
(a) product (b) sum

(c) superfluous star, $*0$ (d) state-merging star, $*1$

(e) target-looping star, $*2$ (f) source-looping star, $*3$

(g) state-introducing star, $*4$

Figure 1: Expansions of complex labeled transitions.



(a) Y-Type, schematic (b) X-Type, schematic (c) Cycle-elimination, exemplary

Figure 2: State-eliminations (a,b) and cycle-elimination (c)

- *X-Type :* $q^- = \{(p_1, \epsilon, q), (p_2, \epsilon, q)\}$, $q^+ = \{(q, \epsilon, r_1), (q, \epsilon, r_2)\}$.
  Then, let $\delta' = \delta \setminus (q^+ \cup q^-) \cup \{(p_1, \epsilon, r_1), (p_1, \epsilon, r_2), (p_2, \epsilon, r_1), (p_2, \epsilon, r_2)\}$.

The q-reduct *of* E *is defined as* $E' = (Q \setminus q, \mathcal{A}, \delta', q_0, F)$ *and we write* $E \rhd_q E'$.

By reverting the transitions for Y-Type elimination, a further rule—though not structurally different from the given Y-Type—is obtained.

DEFINITION 9. *[*ϵ*Cycle-Elimination] Let* $\gamma = \{(q_i, \epsilon, q_i') | 1 \leq i \leq n\}$ *be a cycle of* $E = (Q, \mathcal{A}, \delta, q_0, F)$. *Let* $Q' = Q \setminus \{q_1, \ldots, q_n\} \cup q_\gamma$ *and* $\delta' = \delta \setminus \gamma \cup \{(p, \alpha, q_\gamma) | (p, \alpha, q_i) \in \delta\} \cup \{(q_\gamma, \beta, r) | (q_i, \beta, r) \in \delta\}$. *The* γ-reduct *of* E *is defined as* $E = (Q', \mathcal{A}, \delta', q_0, F)$.

Note that both state- and cycle-eliminations strictly reduce the size of an EFA without re-introducing complex labels. Eliminations are illustrated in Fig.2.
Exhaustive application of expansions and eliminations to A_α^0 (or any EFA, for that matter) yields an ϵNFA. A primitive algorithm is given below.

Algorithm 1 RegEx $\to$ ϵNFA

$A \leftarrow A^0_\alpha$
while A is not an NFA **do**
  choose a complex-labeled transition t in A
  let $A \lhd_t A'$
  if $\lhd_t$ introduced some $e = (q, \epsilon, q')$ **then**
    if q can be eliminated **then**
      let $A' \rhd_q A''$
      $A' \leftarrow A''$
    if q' can be eliminated **then**
      let $A' \rhd_{q'} A''$
      $A' \leftarrow A''$
    if e is part of some ϵ-cycle γ **then**
      let $A' \rhd_\gamma A''$
      $A' \leftarrow A''$
  $A \leftarrow A'$
end while

	$\lhd_\bullet$	$\lhd_+$	$\lhd_{*0}$	$\lhd_{*1}$	$\lhd_{*2}$, $\lhd_{*3}$	$\lhd_{*4}$	$\rhd_\gamma$	$\rhd_q$
$\Delta(\lvert Q\rvert)$	1	0	0	-1	0	1	-($\lvert\gamma\rvert - 1$)	-1
$\Delta(\lvert\delta\rvert)$	1	1	0	0	1	2	-$\lvert\gamma\rvert$	-1 or 0

Table 1: Number of elements introduced (i.e., removed, if negative) upon expansion and elimination, broken down to states and transitions.

5 Analysis

Let A_α denote an ϵNFA constructed by our algorithm from A^0_α. We start by bounding $|A_\alpha|$ from above. To this end, we refine the definition of $|\alpha|_*$. Let $|\alpha|_{*i}$ denote the number of stars in α, that will be $*i$-expanded. Clearly, $|\alpha|_* = \sum_{0 \le i \le 4} |\alpha|_{*i}$.

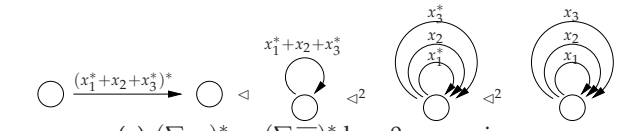
THEOREM 10. *The size of an automaton built from α by our algorithm is bounded by*

$$|A_\alpha| \le |\alpha| + 2|\alpha|_{*4} - |\alpha|_+ + 2$$

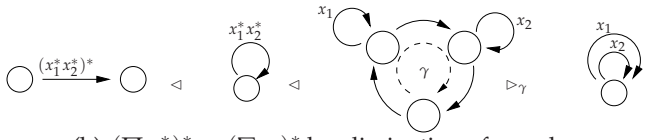
If this bound is tight then neither state-elimination nor $*0$, $*1$-expansion is applied.

PROOF. A^0_α is of size 3. The number of elements introduced upon expansion is determined by $|\alpha|_\bullet, |\alpha|_+, \ldots$, weighted by the entries in Tab. 1. Using $|\alpha|_\mathcal{A} = |\alpha|_\bullet + |\alpha|_+ + 1$ and $|\alpha| = |\alpha|_\bullet + |\alpha|_+ + |\alpha|_{*0} + \ldots + |\alpha|_{*4} + |\alpha|_\mathcal{A}$, this yields:

$$\begin{aligned} |A_\alpha| &\le 2|\alpha|_\bullet + |\alpha|_+ - |\alpha|_{*1} + |\alpha|_{*2,3} + 3|\alpha|_{*4} + 3 \\ &= |\alpha| + |\alpha|_\bullet - |\alpha|_{*0} - 2|\alpha|_{*1} + 2|\alpha|_{*4} - |\alpha|_\mathcal{A} + 3 \\ &\le |\alpha| + |\alpha|_\bullet + 2|\alpha|_{*4} - |\alpha|_\mathcal{A} + 3 \\ &= |\alpha| + 2|\alpha|_{*4} - |\alpha|_+ + 2 \end{aligned}$$



(a) $(\sum \alpha_i)^* \equiv (\sum \overline{\alpha_i})^*$ by $*0$-expansions.



(b) $(\prod \alpha_i^*)^* \equiv (\sum \alpha_i)^*$ by elimination of ϵ-cycles.

Figure 3: Transformations respect the equivalences given in Prop. 12 (ϵ-labels are omitted).

The first inequality results from state- and ϵ-cycle eliminations, the second from $*0$- and $*1$-expansions, thus equality holds in absence of these transformations.
The conversion ratio of a worst-case expression can be read immediately from this term; since we will refer to this quotient rather often, we restate it explicitly in

COROLLARY 11. *Let α be worst-case, then*

$$\frac{|A_\alpha|}{|\alpha|} = 1 + \frac{2|\alpha|_{*4} - |\alpha|_+ + 2}{|\alpha|}$$

PROPOSITION 12. *Both sides in each of the following equivalences will be expanded to the same (sub)automaton:*

$$(\alpha^*)^* \equiv \alpha^* \quad \textit{and} \quad (\sum \alpha_i)^* \equiv (\sum \overline{\alpha_i})^* \quad \textit{and} \quad (\prod \alpha_i^*)^* \equiv (\sum \alpha_i)^*$$

where $\overline{\alpha_i} = \beta_i$, if $\alpha_i = \beta_i^*$ and α_i otherwise.

PROOF. The first two equivalences are realized by $*0$-expansion, the third by ϵ-cycle-elimination. Examples are given in Fig. 3.

COROLLARY 13. *Let α be worst-case, then $|\alpha|_{*0}{=}|\alpha|_{*1}{=}0$, further both a sum with starred operands and a maximally starred product are not starred themselves.*

PROOF. By Prop. 12 we know that such sums and products would lead to $*0$-/$*1$-expansions and eliminations. Since for worst-case expressions equality in Thm. 10 holds and thus said conversions do not occur, the claim follows.
We proceed with a series of results, each putting additional constraints to the structure of a worst-case expression. Almost all proofs work by a line of argumentation that is common in extremal combinatorics: assume α is worst-case, i.e., extremal with respect to conversion-ratio, then infer some further property by contradicting extremality of α.

PROPOSITION 14. *A worst-case expression contains stars.*

PROOF. Let α be worst-case with $|\alpha|_*{=}0$. Cor. 11 implies $\frac{|A_\alpha|}{|\alpha|} \leq 1 + \frac{2}{|\alpha|}$, the right-hand side of which drops below 1.4, if $|\alpha| \geq 5$. Since by Cor. 3, the conversion-ratio is bounded from below by $1.4\bar{6}$, the assumption $|\alpha|_*{=}0$ is wrong, if α is worst-case.

LEMMA 15. *Let γ^* be a proper subexpression of α. Then γ^* will be $*4$-expanded iff*
- *it is operand to a sum which is not starred, or*
- *without loss of generality it occurs rightmost in a star-maximal product.*

PROOF. The first case is clear by looking at the expansion of some $\gamma^*+\beta$: If a transition labeled like this is a loop, γ^* will be $*0$-expanded, otherwise it will definitely be $*4$-expanded. The second case is more involved: If γ^* is an infix, say, $\alpha_1\gamma^*\alpha_2$, we distinguish 3 cases: If both α_i are non-starred, γ^* will be $*1$-expanded. If only one of the α_i is non-starred, then γ^* can be $*2$- or $*3$-expanded by introducing a loop at the state incident to the transition labeled with the non-starred α_i. Finally, if both α_i are starred, we can by confluence assume that expansions will be applied from left to right. Then, every starred factor will be $*2$-expanded until the final one necessitates $*4$-expansion. This embraces all possible cases, giving both directions of the statement.

LEMMA 16. *Let α be worst-case, assume $\gamma^*\in sub(\alpha)$ is $*4$-expanded. Then γ^* is operand to a sum.*

PROOF. By Lem. 15, γ^* is either operand to a sum or rightmost in a star-maximal product. Assume the latter, thus $\pi = \pi_1^* \bullet \ldots \bullet \pi_{n-1}^* \bullet \gamma^*$. Construct α' from α by replacing π with $\sigma = \pi_1^* + \ldots + \pi_{n-1}^* + \gamma^*$. Then $|\alpha|=|\alpha'|$, however $2|\alpha'|_{*4} - |\alpha'|_+ = 2|\alpha|_{*4} - |\alpha|_+ + n - 1$. Since by Prop. 12 π is not starred in α, the stars in σ will not accidentally become $*0$. By Cor. 11, $\frac{|A_{\alpha'}|}{|\alpha'|} > \frac{|A_\alpha|}{|\alpha|}$, thus α is not worst-case. Therefore γ^* is necessarily operand to a sum.
The interrelation between sums and stars in a worst-case expression is further tightened in the following

LEMMA 17. *Let α be worst-case. Then*
1. *every starred subexpression in α is operand to a sum and*
2. *all operands in a maximal sum are starred.*

PROOF.
1. Assume $\gamma^*\in sub(\alpha)$ will not be $*4$-expanded. Construct α' from α by replacing γ^* with γ. Since $|\alpha'|=|\alpha|-1$, yet $|\alpha'|_{*4}=|\alpha|_{*4}$, Cor. 11 again yields $\frac{|A_{\alpha'}|}{|\alpha'|} > \frac{|A_\alpha|}{|\alpha|}$, thus α is not worst-case. Therefore each star in a worst-case expression is subject to $*4$-expansion, thus by Lem. 16 operand to a sum.
2. Let $\sum\sigma_i$ be maximal with some σ_j unstarred, i.e., a product. Construct α' from α by replacing σ_j with σ_j^*. This newly starred expressions will be $*4$-expanded (Lem. 15). Then $|\alpha'| = |\alpha|+1$, $|\alpha'|_{*4} = |\alpha|_{*4}+1$ and by Cor. 11, $|A_{\alpha'}| = |A_\alpha|+2$. Now
$$\frac{|A_{\alpha'}|}{|\alpha'|} = \frac{|A_\alpha|+2}{|\alpha|+1} > \frac{|A_\alpha|}{|\alpha|} \quad \text{iff} \quad |A_\alpha| < 2|\alpha|$$
We proceed similar to the proof of Thm. 10, additionally using that the previous item implies $|\alpha|_{*4} \leq 2|\alpha|_+$:
$$\begin{aligned} |A_\alpha| &\leq 2|\alpha|_\bullet + |\alpha|_+ - |\alpha|_{*1} + |\alpha|_{*2,3} + 3|\alpha|_{*4} + 3 \\ &= 2|\alpha| - |\alpha|_+ - 3|\alpha|_{*1} - |\alpha|_{*2,3} + |\alpha|_{*4} + 3 - 2|\alpha|_{\mathcal{A}} \\ &= 2|\alpha| - 2|\alpha|_+ - |\alpha|_\bullet - 3|\alpha|_{*1} - |\alpha|_{*2,3} + |\alpha|_{*4} + 2 - |\alpha|_{\mathcal{A}} \\ &\leq 2|\alpha| - |\alpha|_+ - 2|\alpha|_\bullet + 1 \end{aligned}$$

By assumption, $|\alpha|_+ \geq 1$, any further binary operator pushes the right-hand side strictly below $2|\alpha'|$. Indeed, the only expression containing only one $+$ as binary operator, that reaches a conversion-ratio of 2, is $x_1^* + x_2^*$, which is of claimed structure.

LEMMA 18. *A worst-case expression α has no subexpression of the form*

$$\phi = (\prod_i \sum_j \sigma_{ij}^*)^*$$

PROOF. If $\phi \in sub(\alpha)$, ϵ-cycle elimination would occur upon expansion. By Cor. 11 then α would not be worst-case.
This allows us to provide a pretty detailed template of a worst-case expression:

LEMMA 19. *Let α be worst-case. Then the structure of α is*

$$\alpha = \prod_{i=1}^{n} \sum_{j=1}^{k_i} \sigma_{ij}^* \quad \textit{where} \quad \sigma_{i,j} \in \mathcal{A}$$

PROOF. By Prop. 14, a worst-case expression contains starred subexpressions, so fix some σ_{ij}^* which is by Lem. 17 operand to a sum. A maximal sum with stars is a factor, since it may not be starred itself and is already maximal. Further, σ_{ij} is necessarily a maximal product. If its operands were maximally starred sums, this would contradict Lem. 18, thus σ_{ij} is a product of literals. Then, σ_{ij} influences the conversion-ratio as given in Cor. 11 only by its length, which has to be minimized in order to maximize the ratio. Thus σ_{ij} is a symbol from the alphabet. From Lem. 18 it also follows that α itself may not be starred.
It remains to analyze the influence of the number of summands (the k_i in Lem. 19) on conversion-ratio. This is done in the proof of our main

THEOREM 20. *An expression α is worst-case, if its structure is*

$$\alpha = \prod_{i=1}^{n} \sum_{j=1}^{2+(i \bmod 2)} x_{ij}^* \quad \textit{where} \quad x_{ij} \in \mathcal{A}$$

PROOF. Let α be of the general structure given in Lem. 19, the FA produced by a series of expansions from A_α^0 is sketched in Fig. 4. The sizes of these objects are

$$\begin{aligned} |\alpha| &= (n-1) + \sum_{i=1}^{n} (3k_i - 1) = 3\sum_{i=1}^{n} k_i - 1 \\ |A_\alpha| &= \sum_{i=1}^{n} 4k_i + n - 1 = 4\sum_{i=1}^{n} k_i + n - 1 \end{aligned}$$

thus the ratio is

$$\frac{|A_\alpha|}{|\alpha|} = \frac{4\sum k_i + n - 1}{3\sum k_i - 1} = 1 + \frac{\sum k_i + n}{3\sum k_i - 1}$$

The fraction on the right-hand side is maximized, if n is maximal with respect to $\sum k_i$, or equivalently, if $\sum k_i$ is minimal. Two restrictions result from prohibiting state-elimination, namely that $\forall i : k_i \geq 2$ and if k_i=2 then k_{i-1}>2 and k_{i+1}>2 (if they exist). Thus $\sum k_i$ is minimal, if k_i alternates between 2 and 3, i.e., $k_i = 2 + (i \bmod 2)$.

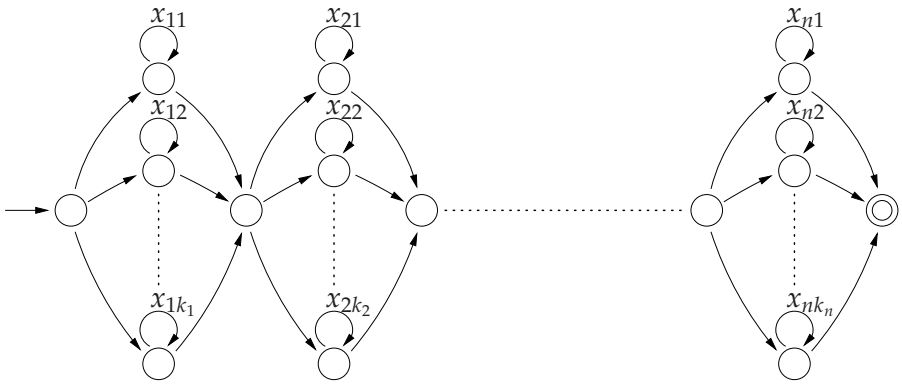


Figure 4: Automaton constructed from an expression as given in Lem. 19 (ϵ-labels are omitted).

COROLLARY 21. *The size of an automaton produced by our construction is bounded by* $\frac{22}{15}|\alpha| + 1$. *The construction is optimal.*

PROOF. The value is reached by the expression given in Thm. 20, which was proven to give the maximal ratio of sizes. Since by Cor. 3 $\frac{22}{15}|\alpha| + 1$ is also a lower bound, the bound is tight, hence the construction is optimal.

6 Conclusions & Remarks

We have given a construction for converting regular expressions into equivalent ϵNFAs. To our knowledge it is the only provably optimal construction so far. It should be mentioned that the generated automata differ from these constructed in [IY03] only by the effects of state-elimination. This element is crucial however, both for raising the lower bound as well as for upper bound analysis as we did. On a practical detail, preprocessing the input to *reduced* expressions (as done in [IY03]) is in part realized upon execution of our algorithm.

Treatment of $\emptyset$ in expressions can easily be added to our algorithm by considering it a literal throughout the expansion/reduction-sequence and adding a final step: removing $\emptyset$-labeled transitions followed by running some reachability algorithm. The final step will reduce the size of the automaton, thus the bound is maintained even if $\emptyset$ does not count into the expressions' size. Since we consider $\emptyset$ as being of no practical relevance, it was omitted from formal treatment.
Maybe more interesting, Kleene-$+$ can be implemented by reformulating $*$-expansions, where additional ϵ-transitions need to be introduced. This yields smaller FAs than by applying the equivalence $\alpha^+ \equiv \alpha\alpha^*$ (which would double the number of elements introduced by α), yet it is not feasible with the given bound.

Finally note that the construction not unique in the general case, since state-eliminations is not confluent. This can be remedied by adding rules that take the in- and out-degrees of the states adjacent to the eliminated one into consideration, however this is not at the attention of this paper. A closer analysis will be available in a future article.

References

[BS86] Gerard Berry and Ravi Sethi. From regular expressions to deterministic automata. *Theoretical Computer Science*, 48:117–126, 1986.

[EKSW05] Keith Ellul, Bryan Krawetz, Jeffrey Shallit, and Ming-Wei Wang. Regular expressions: new results and open problems. *Journal of Automata, Languages and Combinatorics*, 10(4):407–437, 2005.

[Glu61] Victor Michailowitsch Glushkov. The abstract theory of automata. *Russian Mathematical Surveys*, 16:1–53, 1961.

[Hue80] Gérard Huet. Confluent reductions: Abstract properties and applications to term rewriting systems. *Journal of the ACM*, 27(4):797–821, 1980.

[IY03] Lucian Ilie and Sheng Yu. Follow automata. *Information and Computation*, (186):140–162, 2003.

[Kle65] Stephen Cole Kleene. *Representation of Events in Nerve Nets and Finite Automata*, pages 3–41. Annals of Mathematics Studies. 1965.

[MY60] Robert McNaughton and Hisao Yamada. Regular expressions and state graphs for automata. *IRE Transactions on Electronic Computers*, 9(1):39–47, 1960.

[OF61] Gene Ott and Neil H. Feinstein. Design of sequential machines from their regular expressions. *Journal of the ACM*, 8(4):585–600, 1961.

[SSS88] Seppo Sippu and Eljas Soisalin-Soininen. *Parsing Theory*. EATCS Monographs on Theoretical Computer Science. Springer, 1988.

[Tho68] Ken Thompson. Regular expression search algorithm. *Communications of the ACM*, 11(6):419–422, 1968.

[Wat94] Bruce W. Watson. A taxonomy of finite automata construction algorithms. Technical Report Computing Science Note 93/43, Eindhoven University of Technology, may 1994.

[Woo87] Derick Wood. *Theory of Computation*. John Wiley & Sons, Inc., 1987.

A Appendix

LEMMA 6. $\lhd$ is locally confluent modulo isomorphism.

PROOF. First, assume one of the transitions is labeled by either a product or a sum:

- Let $t_1 = (q, \alpha \bullet \beta, q')$. Upon expansion a bridge-state q'' will be introduced, however the number of arcs leaving and reaching q and q' will remain constant. The structure of A will change insofar as that an arc will be elongated. Since any $\lhd_{t_2}$ will at most have the effect on t_1 that one of its states might be renamed (upon $*1$-expansion), the order of $\lhd_{t_1}, \lhd_{t_2}$ is irrelevant.
- If $t_1{=}(q, \alpha + \beta, q')$, informal reasoning is that an arc is merely doubled. Looking at Def. 4, the booleans $q^+{>}1$ etc. are not changed by such an operation.

Now let both t_i be star-labeled. Note that the statement is trivial, if expansions take place in 'different parts' of the EFA, so let t_1, t_2 share at least a common state. If the transitions are parallel, both will be $*4$-expanded anyway. Further, $*0$-expansion does not change the structure of the state-graph at all, i.e., neither of t_1, t_2 is a loop. So assume $t_1 = (p, \alpha^*, q)$, $t_2 = (q, \alpha^*, r)$ where $p \neq q \neq r$. Some of the possible combinations are shown in Fig. 5, the remaining are a simple exercise.

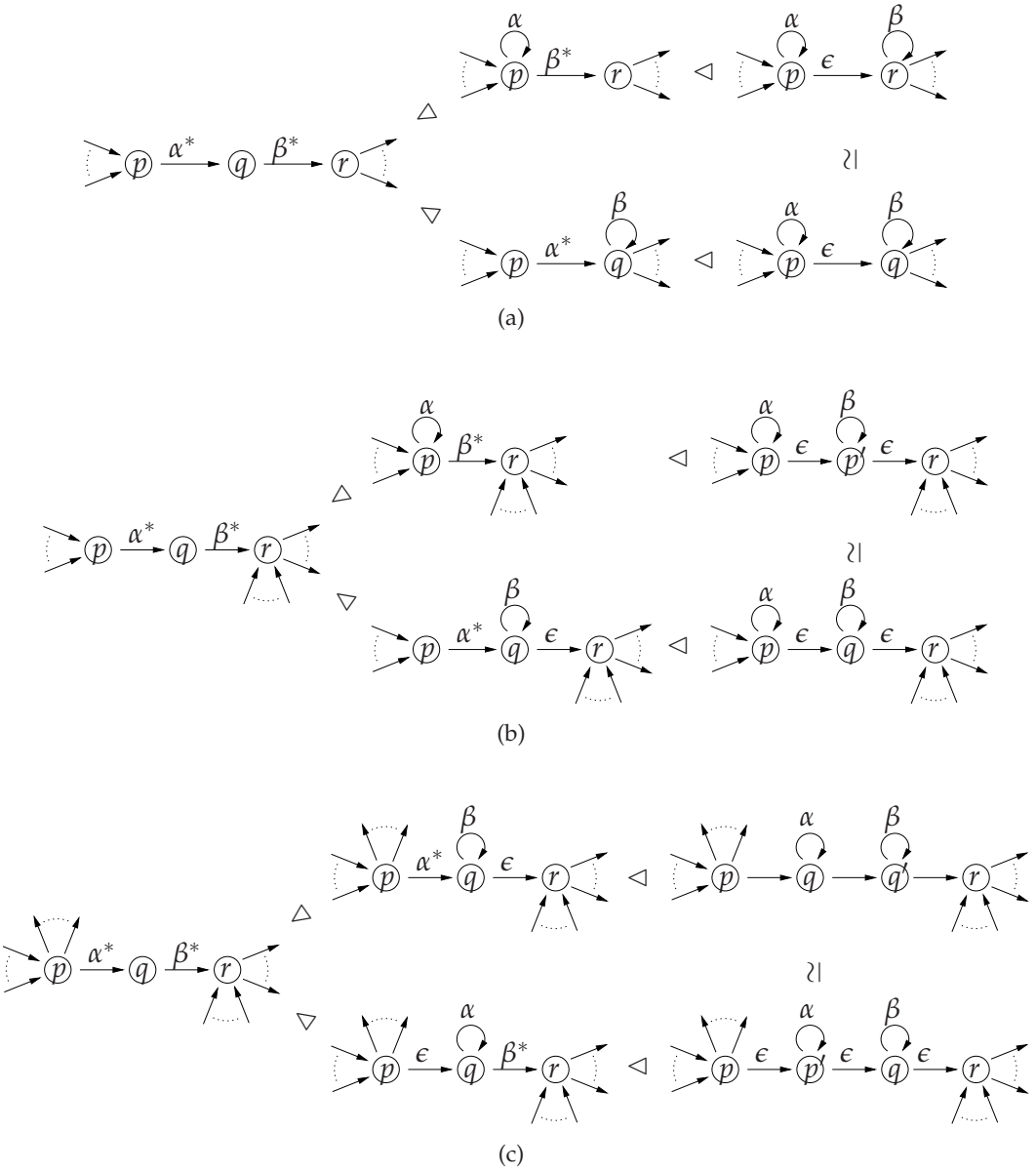


Figure 5: Examples for confluence of expanding consecutive starred transitions. Isomorphism is denoted by $\simeq$.

Foundations of Software Technology and Theoretical Computer Science (Bangalore) 2008.
Editors: R. Hariharan, M. Mukund, V. Vinay; pp 223-234

The unfolding of general Petri nets*

Jonathan Hayman and Glynn Winskel
Computer Laboratory, University of Cambridge, England

ABSTRACT. The unfolding of (1-)safe Petri nets to occurrence nets is well understood. There is a universal characterization of the unfolding of a safe net which is part and parcel of a coreflection from the category of occurrence nets to the category of safe nets. The unfolding of general Petri nets, nets with multiplicities on arcs whose markings are multisets of places, does not possess a directly analogous universal characterization, essentially because there is an implicit symmetry in the multiplicities of general nets, and that symmetry is not expressed in their traditional occurrence net unfoldings. In the present paper, we show how to recover a universal characterization by representing the symmetry in the behaviour of the occurrence net unfoldings of general Petri nets. We show that this is part of a coreflection between enriched categories of general Petri nets with symmetry and occurrence nets with symmetry.

1 Introduction

There is a wide array of models for concurrency. In [16], it is shown how category theory can be applied to describe the relationships between them by establishing adjunctions between their categories; the adjunctions often take the form of coreflections. This leads to uniform ways of defining constructions on models and provides links between concepts such as bisimulation in the models [5].

Only partial results have been achieved in relating Petri nets to other models for concurrency since, in general, there is no coreflection between occurrence nets and more general forms of net that allow transitions to deposit more than one token in any place or in which a place can initially hold more than one token. The reason for this, as we shall see, is that the operation of unfolding such a net to form its associated occurrence net does not account for the *symmetry* in the behaviour of the original net due to places being marked more than once. In this paper, we define the symmetry in the unfolding and use this to obtain a coreflection between general nets and occurrence nets *up to symmetry*.

Of course, there are undoubtedly several ways of adjoining symmetry to nets. The method we use was motivated by the need to extend the expressive power of event structures and the maps between them [14, 15]. Roughly, a symmetry on a Petri net is described as a relation between its runs as causal nets, the relation specifying when one run is similar to another up to symmetry; of course, if runs are to be similar then they should have similar futures as well as pasts. Technically and generally, a relation of symmetry is expressed as a span of open maps which form a pseudo equivalence.

This general algebraic method of adjoining symmetry is adopted to define symmetry in (the paths of) nets, which we use to relate the categories of general nets with symmetry and occurrence nets with symmetry. Another motivation for this work is that Petri nets provide a useful testing ground for the general method of adjoining symmetries. For example, the

*An extended version is available as a Computer Laboratory Technical Report.

FSTTCS 2008
IARCS Annual Conference on
Foundations of Software Technology and Theoretical Computer Science
http://drops.dagstuhl.de/opus/volltexte/2008/1755

present work has led us to drop the constraint in [14, 15] that the morphisms of the span should be jointly monic, in which case the span would be an equivalence rather than a pseudo equivalence. (A similar issue is encountered in the slightly simpler setting of nets without multiplicities [4].) Motivated by the categories of nets encountered, the method for adjoining symmetry is also extended to deal with more general forms of model such as those without all pullbacks.

2 Varieties of Petri nets

We begin by introducing Petri nets. It is unfortunately beyond the scope of the current paper to give anything but the essential definitions of the forms of net that we shall consider; we instead refer the reader to [9, 16] for a fuller introduction.

DEFINITION 1. *A general Petri net is a 5-tuple,*

$$G = (P, T, Pre, Post, \mathbb{M}),$$

comprising a set P of places *(or* conditions*); a set T of* transitions *(or* events*) disjoint from P; a* pre-place *multirelation, $Pre \subseteq_\mu T \times P$; a* post-place *$\infty$-multirelation, $Post \subseteq_{\mu_\infty} T \times P$; and a set $\mathbb{M}$ of ∞-multisets of P forming the set of* initial markings *of G. Every transition must consume at least one token:*

$$\forall t \in T\ \exists p \in P.\ Pre[t, p] > 0.$$

This is a mild generalization of the standard definition of Petri net in that we allow there to be a *set* of initial markings rather than just one initial marking, and will prove important later. In the case where a general net has precisely one initial marking, we say that the net is *singly-marked*.

A morphism of general nets embeds the structure of one net into that of another in way that preserves the token game for nets — see [13].

DEFINITION 2. *Let $G = (P, T, Pre, Post, \mathbb{M})$ and $G' = (P', T', Pre', Post', \mathbb{M}')$ be general Petri nets. A morphism $(\eta, \beta) : G \to G'$ is a pair consisting of a partial function $\eta : T \to_* T'$ and an ∞-multirelation $\beta \subseteq_{\mu_\infty} P \times P'$ which jointly satisfy:*

- *for all $M \in \mathbb{M}$: $\beta \cdot M \in \mathbb{M}'$*
- *for all $t \in T$: $\beta \cdot (Pre \cdot t) = Pre' \cdot \eta(t)$ and $\beta \cdot (Post \cdot t) = Post' \cdot \eta(t)$*

We write $\eta(t) = *$ if $\eta(t)$ is undefined and in the above requirement regard $*$ as the empty multiset, so that if $\eta(t) = *$ then $\beta \cdot (Pre \cdot t)$ and $\beta \cdot (Post \cdot t)$ are both empty.

The category of general Petri nets with multiple initial markings is denoted $\mathbf{Gen}^\sharp$, and we denote by **Gen** the category of singly-marked general nets (nets with one initial marking).

One simplification of general nets is to require that multirelations *Pre* and *Post* are *relations* rather than (∞)-multirelations and that every initial marking must be a *set* of places rather than an ∞-multiset. We shall call such nets *P/T nets*. The relations *Pre* and *Post* of a

P/T net may equivalently be seen as a *flow* relation $F \subseteq (P \times T) \cup (P \times T)$ describing how places and transitions are connected:

$$p\ F\ t \stackrel{\triangle}{\iff} Pre(p,t) \qquad t\ F\ p \stackrel{\triangle}{\iff} Post(t,p).$$

Any P/T net can therefore be defined as a 4-tuple $G = (P, T, F, \mathbb{M})$ by giving its flow relation. An important property that a P/T net can possess is (1-)*safety*, which means that any reachable marking is a set (*i.e.* there is no reachable marking that has more than one token in any place) — we say that a marking is *reachable* if it can be reached by any sequence of transitions from any initial marking according to the standard token game for nets.

Safe nets can be refined further to obtain *occurrence nets*.

DEFINITION 3. *An* occurrence net $O = (B, E, F, \mathbb{M})$ *is a safe net satisfying the following restrictions:*

1. $\forall M \in \mathbb{M} : \forall b \in M : (Pre \cdot b = \emptyset)$
2. $\forall b' \in B : \exists M \in \mathbb{M} : \exists b \in M : (b\ F^*\ b')$
3. $\forall b \in B : (|Pre \cdot b| \leq 1)$
4. F^+ *is irreflexive and, for all* $e \in E$, *the set* $\{e' \mid e'\ F^*\ e\}$ *is finite*
5. $\#$ *is irreflexive, where*

$$\begin{aligned} e\#_{\mathrm{m}}e' &\iff e \in E\ \&\ e' \in E\ \&\ e \neq e'\ \&\ Pre \cdot e \cap Pre \cdot e' \neq \emptyset \\ b\#_{\mathrm{m}}b' &\iff \exists M, M' \in \mathbb{M} : (M \neq M'\ \&\ b \in M\ \&\ b' \in M') \\ x\#x' &\iff \exists y, y' \in E \cup B : y\#_{\mathrm{m}}y'\ \&\ y\ F^*\ x\ \&\ y'\ F^*\ x' \end{aligned}$$

Singly-marked occurrence nets can be seen to coincide with the original definition of occurrence net [8].

By ensuring that any condition occurs as the postcondition of at most one event, the constraints above allow the flow relation F to be seen to represent causal dependency. Since the flow relation is required to be irreflexive, as is the *conflict* relation $\#$, every condition can occur in some reachable marking and every event can take place in some reachable marking. Two elements of the occurrence net are in conflict if the occurrence of one precludes the occurrence of the other at any later stage.

The *concurrency* relation $\mathrm{co}_O \subseteq (B \cup E) \times (B \cup E)$, indicating that two elements of the occurrence net are concurrent (may occur at the same time in some reachable marking) if they neither causally depend on nor conflict with each other, is defined as:

$$x\ \mathrm{co}_O\ y \stackrel{\triangle}{\iff} \neg(x\#y \text{ or } x\ F^+\ y \text{ or } y\ F^+\ x)$$

We often drop the subscript O and write co for the relation. The concurrency relation is extended to sets of conditions A in the following manner:

$$\mathrm{co}\ A \stackrel{\triangle}{\iff} (\forall b, b' \in A :\ b\ \mathrm{co}\ b') \text{ and } \{e \in E \mid \exists b \in A . e\ F^*\ b\} \text{ is finite}$$

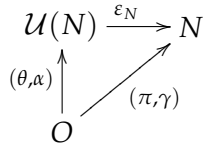
The final class of net that we shall make use of is *causal* nets. These are well-known representations of paths of general nets, recording how a set of consistent events (events that do not conflict) causally depend on each other through the encountered markings of conditions.

DEFINITION 4. *A causal net $C = (B, E, F, \mathbb{M})$ is an occurrence net with at most one initial marking for which the conflict relation # is empty.*

2.1 Unfolding

Occurrence nets can be used to give the semantics of more general forms of net. The process of forming the occurrence net semantics of a net is called *unfolding*, first defined for safe nets in [8]. The result of unfolding a net G is an occurrence net $\mathcal{U}(G)$ accompanied by a morphism $\varepsilon_G : \mathcal{U}(G) \to G$ relating the unfolding back to the original net.

For a safe net N, we are able to say that the occurrence net $\mathcal{U}(N)$ and morphism $\varepsilon_N : \mathcal{U}(N) \to N$ are *cofree*. That is, for any occurrence net O and morphism $(\pi, \gamma) : O \to N$, there is a *unique* morphism $(\theta, \alpha) : O \to \mathcal{U}(N)$ such that the following triangle commutes:



This result, first shown in [12] (for singly-marked nets; the generalization to multiply-marked nets is straightforward), ensures that $\mathbf{Occ}^\sharp$ is a coreflective subcategory of the category of safe nets, the operation of unfolding giving rise to a functor that is right-adjoint to the obvious inclusion functor. In fact, the result also applies to give a coreflection between occurrence nets and P/T nets and, more generally still, to give a coreflection between occurrence nets and nets with single multiplicity in the post-places of each transition and that have at most one token in each place in their initial markings, as shown in [6].

A coreflection is not, however, obtained when we consider the unfoldings of arbitrary general nets (either singly- or multiply-marked). The problem does not lie in defining the unfolding of general nets, which is characterized as follows:

PROPOSITION 5. *The unfolding $\mathcal{U}(G) = (B, E, F, \mathbb{M}_0)$ of $G = (P, T, Pre, Post, \mathbb{M})$ is the unique occurrence net to satisfy*

$$\begin{aligned}
B &= \{(M, p, i) \mid M \in \mathbb{M} \,\&\, p \in P \,\&\, 0 \leq i < M[p]\} \\
&\quad \cup \{(\{e\}, p, i) \mid e \in E \,\&\, p \in P \,\&\, 0 \leq i < (Post \cdot \eta(e))[p]\} \\
E &= \{(A, t) \mid A \subseteq B \,\&\, t \in T \,\&\, \mathrm{co}\ A \,\&\, \beta \cdot A = Pre \cdot t\} \\
&\quad b\ F\ (A, t) \iff b \in A \\
&\quad (A, t)\ F\ b \iff \exists p, i : (b = (\{(A, t)\}, p, i)) \\
\mathbb{M}_0 &= \{\{(M, p, i) \mid (M, p, i) \in B\} \mid M \in \mathbb{M}\},
\end{aligned}$$

where co *and # are the concurrency and conflict relations arising from F on B and E. Furthermore, $\eta : E \to P$ defined as $\eta(A, t) = t$ and $\beta : B \to P$ defined as $\beta(X, p, i) = p$ form a morphism $\varepsilon_G = (\eta, \beta) : \mathcal{U}(G) \to G$ in $\mathbf{Gen}^\sharp$, regarding the function β as a multirelation.*

The reason why we do not obtain a coreflection between the categories $\mathbf{Occ}^\sharp$ and $\mathbf{Gen}^\sharp$ (or **Occ** and **Gen**) is that the uniqueness property required for cofreeness fails. That is,

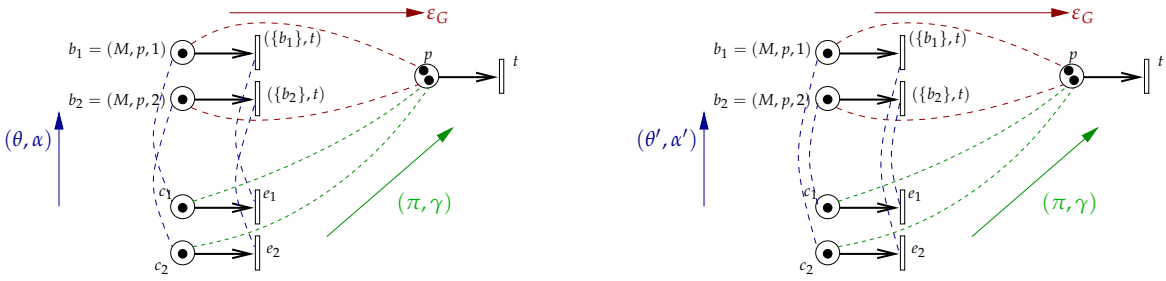
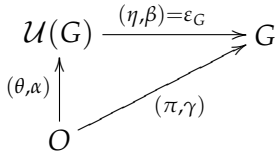


Figure 1: Non-uniqueness of mediating morphism (all multiplicities 1)

the morphism (θ, α) need not be the *unique* such morphism making the diagram above commute. In Figure 1, we present a general net G, its unfolding $\mathcal{U}(G)$ with morphism ε_G and an occurrence net O (which happens to be isomorphic to $\mathcal{U}(G)$) with morphism $(\pi, \gamma) : O \to G$ alongside two distinct morphisms $(\theta, \alpha), (\theta', \alpha') : O \to \mathcal{U}(G)$ making the diagram commute.

In the net $\mathcal{U}(G)$ in Figure 1, the two conditions b_1 and b_2 are symmetric: they arise from there being two indistinguishable tokens in the initial marking of G in the place p. The events $(\{b_1\}, t)$ and $(\{b_2\}, t)$ are also symmetric since they are only distinguished by their symmetric pre-conditions; they have common image under ε_G. Our goal shall be to show that there is a unique mediating morphism *up to symmetry*, *i.e.* any two morphisms from O to $\mathcal{U}(G)$ making the diagram commute are only distinguished through their choice of symmetric elements of the unfolding. We first summarize the part of the cofreeness property that does hold.

THEOREM 6. *Let G be a general Petri net, O be an occurrence net and $(\pi, \gamma) : O \to G$ be a morphism in $\mathbf{Gen}^\sharp$. There is a morphism $(\theta, \alpha) : O \to \mathcal{U}(G)$ in $\mathbf{Gen}^\sharp$ such that the following diagram commutes:*



Furthermore, if the net G is a P/T net then (θ, α) is the unique such morphism.

It will be of use later to note that if the multirelation γ above is a function then so is α.

2.2 Pullbacks

The framework for defining symmetry in general nets, to be described in the next section, will require a subcategory which has pullbacks. Whereas it was shown in [3] that the category of singly-marked safe nets has pullbacks, the category of singly-marked general nets does not. Roughly, this is for two reasons: the category with multirelations as morphisms does not have pullbacks; and allowing only singly-marked nets obstructs the existence of pullbacks. It is the latter obstruction that led us to the earlier relaxation of the definition

of nets, to permit them to have a set of initial markings rather than precisely one initial marking. To obtain a category of general nets with pullbacks, we restrict attention to *folding* morphisms between general nets (with multiple initial markings):

DEFINITION 7. *A morphism $(\eta, \beta) : G \to G'$ is a* folding *if both η and β are total functions.*

Denote the category of general nets with folding morphisms $\mathbf{Gen}^\sharp_f$, its full subcategory of occurrence nets $\mathbf{Occ}^\sharp_f$, and the full subcategory of causal nets $\mathbf{Caus}^\sharp_f$.

PROPOSITION 8. *The category $\mathbf{Gen}^\sharp_f$ has pullbacks.*

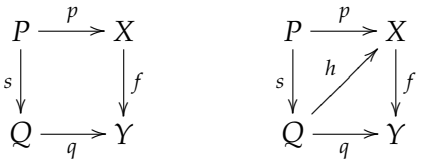
The category $\mathbf{Occ}^\sharp$ has pullbacks, though we will only need pullbacks of folding morphisms. Pullbacks in $\mathbf{Occ}^\sharp_f$ are obtained by taking the corresponding pullbacks in $\mathbf{Gen}^\sharp_f$. The following lemma expresses how pullbacks in subcategories with folding morphisms are not disturbed in moving to larger categories with all morphisms, though in the case of general nets we have to settle for them becoming weak pullbacks.[†]

LEMMA 9. *(i)* *The inclusion functor $\mathbf{Occ}^\sharp_f \hookrightarrow \mathbf{Occ}^\sharp$ preserves pullbacks.*
(ii) *The inclusion functor $\mathbf{Occ}^\sharp_f \hookrightarrow \mathbf{Gen}^\sharp_f$ preserves pullbacks.*
(iii) *The inclusion functor $\mathbf{Gen}^\sharp_f \hookrightarrow \mathbf{Gen}^\sharp$ preserves weak pullbacks.*

3 Categories with symmetry

It is shown in [14] how *symmetry* can be defined between the paths of event structures, and more generally on any category of models satisfying certain properties. The absence of pullbacks in the category $\mathbf{Gen}^\sharp$ obliges us to extend the method when introducing symmetry to general nets and their unfoldings.

The definition of symmetry makes use of *open* morphisms [5]. Let $\mathcal{C}_0$ be a category (typically a category of models such as Petri nets) with a distinguished subcategory $\mathcal{P}$ of path objects (such as causal nets), to describe the shape of computation paths, and morphisms specifying how a path extends to another. A morphism $f : X \to Y$ in $\mathcal{C}_0$ is *$\mathcal{P}$-open* if, for any morphism $s : P \to Q$ in $\mathcal{P}$ and morphisms $p : P \to X$ and $q : Q \to Y$, if the diagram on the left commutes, *i.e.* $f \circ p = q \circ s$, then then there is a morphism $h : Q \to X$ such that the diagram on the right commutes, *i.e.* $h \circ s = p$ and $f \circ h = q$:



The path-lifting property expresses that via f any extension of a path in Y can be matched by an extension in X, and captures those morphisms f which are bisimulations, though understood generally with respect to a form of path specified by $\mathcal{P}$. It can be shown purely

[†]Recall a *weak* pullback is defined in a similar way to a pullback, but without insisting on uniqueness of the mediating morphism.

diagrammatically that open morphisms compose, and therefore form a subcategory, and are preserved under pullbacks in $\mathcal{C}_0$.

Assume categories

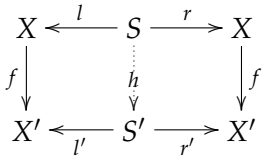
$$\mathcal{P} \subseteq \mathcal{C}_0 \subseteq \mathcal{C}$$

where $\mathcal{P}$ is a distinguished subcategory of path objects and path morphisms, $\mathcal{C}_0$ has pullbacks and shares the same objects as the (possibly larger) category $\mathcal{C}$, with the restriction that the inclusion functor $\mathcal{C}_0 \hookrightarrow \mathcal{C}$ preserves weak pullbacks. Then, we will be able to add symmetry to $\mathcal{C}$, and at the same time maintain constructions dependent on pullbacks of open morphisms which will be central to constructing symmetries on unfoldings.[‡] (The earlier method for introducing symmetry used in [14] corresponds to the situation where $\mathcal{C}_0$ and $\mathcal{C}$ coincide.)

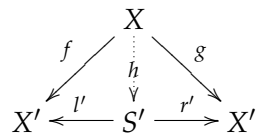
The role of $\mathcal{P} \subseteq \mathcal{C}_0$ is to determine open morphisms; the role of the subcategory $\mathcal{P}$ is to specify the form of path objects and extension, while the, generally larger, category $\mathcal{C}_0$ fixes the form of paths $p : P \to C$ from a path object P in an object C of $\mathcal{C}_0$. Now, just as earlier, we can define open morphisms in $\mathcal{C}_0$, and so by definition those in $\mathcal{C}$.

Now we show how $\mathcal{C}$ can be extended with symmetry to yield a category $\mathcal{SC}$. The objects of $\mathcal{SC}$ are tuples (X, S, l, r) consisting of an object X of $\mathcal{C}$ and two $\mathcal{P}$-open morphisms $l, r : S \to X$ in $\mathcal{C}_0$ which make l, r a pseudo equivalence [1] in the category $\mathcal{C}$ (see Appendix A). The requirements on l and r are slightly weaker than those in [14] in that we do not require that the morphisms l and r are jointly monic.[§]

The morphisms of $\mathcal{SC}$ are morphisms of $\mathcal{C}$ that *preserve symmetry*. Let $f : X \to X'$ be a morphism in $\mathcal{C}$ and (X, S, l, r) and (X', S', l', r') be objects of $\mathcal{SC}$. The morphism $f : X \to X'$ preserves symmetry if there is a morphism $h : S \to S'$ such that the following diagram commutes:



With the definition of symmetry on objects, we can define the equivalence relation $\sim$ expressing when morphisms are *equal up to symmetry*:
Let $f, g : (X, S, l, r) \to (X', S', l', r')$ be morphisms in $\mathcal{SC}$. Define $f \sim g$ iff there is a morphism $h : X \to X'$ in $\mathcal{C}$ such that following diagram commutes in $\mathcal{C}$:



Composition of morphisms in $\mathcal{SC}$ coincides with composition in $\mathcal{C}$ and the two categories share the same identity morphisms. The category $\mathcal{SC}$ is more fully described as a category enriched in equivalence relations.

[‡]We have chosen general conditions that work for our purposes here. It might become useful to replace the role of $\mathcal{P} \subseteq \mathcal{C}_0$ by an axiomatization of a subcategory of open morphisms in $\mathcal{C}$ and in this way broaden the class of situations in which we can adjoin symmetry.

[§]See [4] for an example of a symmetry on a safe net that cannot be expressed with the jointly-monic condition.

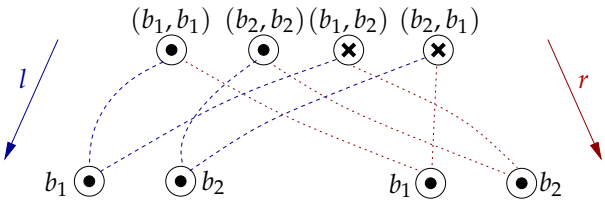


Figure 2: Symmetry in a net with two places

For nets, a reasonable choice for the paths $\mathcal{P}$ would be $\mathbf{Caus}_{\mathrm{f}}^{\sharp}$, taking path objects to be causal nets and expressing path extensions by foldings between them. (There are other possibilities, say restricting to finite causal nets, or the causal nets associated with finite elementary event structures, which would lead to less refined equivalences up to symmetry.) The categories $\mathbf{Caus}_{\mathrm{f}}^{\sharp} \subseteq \mathbf{Gen}_{\mathrm{f}}^{\sharp} \subseteq \mathbf{Gen}^{\sharp}$ meet the requirements needed to construct $\mathcal{S}\mathbf{Gen}^{\sharp}$ — in particular by Lemma 9 (iii), so adjoining symmetry to general nets. The requirements are also met by $\mathbf{Caus}_{\mathrm{f}}^{\sharp} \subseteq \mathbf{Occ}_{\mathrm{f}}^{\sharp} \subseteq \mathbf{Occ}^{\sharp}$ yielding $\mathcal{S}\mathbf{Occ}^{\sharp}$ (this time using Lemma 9 (ii)).

We remark that a folding morphism between general nets is $\mathbf{Caus}_{\mathrm{f}}^{\sharp}$-open in $\mathbf{Gen}_{\mathrm{f}}^{\sharp}$ iff it is $\mathbf{Caus}^{\sharp}$-open in $\mathbf{Gen}^{\sharp}$, and a folding morphism between occurrence nets is $\mathbf{Caus}_{\mathrm{f}}^{\sharp}$-open in $\mathbf{Occ}_{\mathrm{f}}^{\sharp}$ iff it is $\mathbf{Caus}^{\sharp}$-open in $\mathbf{Occ}^{\sharp}$.

4 Symmetry in unfolding

In Section 2.1, we showed how a general Petri net may be unfolded to form an occurrence net. This was shown not to yield a coreflection due to the mediating morphism not necessarily being unique. The key observation was that uniqueness might be obtained by regarding the net up to the evident symmetry between paths in the unfolding. This led us to define a category of general nets with symmetry. To give an example of the forms of symmetry that can be expressed, consider the simple net with two places, b_1 and b_2, both initially marked once. Suppose that we wish to express that the two places are symmetric; for instance, the net might be thought of as the unfolding of the general net with a single place initially marked twice. The span to express that symmetry is presented in Figure 2. Without our extension of the definition of net to allow multiple initial markings, this simple symmetry would be inexpressible. This accompanies the fact that the category of singly-marked general nets (even when restricted to folding morphisms) does not have pullbacks.

In general, the symmetry in an unfolding is obtained by unfolding the *kernel* of the morphism $\varepsilon_G : \mathcal{U}(G) \to G$, which is the pullback of ε_G against itself in $\mathbf{Gen}_{\mathrm{f}}^{\sharp}$:

$$\begin{array}{ccc} S & \xrightarrow{r} & \mathcal{U}(G) \\ {\scriptstyle l}\downarrow & \lrcorner & \downarrow{\scriptstyle \varepsilon_G} \\ \mathcal{U}(G) & \xrightarrow[\varepsilon_G]{} & G \end{array}$$

To see that $(\mathcal{U}(G), \mathcal{U}(S), l \circ \varepsilon_S, r \circ \varepsilon_S)$ is a symmetry, we must show that the morphisms $l \circ \varepsilon_S$ and $r \circ \varepsilon_S$ are $\mathbf{Caus}_{\mathrm{f}}^{\sharp}$-open and form a pseudo equivalence. The latter point follows a

purely diagrammatic argument. Open morphisms from occurrence nets into general nets can be characterized in the following way:

PROPOSITION 10. *Let O be an occurrence net and G be a general net. A morphism $f : O \to G$ is $\mathbf{Caus}^\sharp_f$-open in $\mathbf{Gen}^\sharp_f$ if, and only if, it reflects any initial marking of G to an initial marking of O and satisfies the following property:*

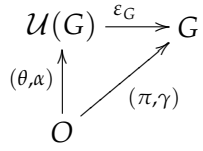
for any subset A of conditions of O such that co *A for which there exists a transition t of G such that $f \cdot A = Pre_G \cdot t$, there exists an event e of O such that $A = Pre_O \cdot e$ and $f(e) = t$.*

The morphism $\varepsilon_G : \mathcal{U}(G) \to G$ of Proposition 5 is readily seen to satisfy this property for any G, and is therefore $\mathbf{Caus}^\sharp_f$-open. The pullback of open morphisms is open [5] so the morphisms l and r are $\mathbf{Caus}^\sharp_f$-open, and therefore $l \circ \varepsilon_S$ and $r \circ \varepsilon_S$ are both open since open morphisms compose to form open morphisms [5]. Note that a morphism between occurrence nets is $\mathbf{Caus}^\sharp_f$-open in $\mathbf{Occ}^\sharp_f$ iff it is $\mathbf{Caus}^\sharp_f$-open in $\mathbf{Gen}^\sharp_f$.

PROPOSITION 11. *The tuple $(\mathcal{U}(G), \mathcal{U}(S), l \circ \varepsilon_S, r \circ \varepsilon_S)$ is an occurrence net with symmetry.*

With the symmetry on $\mathcal{U}(G)$ at our disposal, we obtain the equivalence relation $\sim$ on morphisms from any occurrence net to $\mathcal{U}(G)$. This is used to extend Theorem 6 to obtain cofreeness 'up to symmetry'.

THEOREM 12. *Let G be a general Petri net and O be an occurrence net. For any morphism $(\pi, \gamma) : O \to G$ in $\mathbf{Gen}^\sharp$, there is a morphism $(\theta, \alpha) : O \to \mathcal{U}(G)$ in $\mathbf{Gen}^\sharp$ such that*



commutes, i.e. $\varepsilon_G \circ (\theta, \alpha) = (\pi, \gamma)$. Furthermore, any morphism $(\theta', \alpha') : O \to \mathcal{U}(G)$ in $\mathbf{Gen}^\sharp$ such that $\varepsilon_G \circ (\theta', \alpha') = (\pi, \gamma)$ satisfies $(\theta, \alpha) \sim (\theta', \alpha')$ with respect to the symmetry (S, l, r) on $\mathcal{U}(G)$ defined above (and the identity symmetry on O).

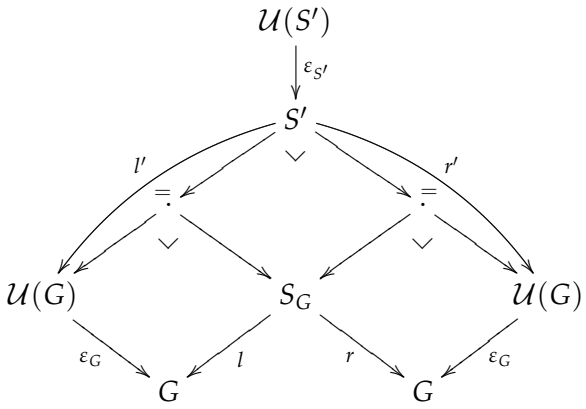
5 A coreflection up to symmetry

We show how the results of the last section are part of a more general coreflection from occurrence nets *with symmetry* to general nets *with symmetry*. In the last section, we showed how to unfold a general net to an occurrence net with symmetry. For the coreflection, we need to extend this construction to unfold general nets themselves with symmetry.

To show that the 'inclusion' $I : \mathcal{S}\mathbf{Occ}^\sharp \to \mathcal{S}\mathbf{Gen}^\sharp$ taking an occurrence net with symmetry (O, S, l, r) to a general net with symmetry is a functor, it is necessary to show that the transitivity property holds of the symmetry in $\mathcal{S}\mathbf{Gen}^\sharp$. For this it is important that pullbacks are not disturbed in moving from $\mathbf{Occ}^\sharp_f$ to the larger category $\mathbf{Gen}^\sharp_f$, as is assured by Lemma 9.

We now have a functor $I : \mathcal{S}\mathbf{Occ}^\sharp \to \mathcal{S}\mathbf{Gen}^\sharp$, respecting $\sim$, regarding an occurrence net with symmetry (O, S, l, r) itself directly as a general net with symmetry.

It remains for us to define the unfolding operation on objects of the category of general nets with symmetry. Its extension to a *pseudo* functor will follow from the biadjunction. Let (G, S_G, l, r) be a general net with symmetry. Let $\varepsilon_G : \mathcal{U}(G) \to G$ be the folding morphism given earlier in Proposition 5. It is open by Proposition 10. The general net (G, S_G, l, r) is 'unfolded' to the occurrence net with symmetry $\mathcal{U}(G, S_G, l, r) = (\mathcal{U}(G), S_0, l_0, r_0)$; its symmetry, $S_0 \triangleq \mathcal{U}(S')$, $l_0 \triangleq l' \circ \varepsilon_{S'}$ and $r_0 \triangleq r' \circ \varepsilon_{S'}$, is given by unfolding the *inverse image* S', l', r' of the symmetry in G along the open morphism $\varepsilon_G : \mathcal{U}(G) \to G$:

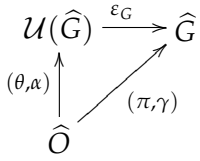


The pullbacks are in $\mathbf{Gen}^\sharp_{\mathrm{f}}$. The diagram makes clear that ε_G is a morphism preserving symmetry.

The construction of the symmetry above depends crucially on the existence of pullbacks in $\mathcal{C}_0$ and the property that pullbacks of open morphisms are open (here weak pullbacks do not suffice) — without this we would not know that l' and r' were open.

Now that we have the inclusion $I : \mathcal{S}\mathbf{Gen}^\sharp \to \mathcal{S}\mathbf{Occ}^\sharp$ and the operation of unfolding a general net with symmetry, we are able to generalize Theorem 6 to give a cofreeness result:

THEOREM 13. *Let $\widehat{G} = (G, S_G, l_G, r_G)$ be a general net with symmetry and $\widehat{O} = (O, S_O, l_O, r_O)$ be an occurrence net with symmetry. For any $(\pi, \gamma) : \widehat{O} \to \widehat{G}$ in $\mathcal{S}\mathbf{Gen}^\sharp$, there is a morphism $(\theta, \alpha) : \widehat{O} \to \mathcal{U}(\widehat{G})$ in $\mathcal{S}\mathbf{Gen}^\sharp$ such that the following diagram commutes:*



Furthermore, (θ, α) is unique up to symmetry: any $(\theta', \alpha') : \widehat{O} \to \mathcal{U}(\widehat{G})$ such that $\varepsilon_{\widehat{G}} \circ (\theta', \alpha') \sim (\pi, \gamma)$ satisfies $(\theta, \alpha) \sim (\theta', \alpha')$.

Technically, we have a biadjunction from $\mathcal{S}\mathbf{Occ}^\sharp$ to $\mathcal{S}\mathbf{Gen}^\sharp$ with I left biadjoint to $\mathcal{U}$ (which extends to a pseudo functor). Its counit is ε and its unit is a natural isomorphism $\widehat{O} \cong \mathcal{U}(\widehat{O})$. In this sense, we have established a coreflection from $\mathcal{S}\mathbf{Occ}^\sharp$ to $\mathcal{S}\mathbf{Gen}^\sharp$ up to symmetry.

6 Conclusion

Occurrence nets were first introduced in [8] together with the operation of unfolding singly-marked safe nets. The coreflection between occurrence nets and safe nets was first shown in [11]. A number of attempts have been made since then to characterize the unfoldings of more general forms of net.

Engelfriet defines the unfolding of (singly-marked) P/T nets in [2]. Rather than giving a coreflection between the categories, the unfolding is characterized as the greatest element of a complete lattice of occurrence nets embedding into the P/T net.

A coreflection between a subcategory of (singly-marked) general nets and a category of embellished forms of transition system is given in [7]. There, the restriction to particular kinds of net morphism is of critical importance; taking the more general morphisms of general Petri nets presented here would have resulted in the cofreeness property failing for an analogous reason to the failure of cofreeness of the unfolding of general nets to occurrence nets without symmetry.

An adjunction between a subcategory of singly-marked general nets and the category of occurrence nets is given in [6]. The restriction imposed on the morphisms of general nets there, however, precludes in general there being a morphism from $\mathcal{U}(G)$ to G in their category of general nets if $\mathcal{U}(G)$, the occurrence net unfolding of G, is regarded directly as a general net. To obtain an adjunction, the functor from the category of occurrence nets into the category of general nets is not regarded as the direct inclusion, but instead occurs through a rather detailed construction and does not yield a coreflection apart from when restricted to the subcategory of semi-weighted nets.

In this paper, we have shown that there is an implicit symmetry between paths in the unfolding of a general net arising from multiplicities in its initial marking and multiplicities on arcs from its transitions. By placing this symmetry on the unfolding, extending the scheme in [14], we are able to obtain its cofreeness up to symmetry, thus characterizing the unfolding up to the symmetry. We then adjoin symmetry to the categories of general nets and occurrence nets (using the standard definition of net morphism) to obtain a coreflection up to symmetry.

It is becoming clear from this and other work [10] that sometimes, in adjoining symmetry, models do not fit the simple scheme outlined in [14] appropriate to event structures and stable families. For example, the category of general nets with *all* morphisms does not have pullbacks as is required for the scheme in [14]. Alongside [10], the consideration of how symmetry may be placed on nets here and in [4] has suggested that we allow more liberal axioms on categories of models which enable their extension with symmetry.

The generalization of nets presented here to allow them to have more than one initial marking is also necessary for equipping other, less general, forms of net, such as safe nets or occurrence nets, with symmetry. In the companion paper [4], we extend the existing coreflection between singly-marked occurrence nets and P/T nets to this setting and show that this yields a coreflection between occurrence nets with symmetry and P/T nets with symmetry. In [4], we exhibit coreflections between event structures and multiply-marked occurrence nets.

References

[1] A. Carboni and E. M. Vitale. Regular and exact completions. *Journal of Pure and Applied Algebra*, 125(1–3):79–116, March 1998.

[2] J. Engelfriet. Branching processes of Petri nets. *Acta Informatica*, 28:575–591, 1991.

[3] E. Fabre. On the construction of pullbacks for safe Petri nets. In *Proc. ICATPN '06*, volume 4024 of *Lecture Notes in Computer Science*, 2006.

[4] J. Hayman and G. Winskel. Symmetry in Petri nets. To appear.

[5] A. Joyal, M. Nielsen, and G. Winskel. Bisimulation from open maps. In *Proc. LICS '93*, volume 127(2) of *Information and Computation*, 1995.

[6] J. Meseguer, U. Montanari, and V. Sassone. On the semantics of Place/Transition Petri nets. *Mathematical Structures in Computer Science*, 7:359–397, 1996.

[7] M. Mukund. Petri nets and step transition systems. *International Journal of Foundations of Computer Science*, 3(4):443–478, 1992.

[8] M. Nielsen, G. Plotkin, and G. Winskel. Petri nets, event structures and domains, Part 1. *Theoretical Computer Science*, 13:85–108, 1981.

[9] W. Reisig. *Petri Nets*. EATCS Monographs on Theoretical Computer Science. Springer-Verlag, 1985.

[10] G. Winskel. The symmetry of stability. Forthcoming.

[11] G. Winskel. A new definition of morphism on Petri nets. In *Proc. STACS '84*, volume 166 of *Lecture Notes in Computer Science*, 1984.

[12] G. Winskel. Event structures. In *Advances in Petri Nets, Part II*, volume 255 of *Lecture Notes in Computer Science*. 1986.

[13] G. Winskel. Petri nets, algebras, morphisms and compositionality. *Information and Computation*, 72(3):197–238, 1987.

[14] G. Winskel. Event structures with symmetry. *Electronic Notes in Theoretical Computer Science*, 172, 2007.

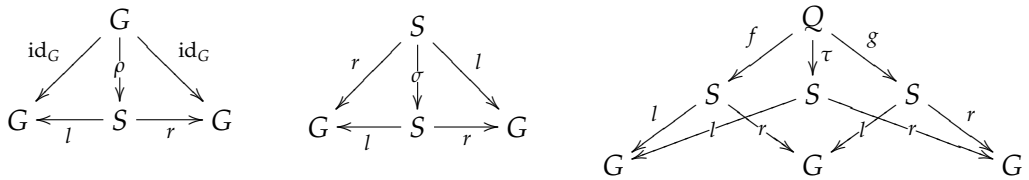
[15] G. Winskel. Symmetry and concurrency. In *Proc. CALCO '07*, May 2007. Invited talk.

[16] G. Winskel and M. Nielsen. Models for concurrency. In *Handbook of Logic and the Foundations of Computer Science*, volume 4, pages 1–148. Oxford University Press, 1995.

A Pseudo equivalences

Assume a category $\mathcal{C}$. Let $l, r : S \to G$ be a pair of morphisms in $\mathcal{C}$. They form a *pseudo equivalence* (and if jointly monic, an *equivalence*) iff there exist morphisms ρ, σ and τ such that the following diagrams commute, where Q, f, g is the pullback of l against r:

Reflexivity **Symmetry** **Transitivity**



Foundations of Software Technology and Theoretical Computer Science (Bangalore) 2008.
Editors: R. Hariharan, M. Mukund, V. Vinay; pp 235-243

Explicit Muller Games are PTIME*

Florian Horn
horn@liafa.jussieu.fr

LIAFA
Université Paris 7
Case 7014,
75205 Paris cedex 13
France

LI7
RWTH
Ahornstraße 55
52056 Aachen
Germany

LABRI
Université Bordeaux 1
351, cours de la Libération
33405 Talence cedex
France

ABSTRACT. Regular games provide a very useful model for the synthesis of controllers in reactive systems. The complexity of these games depends on the representation of the winning condition: if it is represented through a win-set, a coloured condition, a Zielonka-DAG or Emerson-Lei formulae, the winner problem is PSPACE-complete; if the winning condition is represented as a Zielonka tree, the winner problem belongs to NP and co-NP. In this paper, we show that explicit Muller games can be solved in polynomial time, and provide an effective algorithm to compute the winning regions.

1 Introduction

There has been a long history of using infinite games to model reactive processes [BL69, PR89]. The system is represented as a game arena, *i.e.* a graph whose states belong either to Eve (controller) or to Adam (environment). The desired behaviour is represented as an ω-regular winning condition, which naturally expresses temporal specifications and fairness assumptions of transition systems [MP92]. The game is played by moving a token on the arena: when it is in one of Eve's states, she chooses its next location among the successors of the current state; when it is in one of Adam's states, he chooses its next location. The result of playing the game for ω moves is an infinite path of the graph. Eve wins if the path satisfies the specification, and Adam wins otherwise.

A fundamental determinacy result of Büchi and Landweber shows that from any initial state, one of the players has a winning strategy [BL69]. The problem of the winner is in

*This work was supported in part by the French ANR AVERISS.

FSTTCS 2008
IARCS Annual Conference on
Foundations of Software Technology and Theoretical Computer Science
http://drops.dagstuhl.de/opus/volltexte/2008/1756

PSPACE for any reasonable representation of the winning condition [McN93, NRY96], but its exact complexity depends on how the winning condition is represented. For example, if the winning condition is represented as a Zielonka tree [Zie98], the problem of the winner is in NP ∩ co-NP [DJW97]. Hunter and Dawar list in [HD05] five other "general purpose" representations: explicit Muller, win-set, Muller, Zielonka DAGs, Emerson-Lei. They show that the problem of the winner is PSPACE-hard for the last four representations, and leave the complexity of explicit Muller games as an open question. In this paper, we answer this question: the winner problem in explicit Muller games belongs to PTIME. We provide an effective cubic algorithm computing the winning regions of the players.

Outline of the paper. Section 2 recalls the classical notions about regular games, and Section 3 gives an overview of the different representations of regular winning conditions. In Section 4, we introduce the notions of semi-alternation and sensibleness, and show that any explicit Muller game can be translated in polynomial time into a semi-alternating and sensible game. We also study the family of games where Eve wins if *all* the states are visited infinitely often. These games are used repeatedly in our algorithm, which is the subject of Section 5.

2 Definitions

We recall here several classical notions about regular games, and refer the reader to [GTW02] for more details.

Arenas.

An *arena* $\mathcal{A}$ is a directed graph $(\mathcal{Q}, \mathcal{T})$ without deadlocks whose states are partitioned between Eve's states ($\mathcal{Q}_E$, represented as $\bigcirc$'s) and Adam's states ($\mathcal{Q}_A$, represented as $\Box$'s). A sub-arena $\mathcal{A}_{|B}$ of $\mathcal{A}$ is the restriction of $\mathcal{A}$ to a subset B of $\mathcal{Q}$ such that each state of B has a successor in B.

Plays and Strategies.

A *play* on the arena $\mathcal{A}$ is a (possibly infinite) sequence $\rho = \rho_0\rho_1 \dots$ of states such that $\forall i < |\rho|-2,\ (\rho_i, \rho_{i+1}) \in \mathcal{T}$. The set of *occurring states* is $\mathrm{Occ}(\rho) = \{q \mid \exists i \in \mathbb{N}, \rho_i = q\}$, and the set of *limit states* is $\mathrm{Inf}(\rho) = \{q \mid \exists^\infty i \in \mathbb{N}, \rho_i = q\}$.

A *strategy* of Eve on the arena $\mathcal{A}$ is a function σ from $\mathcal{Q}^*\mathcal{Q}_E$ to $\mathcal{Q}$ such that $\forall w \in \mathcal{Q}^*, \forall q \in \mathcal{Q}_E, (q, \sigma(wq)) \in \mathcal{T}$. Strategies can also be defined as *strategies with memory*. In this case, σ is a triple $(M, \sigma^{\mathtt{u}}, \sigma^{\mathtt{n}})$, where M is the (possibly infinite) set of *memory states*, $\sigma^{\mathtt{u}} : (M \times \mathcal{Q}) \to M$ is the *memory update* function, and $\sigma^{\mathtt{n}} : (M \times \mathcal{Q}) \to \mathcal{Q}$ is the *next-move* function. Adam's strategies are defined in a similar way. A strategy is *finite-memory* if M is a finite set, and *memoryless* if M is a singleton.

A (finite or infinite) play ρ is *consistent with* σ if, $\forall i < |\rho|-2,\ \rho_i \in \mathcal{Q}_E \Rightarrow \rho_{i+1} = \sigma(\rho_0 \dots \rho_i)$.

Traps and Attractors.

The *attractor of Eve to the set U in the arena $\mathcal{A}$*, denoted $\mathrm{Attr}_E(U, \mathcal{A})$, is the set of states from where Eve can force the token to go to the set U. It is defined inductively by:

$$\begin{aligned} U_0 &= U \\ U_{i+1} &= U_i \cup \{q \in \mathcal{Q}_E, \exists r \in U_i \mid (q,r) \in \mathcal{T}\} \\ &\quad \cup \{q \in \mathcal{Q}_A \mid \forall r, (q,r) \in E \Rightarrow r \in U_i\} \\ \mathrm{Attr}_E(U, \mathcal{A}) &= \bigcup_{i>0} U_i \end{aligned}$$

The corresponding *attractor strategy to U for Eve* is a positional strategy σ_U such that for any state $q \in \mathcal{Q}_E \cap (\mathrm{Attr}_E(U, \mathcal{A}) \setminus U)$, $q \in U_{i+1} \Rightarrow \sigma_U(q) \in U_i$.

The dual notion of a *trap for Eve* denotes a set from where Eve cannot escape, unless Adam allows her to do so: a set U is a trap for Eve if and only if $\forall q \in U \cap \mathcal{Q}_E, (q,r) \in \mathcal{T} \Rightarrow r \in U$ and $\forall q \in U \cap \mathcal{Q}_A, \exists r \in U \mid (q,r) \in \mathcal{T}$. Notice that a trap is always a sub-arena.

Regular Winning Conditions.

A *regular winning condition* is a specification $\Phi \subseteq \mathcal{Q}^\omega$ on infinite plays which depends only on the set of states visited infinitely often: $\mathrm{Inf}(\rho) = \mathrm{Inf}(\nu) \Rightarrow (\rho \in \Phi \Leftrightarrow \nu \in \Phi)$. Eve *wins a play* ρ if $\rho \in \Phi$. Adam wins if $\rho \notin \Phi$. Regular winning conditions can be described in different ways, which are presented in the next section.

Winning Strategies.

Given a winning condition Φ and a state $q \in \mathcal{Q}$, a strategy σ is *winning for Eve from q* if any play starting in q and consistent with σ is winning for Eve. The *winning region* of Eve is the set of states from where she has a winning strategy. Adam's winning strategies and regions are defined in a similar way.

3 Representations of regular conditions

The most straightforward way to represent a regular condition $\mathcal{F}$ is to provide an explicit list of sets of states $\mathcal{F}_1, \ldots, \mathcal{F}_\ell$: $\mathcal{F} = \{\mathcal{F}_i \mid 1 \leq i \leq \ell\}$. A play ρ is winning for Eve if and only if $\mathrm{Inf}(\rho) \in \mathcal{F}$. The complexity of these *explicit Muller games* is the subject of this paper.

There are several other ways to represent regular conditions. In *win-set games* [McN93], the winner depends only on a subset R of *relevant states*, and the winning condition $\mathcal{R}$ lists only subsets of R: ρ is winning for Eve if $\mathrm{Inf}(\rho) \cap R \in \mathcal{R}$. *Muller games* extend this idea by adding a *colouring function* χ, from the states to a set of colours $\mathcal{C}$. The winning condition $\mathcal{F}$ lists subsets of $\mathcal{C}$, and ρ is winning for Eve if $\chi(\mathrm{Inf}(\rho)) \in \mathcal{F}$. *Emerson-Lei* games [EL85] provide a boolean formula φ, whose variables are the states of $\mathcal{Q}$. A play ρ is winning for Eve if the valuation $\mathrm{Inf}(\rho) \leftarrow \texttt{true}$ and $\mathcal{Q} \setminus \mathrm{Inf}(\rho) \leftarrow \texttt{false}$ satisfies φ.

Zielonka's representation of regular conditions [Zie98] proceeds from a different approach: it focuses on alternation between sets winning for Eve and sets winning for Adam. In his split tree (usually called "Zielonka tree"), the nodes are labelled by sets of colours, the children are subsets of their parent with $\mathcal{C}$ at the root, and a child and its parent are never winning for the same player. Finally, Zielonka DAGs [HD05] are the result of merging the nodes of the Zielonka tree with the same labels.

The complexity of regular games depends directly on the representation of the winning condition:

THEOREM 1. *[DJW97] The problem of the winner in regular games whose winning condition is represented by a Zielonka tree is in* NP $\cap$ *co-*NP .

THEOREM 2. *[HD05] The problem of the winner in win-set games, Muller games, Zielonka DAG games, and Emerson-Lei games are* PSPACE*-complete.*

For explicit Muller games, the best complexity result so far was the membership of the winner problem in PSPACE, derived from the "all-purpose" algorithms of [McN93] and [NRY96]. The main result of this paper is Theorem 3:

THEOREM 3. *The winner problem of explicit Muller games can be solved in polynomial time.*

4 Useful notions for explicit Muller games

We first define three properties of explicit Muller games. A game is:

1. *semi-alternating* if there is no transition between two states of Adam (but there can be between two states of Eve);
2. *sensible* if each set in $\mathcal{F}$ induces a sub-arena of $\mathcal{A}$;
3. *ordered for inclusion* if $i < j \Rightarrow \mathcal{F}_i \not\supseteq \mathcal{F}_j$.

Our algorithm for explicit Muller games, Algorithm 1, relies on the fact that its input satisfies these three properties. However, this does not restrict the generality of our result, since any explicit Muller game can be transformed in polynomial time into an equivalent semi-alternating, sensible, and ordered game of polynomial size. The semi-alternation transformation consists in replacing each state $q \in Q_A$ of Adam by a pair of states $r \in Q_E, s \in Q_A$, as in Figure 1. Each set containing q in the winning condition is modified accordingly: $\mathcal{F} \leftarrow (\lambda q.(r,s))\mathcal{F}$. This is where the classical alternation transformation fails: adding a state to each transition leads to an exponential blow-up in the size of the winning condition.

A game can be made sensible by removing from $\mathcal{F}$ all the sets that do not induce a sub-arena of $\mathcal{A}$: no matter how Eve and Adam play, the limit of the play is a a sub-arena, so the modification is transparent with respect to deciding the winning nature of a play, a strategy, or a state. Finally, ordering the sets for inclusion can be done in quadratic time.

The games of the form $(\mathcal{A}, \{\mathcal{Q}\})$, where Eve wins if and only if the token visits all the states infinitely often, play an important part in our solution to explicit Muller games. These games, which have also been studied in routing problems [DK00, IK02], are easy to solve and there is always only one winner in the whole game:

(a) Original arena $\mathcal{A}$ (b) Semi-alternating arena $\mathbb{A}$

Figure 1: Semi-alternating arena construction

PROPOSITION 4. *Let $\mathcal{A}$ be an arena, and $\mathcal{G}$ be the game $(\mathcal{A}, \{\mathcal{Q}\})$. Either, for any state $q \in \mathcal{Q}$, Eve's attractor to q is equal to $\mathcal{Q}$, and Eve wins everywhere in $\mathcal{G}$, or there is a state $q \in \mathcal{Q}$ such that $\mathrm{Attr}_E(\{q\}, \mathcal{A}) \neq \mathcal{Q}$, and Adam wins everywhere in $\mathcal{G}$.*

PROOF. In the first case, Eve can win with a strategy whose memory states are the states of $\mathcal{Q}$: in the memory state q, she plays the attractor strategy to q, until the token reaches it. She updates then her memory to the next state r, in a circular way. In the second case, Adam can win surely with any trapping strategy out of $\mathrm{Attr}_E(\{q\}, \mathcal{A})$: if the token ever gets out of $\mathrm{Attr}_E(\{q\}, \mathcal{A})$, it never goes back.

5 Solving explicit Muller games in PTIME

Our algorithm takes as input a semi-alternating, sensible explicit Muller game whose winning condition is ordered for inclusion; it returns the winning regions of the players. Each set in $\mathcal{F}$ is considered at most once, starting with the (smallest) set $\mathcal{F}_1$. At each step, the operation of a set $\mathcal{F}_i$ modifies the arena and the winning condition in one of the following ways:

If Adam wins $(\mathcal{A}_{|\mathcal{F}_i}, \{\mathcal{F}_i\})$**,** $\mathcal{F}_i$ is removed from $\mathcal{F}$.

If Eve wins $(\mathcal{A}_{|\mathcal{F}_i}, \{\mathcal{F}_i\})$**, and $\mathcal{F}_i$ is a trap for Adam in $\mathcal{A}$,** Eve's attractor to $\mathcal{F}_i$ in $\mathcal{A}$, $\mathrm{Attr}_E(\mathcal{F}_i, \mathcal{A})$, is removed from $\mathcal{A}$ (and added to the winning region of Eve), and all the sets intersecting $\mathrm{Attr}_E(\mathcal{F}_i, \mathcal{A})$ are removed from $\mathcal{F}$.

If Eve wins $(\mathcal{A}_{|\mathcal{F}_i}, \{\mathcal{F}_i\})$**, and $\mathcal{F}_i$ is not a trap for Adam in $\mathcal{A}$,** a new state $\mathbb{F}_i$, described in Figure 2, is added to $\mathcal{A}$ with the following attributes:

- $\mathbb{F}_i$ is a state of Adam;
- the predecessors of $\mathbb{F}_i$ are all the states of Eve in $\mathcal{F}_i$;
- the successors of $\mathbb{F}_i$ are the successors outside $\mathcal{F}_i$ of the states of Adam in $\mathcal{F}_i$.

Furthermore, the state $\mathbb{F}_i$ is added to all the supersets of $\mathcal{F}_i$ in $\mathcal{F}$, and $\mathcal{F}_i$ itself is removed from $\mathcal{F}$.

The important case, from an intuitive point of view, is the last one: it corresponds to a "threat" of Eve to win by visiting exactly the states of $\mathcal{F}_i$. Adam has to answer by getting out, but he can choose his exit from any of his states. Notice that it would not do to simply replace the whole region $\mathcal{F}_i$ by the state $\mathbb{F}_i$: as in Figure 2, Adam may be able to avoid a

$\mathcal{F} = \{\{a, b\}, \{a, b, c\}\}$

(a) Before

$\mathcal{F} = \{\{a, b, c, \text{“}\{a, b\}\text{”}\}\}$

(b) After

Figure 2: Removal of a set in an explicit Muller condition

state of $\mathcal{F}_i$ in a larger arena, even if he is incapable of doing so in $\mathcal{A}_{|\mathcal{F}_i}$.

As only one state is added each step, the number of states in the game is bounded by $|\mathcal{A}| + |\mathcal{F}|$. The whole procedure is described as Algorithm 1.

In the proof of correctness, we use `typewriter` fonts to denote the modified arena and condition, and *calligraph* fonts to denote the original game. Furthermore, we denote by $\mathcal{F}_{|\mathcal{F}_i}$ the intersection of $\mathcal{F}$ and $\mathcal{P}(\mathcal{F}_i)$, *i.e.* the sets of $\mathcal{F}$ that are also subsets of $\mathcal{F}_i$. We can now proceed to the three main lemmas:

LEMMA 5. *If, in the course of a run of Algorithm 1, the game* $(\mathtt{A}_{|\mathtt{F}_i}, \{\mathtt{F}_i\})$ *is winning for Eve at line 6, then Eve wins everywhere in the game* $(\mathcal{A}_{|\mathcal{F}_i}, \mathcal{F}_{|\mathcal{F}_i})$.

PROOF. Let $\mathcal{H}^1, \dots, \mathcal{H}^k$ be the sets of $\mathcal{F}_{|\mathcal{F}_i}$ such that $(\mathtt{A}_{|\mathtt{H}^j}, \{\mathtt{H}^j\})$ was winning for Eve in the run of Algorithm 1. Notice that $\mathcal{F}_i$ itself is one of these states, say $\mathcal{H}^k$. The σ^j's denote her corresponding winning strategies. We build a strategy σ for Eve in $\mathcal{A}_{|\mathcal{F}_i}$, whose memory states are stacks of pairs $(\mathcal{H}^j, \rho^j)$. At any time, ρ^j is a play of $\mathtt{A}_{|\mathtt{H}^j}$ which can be extended by the current state q. The initial memory state is $(\mathcal{H}^k, \varepsilon)$, and the operation of σ when the memory state is $(\mathcal{H}^j, w)$ and the current state is q is described below:

1. If $q \notin \mathcal{H}^j$, the top pair is removed, and the procedure restarts at step 1 with the new memory. Notice that it may involve further pops if q still does not belong to the top set.
2. If q is a state of Eve, and $\sigma^j(wq)$ is a new state $\mathbb{H}^h$, the memory is modified as follows: w becomes $wq\mathbb{H}^h$, and a new pair $(\mathcal{H}^h, \varepsilon)$ is pushed at the top of the stack. The procedure restarts at step 2 with the new memory. Notice that it may involve further pushes if $\sigma^h(q)$ is also a new state.
3. The new memory state is $(\mathcal{H}^j, wq)$; if q belongs to Eve, she plays $\sigma^j(wq)$.

We claim that σ is winning for Eve in the game $(\mathcal{A}_{|\mathcal{F}_i}, \mathcal{F}_{|\mathcal{F}_i})$. Let ρ be a play consistent with σ, and $\mathcal{H}^j$ be the highest set that is never unstacked. We denote by ρ^j the (infinite) limit of the "play" part. As ρ^j is consistent with σ^j, $\mathrm{Inf}(\rho^j) = \mathtt{H}^j$. Furthermore, $\mathrm{Inf}(\rho) \supseteq \mathrm{Inf}(\rho^j) \cap \mathcal{Q}$ and $\mathrm{Inf}(\rho) \subseteq \mathcal{H}^j$. So, $\mathrm{Inf}(\rho) = \mathcal{H}^j \in \mathcal{F}$, and Lemma 5 follows.

Input: An explicit Muller game $(\mathcal{A}, \mathcal{F})$
Output: The winning regions of Eve and Adam
1 $\mathtt{A} = (\mathtt{Q}, \mathtt{Q}_E, \mathtt{Q}_A, \mathtt{T}) \leftarrow \mathcal{A} = (\mathcal{Q}, \mathcal{Q}_E, \mathcal{Q}_A, \mathcal{T})$;
2 $\mathtt{F} \leftarrow \mathcal{F}$;
3 $\mathtt{W}_E \leftarrow \emptyset$;
4 **while** $\mathtt{F} \neq \emptyset$ **do**
5 $\mathtt{F}_i \leftarrow \text{pop}(\mathtt{F})$;
6 **if** *Eve wins* $(\mathtt{A}_{|\mathtt{F}_i}, \{\mathtt{F}_i\})$ **then**
7 **if** $\mathtt{F}_i$ *is a trap for Adam in* $\mathtt{A}$ **then**
8 remove $\text{Attr}_E(\mathtt{F}_i, \mathtt{A})$ from $\mathtt{A}$ and add it to $\mathtt{W}_E$;
9 remove all the sets intersecting $\text{Attr}_E(\mathtt{F}_i, \mathtt{A})$ from $\mathtt{F}$;
10 **else**
11 add a state $\mathbb{F}_i$ to $\mathtt{Q}_A$;
12 add transitions from $\mathtt{F}_i \cap \mathtt{Q}_E$ to $\mathbb{F}_i$;
13 add transitions from $\mathbb{F}_i$ to $\mathtt{T}(\mathtt{F}_i \cap \mathtt{Q}_A) \setminus \mathtt{F}_i$;
14 add $\mathbb{F}_i$ to all the supersets of $\mathtt{F}_i$ in $\mathtt{F}$;
15 **end**
16 **end**
17 **end**
18 **return** $\mathtt{W}_E \cap \mathcal{Q}, \mathtt{Q} \cap \mathcal{Q}$

Algorithm 1: Polynomial algorithm for explicit Muller games

For Adam, the problem is a little more complex: we need two lemmas, whose proofs are mutually recursive:

LEMMA 6. *If, in the course of a run of Algorithm 1, the game* $(\mathtt{A}_{|\mathtt{F}_i}, \{\mathtt{F}_i\})$ *is winning for Adam at line 6, then Adam wins everywhere in the game* $(\mathcal{A}_{|\mathcal{F}_i}, \mathcal{F}_{|\mathcal{F}_i})$.

LEMMA 7. *If, in the course of a a run of Algorithm 1, the game* $(\mathtt{A}_{|\mathtt{F}_i}, \{\mathtt{F}_i\})$ *is winning for Eve at line 6, then Adam wins everywhere in the game* $(\mathcal{A}_{|\mathcal{F}_i}, \mathcal{F}_{|\mathcal{F}_i} \setminus \{\mathcal{F}_i\})$.

PROOF. We start with the (simpler) proof of Lemma 7. Let $\mathcal{H}^1, \ldots, \mathcal{H}^k$ be the maximal sets, with respect to inclusion, of $\mathcal{F}_{|\mathcal{F}_i}$. There is a winning strategy τ^j for Adam in each $\mathcal{H}^j$: if Adam won $(\mathtt{A}_{|\mathtt{H}^j}, \{\mathtt{H}^j\})$, it is a winning strategy for the game $(\mathcal{A}_{|\mathcal{H}^j}, \mathcal{F}_{|\mathcal{H}^j})$ (recursive use of Lemma 6); if Eve won $(\mathtt{A}_{|\mathtt{H}^j}, \{\mathtt{H}^j\})$, it is a strategy for the game $(\mathcal{A}_{|\mathcal{H}^j}, \mathcal{F}_{|\mathcal{H}^j} \setminus \mathcal{H}^j)$ (recursive use of Lemma 7). The strategy τ for Adam in $(\mathcal{A}_{|\mathcal{F}_i}, \{\mathcal{F}_{|\mathcal{F}_i}\})$ uses k top-level memory states to switch between the $\{\tau^j\}_{1 \leq j \leq k}$. Adam remains in a top-level memory state j only as long as the token is in $\mathcal{H}^j$. As soon as it gets out, he updates it to $(j \mod k) + 1$. His actions when the top-level memory state is j are described below:

- if he won $(\mathtt{A}_{|\mathtt{H}^j}, \{\mathtt{H}^j\})$, he plays τ^j;
- if Eve won $(\mathtt{A}_{|\mathtt{H}^j}, \{\mathtt{H}^j\})$, he plays τ^j unless he can get out of $\mathcal{H}^j$.

We claim that τ is winning for Adam in $(\mathcal{A}_{|\mathcal{F}_i}, \mathcal{F}_{|\mathcal{F}_i})$. Any play ρ consistent with τ falls in exactly one of the three following categories:

- The top-level memory of τ is not ultimately constant; thus $\mathrm{Inf}(\rho)$ is not included in any of the $\mathcal{H}^j$'s, and ρ is winning for Adam.
- The top-level memory of τ is ultimately constant at j, and $(\mathtt{A}_{|\mathtt{H}^j}, \{\mathtt{H}^j\})$ was winning for Adam; ρ is ultimately a play of $\mathcal{A}_{|\mathcal{H}^j}$ consistent with τ^j, so ρ is winning for Adam.
- The top-level memory of τ is ultimately constant at j, and $(\mathtt{A}_{|\mathtt{H}^j}, \{\mathtt{H}^j\})$ was winning for Eve; ρ is ultimately a play of $\mathcal{A}_{|\mathcal{H}^j}$ consistent with τ^j, so Eve can win only by visiting all the states of $\mathcal{H}^j$. But $\mathcal{H}^j$ is not a trap for Adam, and the definition of τ implies that Adam leaves as soon as possible. So, at least one of the states of $\mathcal{H}^j$ was not visited, and ρ is winning for Adam.

This completes the proof of Lemma 7. The proof of Lemma 6 is more involved, due to the necessity to avoid at least one of the states of $\mathcal{F}_i$. By Proposition 4 there is a state q in $\mathtt{F}_i$ such that $X = \mathrm{Attr}_E(\{q\}, \mathtt{A}_{|\mathtt{F}_i})$ is not equal to $\mathtt{A}_{|\mathtt{F}_i}$. It follows from the definition of $\mathtt{A}_{|\mathtt{F}_i}$ that neither $\mathcal{F}_i \cap X$ nor $\mathcal{F}_i \setminus X$ is empty. Adam's strategy is then exactly the same as in the proof of Lemma 7, with the provision that Adam never moves from $\mathcal{F}_i \setminus X$ to X: this guarantees that the token cannot visit infinitely often all the states of $\mathcal{F}_i$, and completes the proof of Lemma 6.

The correctness of Algorithm 1 follows from Lemmas 5, 6, and 7: the first one guarantees that the states in $\mathtt{W}_E \cap \mathcal{Q}$ are winning for Eve, and the others that the states remaining at the end of Algorithm 1 are winning for Adam.

About complexity, there are at most $|\mathcal{F}|$ loops in a run, and the most time-consuming operation is to compute the winner of the games $(\mathtt{A}_{|\mathtt{F}_i}, \{\mathtt{F}_i\})$, which are quadratic in $|\mathtt{A}| \leq (|\mathcal{A}| + |\mathcal{F}|)$. Thus, the worst-case time complexity of Algorithm 1 is $O(|\mathcal{F}| \cdot (|\mathcal{A}| + |\mathcal{F}|)^2)$, which completes the proof of Theorem 3.

6 Conclusion

We have shown that the complexity of the winner problem in explicit Muller game belongs to PTIME, and provided a cubic algorithm computing the winning regions of both players.

It follows from the usual reduction between two-player games and tree automata that the emptiness problem of explicit Muller tree automata can also be solved in polynomial time; a natural question is whether this is also the case for other automata problems.

The existence of a polynomial algorithm for parity games remains an open problem: representing explicitly a parity condition incurs an exponential blow-up in size.

Bibliography

[BL69] J. Richard Büchi and Lawrence H. Landweber. Solving Sequential Conditions by Finite-State Strategies. *Transactions of the American Mathematical Society*, 138:295–311, 1969.

[DJW97] Stefan Dziembowski, Marcin Jurdziński, and Igor Walukiewicz. How Much Memory is Needed to Win Infinite Games? In *Proceedings of the 12th Annual IEEE Symposium on Logic in Computer Science, LICS'97*, pages 99–110. IEEE Computer Society, 1997.

[DK00] Michael J. Dinneen and Bakhadyr Khoussainov. Update Networks and Their Routing Strategies. In *Proceedings of the 26th International Workshop on Graph-Theoretic Concepts in Computer Science, WG'00*, volume 1928 of *Lecture Notes in Computer Science*, pages 127–136. Springer-Verlag, 2000.

[EL85] E. Allen Emerson and Chin-Laung Lei. Modalities for Model Checking: Branching Time Strikes Back. In *Proceedings of the 12th Annual ACM Symposium on Principles of Programming Languages, POPL'85*, pages 84–96, 1985.

[GTW02] Erich Grädel, Wolfgang Thomas, and Thomas Wilke, editors. *Automata, Logics, and Infinite Games: A Guide to Current Research [outcome of a Dagstuhl seminar, February 2001]*, volume 2500 of *Lecture Notes in Computer Science*. Springer-Verlag, 2002.

[HD05] Paul Hunter and Anuj Dawar. Complexity Bounds for Regular Games. In *Proceedings of the 30th International Symposium on Mathematical Foundations of Computer Science, MFCS'05*, volume 3618 of *Lecture Notes in Computer Science*, pages 495–506. Springer-Verlag, 2005.

[IK02] Hajime Ishihara and Bakhadyr Khoussainov. Complexity of Some Infinite Games Played on Finite Graphs. In *Proceedings of the 28th International Workshop on Graph-Theoretic Concepts in Computer Science, WG'02*, volume 2573 of *Lecture Notes in Computer Science*, pages 270–281. Springer-Verlag, 2002.

[McN93] Robert McNaughton. Infinite Games Played on Finite Graphs. *Annals of Pure and Applied Logic*, 65(2):149–184, 1993.

[MP92] Zohar Manna and Amir Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer-Verlag, 1992.

[NRY96] Anil Nerode, Jeffrey B. Remmel, and Alexander Yakhnis. McNaughton Games and Extracting Strategies for Concurrent Programs. *Annals of Pure and Applied Logic*, 78(1–3):203–242, 1996.

[PR89] Amir Pnueli and Roni Rosner. On the Synthesis of a Reactive Module. In *Proceedings of the 16th Annual ACM Symposium on Principles of Programming Languages, POPL'89*, pages 179–190, 1989.

[Zie98] Wieslaw Zielonka. Infinite Games on Finitely Coloured Graphs with Applications to Automata on Infinite Trees. *Theoretical Computer Science*, 200(1–2):135–183, 1998.

Foundations of Software Technology and Theoretical Computer Science (Bangalore) 2008.
Editors: R. Hariharan, M. Mukund, V. Vinay; pp 244-255

The Complexity of Tree Transducer Output Languages

Kazuhiro Inaba[1] **and Sebastian Maneth**[2,3]

[1] The University of Tokyo, kinaba@is.s.u-tokyo.ac.jp

[2] National ICT Australia, sebastian.maneth@nicta.com.au

[3] University of New South Wales, Sydney

ABSTRACT. Two complexity results are shown for the output languages generated by compositions of macro tree transducers. They are in NSPACE(n) and hence are context-sensitive, and the class is NP-complete.

1 Introduction

Macro tree transducers (mtts) [12, 14] are a finite-state machine model of tree-to-tree translations. They are motivated by syntax-directed semantics of programming languages and recently have been applied to XML transformations and query languages [18, 21]. Mtts are a combination of top-down tree transducersand macro grammars [13]. They process the input tree top-down while accumulating several output trees using their context parameters. Sequential composition of mtts gives rise to a powerful hierarchy (the "mtt-hierarchy") of tree translations which contains most known classes of tree translations such as those realized by attribute grammars, by MSO-definable tree translations [5], or by pebble tree transducers [20]. Consider the range, or output language, of a tree translation; it is a set of trees. If we apply "yield" to these trees, i.e., concatenate their leaf symbols from left to right, we obtain a string language. The string languages obtained in this way from the mtt-hierarchy form a large class (containing for instance the IO- and OI-hierarchies [6]) with good properties, such as being a full AFL and having decidable membership, emptiness, and finiteness [7].

In this paper we study the complexity of the output (string or tree) languages of the mtt-hierarchy. Note that we do not explicitly distinguish between string or tree output languages here, because the translation "yield" which turns a tree into its frontier string (seen as a monadic tree) is a particular simple macro tree translation itself and hence the corresponding classes have the same complexity. Small subclasses of our class of languages considered here are the IO-macro languages (or, equivalently, the yields of context-free-tree languages under IO-derivation) and the string languages generated by attribute grammars. Both of these classes are LOG(CFL)-complete by [2] and [10], respectively. Another subclass of our class is that of OI-macro languages, which are equivalent to the indexed languages [1], by [13]. This class is known to be NP-complete [22]. Hence, our class is NP-hard too (even already at level 2). Our first main result is that output languages of the mtt-hierarchy are NP-complete; thus, the complexity remains in NP when going from

FSTTCS 2008
IARCS Annual Conference on
Foundations of Software Technology and Theoretical Computer Science
http://drops.dagstuhl.de/opus/volltexte/2008/1757

indexed languages to the full mtt-hierarchy. In terms of space complexity, languages generated by compositions of top-down tree transducers (mtts without context parameters) are known to be in DSPACE(n) [3]. This result was generalized in [17] to compositions of *total deterministic* mtts. Our second main result is that output languages of the mtt-hierarchy (generated by compositions of *nondeterministic* mtts) with regular tree languages as inputs are in NSPACE(n) and thus are context-sensitive. The approach of our proof can be seen as a generalization of the proofs in [3] and [17]; moreover, we make essential use of the idea of compressed representation of backtracking information, used by Aho in [1] for showing that the indexed languages are in NSPACE(n).

We first solve the "translation membership" problem for a single mtt M. That is, we show that, given trees s and t, we can determine whether or not the pair (s,t) is in M's translation, in linear space and polynomial time with respect to $|s|+|t|$ on a nondeterministic Turing Machine ($|s|$ denotes the size of the tree s). The challenge here is the space complexity; we use a compressed representation of M's output trees for input s, inspired by [19], and then check if t is contained using a recursive procedure in which nodes needed for backtracking are compressed using a trie, similar to Aho's compression of index strings in [1]. Then, we generalize these results from one mtt to compositions of mtts. Here, the challenge is the existence of intermediate trees. Consider the composition τ of two translations realized by mtts: τ_1 followed by τ_2. To check $(s,t)\in\tau$, we nondeterministically guess an intermediate tree u, and check whether $(s,u)\in\tau_1$ and $(u,t)\in\tau_2$. From the complexity result of single mtts, we know that this can be done in $O(|s|+|u|+|t|)$ space. This can, however, be much larger than $O(|s|+|t|)$; the size $|u|$ of the intermediate tree u can actually be double-exponentially larger than $|s|$ and $|t|$. The basic idea to prove the linear size complexity for compositions of mtts is to bound the sizes of all such intermediate input trees. This is achieved by putting the mtts in certain normal forms such that they do not delete much of their input, in the sense that every output tree t has a corresponding input tree of size only linearly larger than $|t|$. Although our approach is similar to [17], the existence of context parameters and nondeterminism together adds new challenges in every step of the proof. For example, consider the mtt M_{dexp} with the following three rules r_0, r_1, and r_2:

$$\langle q_0,\mathtt{a}(x)\rangle \to \langle q,x\rangle(\langle q,x\rangle(\mathtt{e})) \qquad (r_0)$$
$$\langle q,\mathtt{a}(x)\rangle(y) \to \langle q,x\rangle(\langle q,x\rangle(y)) \qquad (r_1)$$
$$\langle q,\mathtt{e}\rangle(y) \to +(\mathtt{b}(y,y),\mathtt{c}(y,y)) \qquad (r_2)$$

Here, $+$ denotes a nondeterministic choice; e.g., when the state q reads an input node labeled `e`, it generates an output node labeled either `b` or `c`. This mtt takes a tree of form $\mathtt{a}(\mathtt{a}(\cdots\mathtt{a}(\mathtt{e})\cdots))$ as input (with n occurrences of `a`) and generates a full binary tree of height 2^n (note that, without parameters, the height growth can only be linear) with each non-leaf node arbitrarily labeled either `b` or `c`. Therefore, the size of the set of possible output trees is $2^{2^{2^n}}$. To decide whether $(s,t)\in\tau_{M_{\mathrm{dexp}}}$ for given trees s and t, we essentially have to find the correct choice among the triple exponentially many candidates. To address the issue, we (1) instead of solving the membership problem for all mtts, only deal with mtts in the above mentioned non-deleting normal form, and which are linear with respect to the input variables, and (2) exploit the compressed representation of outputs of mtts [19] for manipulating the output set.

2 Preliminaries

The notation used in this paper will be the same as that used in [17], except that we denote the sequential composition by the operator ; instead of $\circ$, and the label of a tree node by $label(t, v)$ instead of $t[v]$. We denote by $pos(t) \subseteq \mathbb{N}^*$ the set of nodes of a tree t.

A *macro tree transducer (mtt)* M is a tuple $(Q, \Sigma, \Delta, q_0, R)$, where Q is the ranked alphabet of *states*, Σ and Δ are the *input* and *output* alphabets, $q_0 \in Q^{(0)}$ is the *initial state*, and R is the finite set of *rules* of the form $\langle q, \sigma(x_1, \ldots, x_k)\rangle(y_1, \ldots, y_m) \to r$ where $q \in Q^{(m)}$, $\sigma \in \Sigma^{(k)}$, and r is a tree in $T_{\Delta \cup (Q \times X_k) \cup Y_m}$. Rules of such form are called $\langle q, \sigma\rangle$-rules, and the set of right-hand sides of all $\langle q, \sigma\rangle$-rules is denoted by $R_{q,\sigma}$. We always assume $\Sigma^{(0)}$ and $\Delta^{(0)}$ (and thus, T_Σ and T_Δ) are non-empty. The rules of M are used as term rewriting rules in the usual way. We denote by $\Rightarrow_M$ the derivation relation of M on $T_{(Q \times T_\Sigma) \cup \Delta}$, and by $u{\downarrow_M}$ the set $\{t \in T_\Delta \mid u \Rightarrow_M^* t\}$. Note that "state-calls" $\langle q, x_i\rangle$ can be nested and therefore different orders of evaluation yield different trees. Unless otherwise specified, we assume the *outside-in* (OI) derivation in which we always rewrite the outermost (= top-most) state calls. By Corollary 3.13 of [12], this order of evaluation yields the same set of output trees as the *unrestricted* order, i.e., the case where no restriction is imposed on the order of evaluation. The *translation realized by* M is the relation $\tau_M = \{(s, t) \in T_\Sigma \times T_\Delta \mid t \in \langle q_0, s\rangle{\downarrow_M}\}$. We denote by MT the class of translations realized by mtts. An mtt is called a top-down tree transducer (tt) if all its states are of rank 0; the corresponding class of translations is denoted by T. We call an mtt *deterministic* (*total*, respectively) if for every $\langle q, \sigma\rangle \in Q \times \Sigma$, the number $|R_{q,\sigma}|$ of rules is at most (at least) one; the corresponding classes of translations are denoted by prefix D ($_t$). An mtt is *linear* (denoted by prefix L) if in every right-hand side of its rules each input variable $x_i \in X$ occurs at most once. The same notation is used for tts; for instance, D_tT denotes the class of translations realized by total deterministic tts.

For a technical reason, we define a slight extension of mtts. We fix the set of choice nodes $C = \{\theta^{(0)}, +^{(2)}\}$ and assume it to be disjoint with other alphabet. An *mtt with choice and failure (mttcf)* M is a tuple $(Q, \Sigma, \Delta, q_0, R)$ defined as for normal mtts, except that the right-hand sides of rules are trees in $T_{\Delta \cup (Q \times X_k) \cup Y_m \cup C}$. The derivation relations ($\Rightarrow_M$ and $\downarrow_M$) and the realized translation (τ_M) are defined similarly as for mtts, with two additional rewrite rules: $+(t_1, t_2) \Rightarrow_M t_1$ and $+(t_1, t_2) \Rightarrow_M t_2$. Thus, $+$ denotes nondeterministic choice and θ denotes failure (because there is no rule for it). Again, we assume the outside-in evaluation order. For a right-hand side r of an mttcf, we say a position $v \in pos(r)$ is *top-level* if for all proper prefixes v' of v, $label(r, v') \in \Delta \cup C$. We say an mttcf is *canonical* if for every right-hand side r and for every top-level position $v \in pos(r)$, $label(r, v) \notin C$. The idea of the choice and failure nodes comes from [12]. There they show that any MT generating trees in T_Δ can be regarded as a D_tMT generating "choice trees" in $T_{\Delta \cup C}$; a choice tree each of the choice trees denotes the set of possible output trees by interpreting θ as the empty set and $+(c_1, c_2)$ as the union of the sets denoted by c_1 and c_2.

3 Complexity of a Single MTT

In this section we show that for any canonical mttcf M having properties called *path-linear* and *non-erasing*, there is a nondeterministic Turing Machine that decides whether a given

pair (s,t) of trees is in τ_M in $O(|s|+|t|)$ space and in polynomial time with respect to $|s|+|t|$. Thus, this "translation membership" problem is in NSPACE(n) and NP. Two previous works on the same membership problem for restricted classes of macro tree transducers – for total deterministic mtts [17] and for nondeterministic mtts without parameters (top-down tree transducers) [3] – both give DSPACE(n) algorithms. First let us briefly explain where the difficulty arises in our case, i.e., with nondeterminism and parameters. For total deterministic mtts, the DSPACE(n) complexity is proved via a reduction to the case of linear total deterministic mtts, and then to attribute grammars (which are deterministic by default), whose output languages are LOG(CFL)-complete and therefore have DSPACE$(\log(n)^2)$ membership test [10]. For nondeterministic tts, the complexity is achieved by a straightforward backtracking-based algorithm; given the input tree s and the output tree t, it generates each possible output of s by simulating the recursive execution of state calls, while comparing with t. The following two facts imply the DSPACE(n) complexity: (1) the depth of the recursion is at most the height of s, and (2) to backtrack we only need to remember for each state call the rule that was applied (which requires constant space). Note that neither (1) nor (2) hold for mtts; the recursion depth can be exponential and the actual parameters passed to each state call must also be remembered for backtracking.

Here we concentrate on a restricted class of mttcfs, namely, *canonical*, *non-erasing*, and *path-linear* mttcfs, which is exactly the class of mttcfs needed later in Section 4, to obtain the complexity result for the output languages of the mtt-hierarchy. For a canonical mtt, we define a right-hand side of a rule to be *non-erasing* if it is *not* in Y. A canonical mttcf is *non-erasing* if the right-hand sides of all its rules are non-erasing. An mttcf is *path-linear* if a subtree of the form $\langle q, x_i\rangle(\cdots\langle p, x_j\rangle(\cdots)\cdots)$ in its rules implies $i \neq j$.

Making MTTCFs Total Deterministic Let M be a canonical, non-erasing, and path-linear mttcf. It is easy to see that we can always construct a total deterministic mttcf M' equivalent to M by simply taking $\langle q, \sigma(\cdots)\rangle(\cdots) \to +(r_1, \cdots, +(r_n, \theta)\cdots)$ for $\{r_1, \ldots, r_n\} = R_{q,\sigma}$. Then, $M' = (Q, \Sigma, \Delta, q_0, R')$ can be seen as a total deterministic *mtt* $N = (Q, \Sigma, \Delta \cup C, q_0, R')$ whose outputs are the choice trees denoting sets of output trees of M. The canonicity and the non-erasure of M implies that in any right-hand side $r \in R'$ and every position $\nu \in pos(r)$ with $label(\nu) \in Y$, there exists a proper prefix ν' of ν with $label(\nu') \neq +$. Path-linearity is preserved from M to M'.

Compressed Representation Our approach is to represent the output choice tree $\tau_N(s)$ in a compact (linear size) structure, and then compare it to the given output tree t. Given a total deterministic mtt N and an input tree $s \in T_\Sigma$, we can, in time $O(|s|)$, calculate a straight-line context-free tree grammar (or SLG, a context-free tree grammar that has no recursion and generates exactly one output) of size $O(|s|)$ that generates $\tau_N(s)$, using the idea of [19]. Rather than repeating the full construction of [19], we here give a direct representation of the nodes of $\tau_N(s)$.

Let N be a total, deterministic, non-erasing, and path-linear mtt with output alphabet $\Delta \cup C$ and let s be an input tree. Let $E = \{(r, \nu) \mid q \in Q, \sigma \in \Sigma, r \in R_{q,\sigma}, \nu \in pos(r)\}$. For a list $e = (r_0, \nu_0)\ldots(r_n, \nu_n)$ of elements of E, we define $orig(e)$ (the *orig*in of e) as $\epsilon.i_0 \ldots i_{k-1}$ where k is the smallest index satisfying $label(r_k, \nu_k) \notin Q \times X$ (or, let $k = n+1$ when all labels are in $Q \times X$) and i_j is the number such that $\langle q, x_{i_j}\rangle = label(r_j, \nu_j)$ for some q. We call e

well-formed if $label(r_i, \nu_i) \in Q \times X$ for every $i < n$, $label(r_n, \nu_n) \in \Delta \cup C$, and $orig(e) \in pos(s)$. Intuitively, e is a partial derivation or a "call stack" of the mtt N. Each node of $\tau_N(s)$ can be represented by a well-formed list, which can be stored in $O(|s|)$ space because its length is at most $1 +$ (height of s) and the size of each element depends only on the size of the fixed mtt, not on $|s|$. Note that e can represent many nodes in $\tau_N(s)$ if the mtt is non-linear in the parameters. For instance, for M_{dexp} from the Introduction and the input tree $s_3 = \mathtt{a}(\mathtt{a}(\mathtt{a}(\mathtt{e})))$, the list $(r_0, \epsilon.1)(r_1, \epsilon.1)(r_1, \epsilon.1)(r_2, \epsilon.1)$ represents all $\mathtt{b}$-nodes at depth 16 of the tree $\tau_{M_{\text{dexp}}}(s_3)$, of which there are 2^8 many. The label *c-label*(e) of the node represented by e is $label(r_n, \nu_n)$. The operation *c-child*(e, i) which calculates the representation of the i-th child of the node represented by e is defined in terms of the following three operations. For a well-formed list $e = (r_0, \nu_0) \dots (r_n, \nu_n)$ with $rank(\text{c-}label(e)) = m$, we define $down_i(e)$ for $1 \leq i \leq m$ as $(r_0, \nu_0) \dots (r_n, \nu_n.i)$. For $e = (r_0, \nu_0) \dots (r_n, \nu_n)$ such that $label(r_n, \nu_n) = y_i \in Y$, we define $pop(e) = (r_0, \nu_0) \dots (r_{n-1}, \nu_{n-1}.i)$. For a list $e = (r_0, \nu_0) \dots (r_n, \nu_n)$ where $label(r_n, \nu_n) = \langle q, x_j \rangle \in Q \times X$, we define $expand(e) = (r_0, \nu_0) \dots (r_n, \nu_n)(r_{n+1}, \epsilon)$ where r_{n+1} is the right-hand side of the unique $\langle q, label(s, orig(e)) \rangle$-rule. Then, the operation *c-child*(e, i) is realized by the following algorithm: first apply $down_i$ to e, then repeatedly apply *pop* as long as possible, and then repeatedly apply *expand* as long as possible. The non-erasure of N ensures that this yields a well-formed list; in the last step, when *expand* cannot be applied to $e = \dots (r_n, \nu_n)$, $label(r_n, \nu_n)$ is obviously not in $Q \times X$ and by non-erasure is not in Y, hence it is in $\Delta \cup C$. Since the length of a well-formed list is bounded by $|s|$ and *pop* (and *expand*, respectively) always decreases (increases) the length of the list by one, each of them are executed at most $|s|$ times in the calculation of *c-child*. Hence, *c-child* runs in polynomial time with respect to $|s|$. Similarly, the representation of the root of $\tau_N(s)$ is obtained in polynomial time by repeatedly applying *expand* as long as possible to $e_0 = (r_0, \epsilon)$ where r_0 denotes the right-hand side of the unique $\langle q_0, label(s, \epsilon) \rangle$-rule. Note that a similar list representation is used in the proof of Theorem 3 in [4].

Matching Algorithm with NP Time Complexity Let $t \in T_\Delta$. Figure 1 shows the nondeterministic algorithm MATCH that decides, given a well-formed list e and a node v of t, whether the set of trees represented by the choice tree at e contains the subtree of t rooted at v. The operations *c-label* and *c-child* are defined as above. The operations *label*, *rank*, and *child* are basic tree operations, assumed to run in polynomial time with respect to $|t|$. If we apply MATCH to the representations of the root nodes of $\tau_N(s)$ and $v = \epsilon$, we can decide whether $(s, t) \in \tau_M$. Since this is the standard top-down recursive comparison of two trees, the correctness of the algorithm should be clear.

```
MATCH (e, v)
1:  while label(e) = + do
2:      e ← c-child(e, k) where k = 1 or 2,
                    nondeterministically chosen
3:  if c-label(e) ≠ label(v) then
4:      return false
5:  else if rank(label(v)) = 0 then
6:      return true
7:  else
8:      for i = 1 to rank(label(v)) do
9:          if not MATCH(c-child(e, i), child(v, i)) then
10:             return false
11:     return true
```

Figure 1: Matching Algorithm

In each nondeterministic computation, MATCH is called once for each node of t. In each call, the while-loop iterates at most $c|s|$ times for a constant c. This is due to non-erasure, i.e., for every Y-node in right-hand sides there exists a non-$+$ ancestor node. If we once *expand* a list for obtaining *c-child*, we never see Y-nodes in right-hand sides (thus never

pop) before seeing some Δ-node. Thus, during the while-loop, the sequence of applied operations must be: first *pop*'s and *down*'s are applied, and then *expand* is applied (if any), and after that no *pop* is applied, i.e., the only operations applied are *expand* or *down*. In other words, it has to be in the regular set $(pop|down)^*(expand|down)^*$. However, since the length of a well-formed list is at most $|s|$, we can continuously *pop* without *expand*ing at most $|s|$ times, and the same for *expand* without *pop*ping. Also, the numbers of continuous *down*'s are bounded by the height of the right-hand sides of the rules of N. Thus, the loop terminates after at most $2 \cdot (1 +$ the maximum height of right-hand sides of $N) \cdot |s|$ iterations. Altogether, the total running time is polynomial in $|s| + |t|$.

Linear Space Complexity The MATCH algorithm takes $O((|s| + \log|t|)|t|)$ space if naively implemented, because in the worst case the depth of recursion is $O(|t|)$ and we have to remember e (which costs $O(|s|)$ space) and v ($O(\log(|t|))$ space at least, depending on the tree node representation) in each step of the recursion. However, note that the lists of nodes share common prefixes! Suppose the root node is represented by $(r_0, \nu_0)(r_1, \nu_1)(r_2, \nu_2)(r_3, \nu_3)$ and its child node is obtained by applying $down_1$, *pop*, and *expand*. Then the child node is of the form $(r_0, \nu_0)(r_1, \nu_1)(r_2, \nu_2')(r_3', \nu_3')$, which shares the first two elements with the root node representation. We show that if we store lists of nodes with common prefixes maximally shared, then, in the case of path-linear mtts, their space consumption becomes $O(|s| + |t|)$. The idea of sharing lists resembles the proof of context-sensitivity of indexed languages [1].

We encode a list of well-formed lists as a tree, written in parenthesized notation on the tape. For example, the list of three lists $[\rho_1\rho_2\rho_3, \rho_1\rho_2\rho_4, \rho_1\rho_5\rho_6]$ is encoded as $\rho_1(\rho_2(\rho_3, \rho_4), \rho_5(\rho_6))$. Since the number of parentheses is $\leq 2n$ and that of commas is $\leq n$ where n denotes the number of nodes, the size of this representation is $O(n)$. When the function MATCH is recursively called, we add the current e to the end of the list. The addition is represented as an addition to the rightmost path. As an example, let $e = \rho_1\rho_5\rho_7\rho_8$. The common prefix $\rho_1\rho_5$ with the current rightmost path $\rho_1\rho_5\rho_6$ is shared, and the suffix $\rho_7\rho_8$ is added as the rightmost child of the ρ_5-node. Then, we have a new tree $\rho_1(\rho_2(\rho_3, \rho_4), \rho_5(\rho_6, \rho_7(\rho_8)))$. Removal of the last list, which happens when MATCH returns, is the reverse operation of addition; the rightmost leaf and its ancestors that have only one descendant leaf are removed. Note that, since by definition a well-formed list cannot be a prefix of any other well-formed lists, each well-formed list always corresponds to a leaf node of the tree. It is straightforward to implement these two operations in linear space and in polynomial time.

Let us consider what happens if we apply this encoding to the output of a *path-linear* mtt. In the algorithm MATCH we only proceed downwards in the trees, i.e., the parameter e' to the recursive calls is always obtained by applying *c-child* several times to the previous parameter e. Thus, the lists $[e_0, e_1, \ldots, e_n]$ of node representations we have to store during the recursive computation always satisfy the relation $e_j \in \textit{c-child}^+(e_i)$ for every $i < j$. Let $e = (r_0, \nu_0) \ldots (r_m, \nu_m)$ and $e' = (r_0', \nu_0') \ldots (r_m', \nu_m')$ be proper prefixes of different elements in the same list satisfying the condition (here we assume that e is taken from the element preceding the one where e' is taken). Then, $orig(e) = orig(e')$ only if $e = e'$. This can be proved by contradiction. Suppose $orig(e) = orig(e')$ and $e \neq e'$, and the j-th elements are the first difference between e and e'. Recall that e' is a prefix of a well-formed list obtained by repeatedly applying *c-child* to another well-formed list, of which e is a prefix. Then it

must be the case that $r_j = r'_j$ (by definition of *expand*, r_j and r'_j are uniquely determined from $(r_0, \nu_0) \dots (r_{j-1}, \nu_{j-1})$ and $(r'_0, \nu'_0) \dots (r'_{j-1}, \nu'_{j-1})$, which are equal) and ν_j is a proper prefix of ν'_j. However, due to path-linearity, the input variable at ν_j and ν'_j must be different, which contradicts $orig(e) = orig(e')$. Therefore, we can associate a unique node in $pos(s)$ with each proper prefix of the lists, which means that the number of distinct proper prefixes is at most $|s|$. Similarly, it can be shown that adding only to the rightmost path is sufficient for maximally sharing all common prefixes. Suppose not, then there must be in the list three nodes of the forms $e_1 = e.(r, \nu).e'_1$, $e_2 = e.(r, \nu').e'_2$, and $e_3 = e.(r, \nu).e'_3$ with $\nu \neq \nu'$ in this order. Note that if this happened, then the prefix $e.(r, \nu)$ would not be shared by the rightmost addition. However, $e_2 \in c\text{-}child^+(e_1)$ implies that ν is a proper prefix of ν', and by $e_3 \in c\text{-}child^+(e_2)$, ν' is a proper prefix of ν, which is a contradiction. Hence, the number of nodes except leaves in the tree encoding equals the number of distinct proper prefixes, which is at most $|s|$. We can bound the number of leaves by $|t|$, the maximum depth of the recursion. So, the size of the tree encoding of a list of nodes is $O(|s| + |t|)$. We can easily remember the whole list of v's in $O(|t|)$ space. Since in the lists $[v_1, \dots, v_n]$, v_{i+1} is always a child node of v_i, we only need to remember the child number for each node. For example, the list $[\epsilon, \epsilon.2, \epsilon.2.1]$ can be encoded as $[\epsilon, 2, 1]$. Thus, we only need $\leq height(t)$ many numbers, each of which is between 1 and the maximal rank of symbols in Δ, which is a constant.

THEOREM 1. *Let M be a canonical, non-erasing, and path-linear mttcf. There effectively exists a nondeterministic Turing Machine which, given any s and t as input, determines whether $(s, t) \in \tau_M$ in $O(|s| + |t|)$ space and in polynomial time with respect to $|s| + |t|$.*

4 Complexity of Compositions of MTTs

As explained in the Introduction, the key idea for obtaining linear-size complexity for compositions of mtts is to bound the size of all intermediate input trees, and this is achieved by putting the mtts into "non-deleting" forms. In the same way as for total deterministic mtts [17], we classify the "deletion" in mtts into three categories – *erasing*, *input-deletion*, and *skipping* (a similar classification without erasing, which is a specific use of parameters, is also used in the case of nondeterministic tts [3]). The resolution of each kind of deletion, however, requires several new techniques and considerations compared to previous work, due to the interaction of nondeterminism and parameters. In the rest of this paper, we first explain how we eliminate each kind of deletion, and then show the main results.

Erasing We first consider "erasing" rules – rules of the form $\langle q, \sigma(\cdots)\rangle(y_1, \dots, y_m) \to y_i$, as defined in Section 3. An application of such a rule consumes one input σ-node without producing any new output symbols; hence it is deleting a part of the input. Note that if the rank of σ is non-zero, then a rule as above is at the same time also input-deleting, which is handled in Section 4. In the case of total deterministic mtts, "non-erasing" is a normal form, i.e., for every total deterministic mtt there is an equivalent one without erasing rules. Unfortunately, we could not find such a normal form for nondeterministic mtts with OI semantics. Note that for OI context-free tree grammars (essentially mtts without input: think of $\langle q, x_i\rangle$ as a nonterminal N_q, or equivalently, think of macro grammars [13] or indexed

grammars [1], with trees instead of strings in right-hand sides), it has been shown [16] that there is *no* non-erasing normal form. The problems is, that "inline expansion", as used to obtain non-erasing total deterministic mtts, generates copies of evaluated trees, which may not correctly model the OI semantics of the original transducer. Therefore, we move from normal mtts to *mtts with choice and failure*. The example above can be represented by an mttcf rule $\langle q_1, \mathtt{a}(x_1, x_2)\rangle \to \langle q_2, x_1\rangle(+(\mathtt{B}, +(\mathtt{C}, \mathtt{A}(\mathtt{B}, \mathtt{C}))))$, for instance. We will show that every mtt can be simulated by a non-erasing mttcf.

LEMMA 2. *Let M be a mtt. There effectively exists a linear tt E and a canonical mttcf M' such that M' is non-erasing and $\tau_E ; \tau_{M'} = \tau_M$. Path-linearity is preserved from M to M'.*

PROOF. The idea is, we first predict all erasing beforehand and annotate each input node by the information of erasing, by using a preprocessing linear tt. Then we replace all erasing state calls (e.g., $\langle q, x_1\rangle(u_1)$ with the rule $\langle q, \ldots\rangle(y_1) \to y_1$) in the right-hand sides of rules with the result of the erasing call (e.g., u_1). Note that we have to deal with nondeterminism. Suppose we have two rules $\langle q, \sigma\rangle(y_1, y_2) \to y_1$ and $\langle q, \sigma\rangle(y_1, y_2) \to y_2$ and a state call $\langle q, x_1\rangle(u_1, u_2)$ in a right-hand side. In order to preserve the nondeterminism, we replace the state call by $+(u_1, u_2)$.

Let $M = (Q, \Sigma, \Delta, q_0, R)$. We define E to be a nondeterministic linear tt with the set of states $P = [Q \to 2^{\{1,\ldots,n\}}] \cup \{p_0\}$ (functions from Q to $2^{\{1,\ldots,n\}}$ where n is the maximum rank of the states of Q, and one distinct state p_0, which is the initial state), the input alphabet Σ, the output alphabet $\Sigma_p = \{(\sigma, p_1, \ldots, p_k)^{(k)} \mid \sigma^{(k)} \in \Sigma, p_i \in P\}$, and the following rules for every $\sigma^{(k)} \in \Sigma$ and $p_1, \ldots, p_k \in [Q \to 2^{\{1,\ldots,n\}}]$: $\langle p, \sigma(x_1, \ldots, x_k)\rangle \to (\sigma, p_1, \ldots, p_k)(\langle p_1, x_1\rangle, \ldots, \langle p_k, x_k\rangle)$ where $p \in \{p_0, (q \mapsto \bigcup\{f(r) \mid \langle q, \ldots\rangle(\ldots) \to r \in R\})\}$ with f recursively defined as follows: $f(y_i) = \{i\}$, $f(\delta(\ldots)) = \emptyset$, and $f(\langle q', x_j\rangle(r_1, \ldots, r_m)) = \bigcup\{f(r_i) \mid i \in p_j(q')\}$. The transducer E modifies the label $\sigma^{(k)}$ of each input node into the form $(\sigma^{(k)}, p_1, \ldots, p_k)$. The annotated information p_i intuitively means "if a state q of M is applied to the i-th child of the node, it will erase and return directly the e-th parameter for $e \in p_i(q)$". If $p_i(q) = \emptyset$ then no erasing will happen. The rule of E is naturally understood if it is read from right to left, as a bottom-up translation. Formally speaking, the following claim holds. It is easily proved by induction on the structure of s.

Claim: (1) For each $s \in T_\Sigma$ and $q \in Q^{(m)}$, there is a unique $p \in P \setminus \{p_0\}$ such that $\langle p, s\rangle\downarrow_E \neq \emptyset$, and $e \in p(q)$ if and only if $y_e \in \langle q, s\rangle(y_1, \ldots, y_m)\downarrow_M$. (2) Let us denote by $[s]$ such p determined by s. The output $s' \in \tau_E(s)$ is unique. For $b \in pos(s) = pos(s')$, $label(s', b) = (label(s, b), [s|_{b.1}], \ldots, [s|_{b.k}])$ where $s|_\nu$ is the subtree of s rooted at the node ν.

Then, let $M' = (Q, \Sigma_p, \Delta, q_0, R')$ with $R' = \{\langle q, (\sigma, p_1, \ldots, p_k)(x_1, \ldots, x_k)\rangle(y_1, \ldots, y_m) \to r' \mid r \in R_{q,\sigma}, r' \in ne(r), r' \notin Y\}$ where the set $ne(r)$ is defined inductively by $ne(r) = \{y_j\}$ if $r = y_j$ and $ne(r) = \{\delta(r'_1, \ldots, r'_l) \mid r'_i \in ne(r_i)\}$ if $r = \delta(r_1, \ldots, r_l)$, and $ne(r) = \bigcup\{ne(r_i) \mid i \in p_j(q')\} \cup \{\langle q', x_j\rangle(nep(r_1), \ldots, nep(r_l))\}$ if $r = \langle q', x_j\rangle(r_1, \ldots, r_l)$, and nep defined as follows: $nep(y_j) = y_j$, $nep(\delta(r_1, \ldots, r_l)) = \delta(nep(r_1), \ldots, nep(r_l))$, and $nep(\langle q', x_j\rangle(r_1, \ldots, r_l)) = +(u_1, +(u_2, \ldots, +(u_z, \theta)\cdots))$ where $\{u_1, \ldots, u_z\} = ne(\langle q', x_j\rangle(r_1, \ldots, r_l))$. The correctness of this construction is proved by induction on the structure of the input tree s. ∎

Input-Deletion The second kind of deletion we investigate is "input-deletion". For instance, if there is the rule $\langle q_0, \mathtt{a}(x_1, x_2)\rangle \to \mathtt{A}(\langle q_0, x_2\rangle)$ for the initial state q_0 and the input

is of the form $\mathrm{a}(t_1, t_2)$, then the subtree t_1 is never used for the output calculation. Although total deterministic mtts can be made *nondeleting* (i.e., to always traverse all subtrees of *every* input tree) by preprocessing with a deleting linear tt [17], it becomes more difficult for nondeterministic mtts. The point is, under nondeterminism, we cannot argue the input-deleting property of each *transducer*. Rather, we can only argue whether each *computation* is input-deleting or not. This is a weaker version of the nondeletion condition used for total deterministic mtts, but it is sufficient for our purpose.

In order to speak more formally, here we define the notion of *computation tree* (following the method of [3], but extending it to deal with accumulating parameters). For any finite set P, we define the ranked alphabet $\underline{P} = \{\underline{p}^{(1)} \mid p \in P\}$. Let $M = (Q, \Sigma, \Delta, q_0, R)$ be an mttcf and $s \in T_\Sigma$. The set $COMP(M, s)$ is the set of trees $comp\langle q_0, \underline{\epsilon}\rangle\downarrow \subseteq T_{\Delta \cup \underline{pos(s)}}$ called *computation trees* (or sometimes, simply *computations*). The derivation $comp\langle q_0, \underline{\epsilon}\rangle\downarrow$ is carried out under the following set of rewriting rules with outside-in derivation: $+(u_1, u_2) \to u_1$, $+(u_1, u_2) \to u_2$, and $comp\langle q, \underline{\nu}\rangle(\vec{y}) \to f_\nu(r)$ for $q \in Q$, $\nu \in pos(s)$, $r \in R_{q, label(s,p)}$ where f_ν is inductively defined as $f_\nu(y_i) = y_i$, $f_\nu(\delta(r_1, \ldots, r_k)) = \underline{\nu}(\delta(f_\nu(r_1), \cdots, f_\nu(r_k))$, and $f_\nu(\langle q', x_j\rangle(r_1, \ldots, r_k)) = comp\langle q', \underline{\nu.j}\rangle(f_\nu(r_1), \cdots, f_\nu(r_k)))$. Intuitively, $COMP(M, s)$ is the set of trees $\langle q_0, s\rangle\downarrow$ where the parent of each Δ-node is a monadic node labeled by the position in the input tree s that generated the Δ-node. For example, the output tree $\underline{\epsilon}(\alpha(\underline{\epsilon.1}(\beta), \underline{\epsilon.2}(\gamma(\underline{\epsilon}(\delta)))))$ means that the α and δ nodes are generated at the root node of the input tree, and the β and γ nodes are generated at the first and the second child of the root node, respectively. Let *delpos* be the translation that removes all $\underline{\nu} \in \underline{pos(s)}$ nodes. It is easily proved by induction on the number of derivation steps that $delpos(\overline{COMP}(M, s)) = \langle q_0, s\rangle\downarrow_M$, i.e., if we remove all $\underline{pos(s)}$ nodes from a computation tree, we obtain an output tree of the original mtt.

We say that a computation tree u is *non-input-deleting* if for every leaf position $\nu \in pos(s)$, there is at least one node in u labeled by $\underline{\nu}$. Note that the rewriting rules of *comp* corresponding to erasing rules do not generate any $\underline{pos(s)}$ node. Thus, non-input-deletion implies that not only some state is applied to every leaf, but also a *non-erasing* rule of some state must be applied.

LEMMA 3. *Let M be a canonical non-erasing mttcf. There effectively exists a linear tt I and a canonical non-erasing mttcf M' such that $\tau_M = \tau_I ; \tau_{M'}$, and for every input-output pair $(s, t) \in \tau_M$, there exists a tree s' and a computation tree $u \in COMP(M', s')$ such that $(s, s') \in \tau_I$, $t = delpos(u)$, and u is non-input-deleting. Also, M' is path-linear if M is.*

PROOF. Let $M = (Q, \Sigma, \Delta, q_0, R)$. We define I as $(\{d\}, \Sigma, \Sigma', d, U)$ where $\Sigma' = \{(\sigma, i_1, \ldots, i_m)^{(m)} \mid \sigma^{(k)} \in \Sigma, 1 \le i_1 < \cdots < i_m \le k\}$ and $U = \{\langle d, \sigma(x_1, \ldots, x_k)\rangle \to (\sigma, i_1, \ldots, i_m)(\langle d, x_{i_1}\rangle, \ldots, \langle d, x_{i_m}\rangle) \mid (\sigma, i_1, \ldots, i_m) \in \Sigma'\}$. The transducer I reads the input and nondeterministically deletes subtrees while encoding the numbers of the non-deleted subtrees in the current label. We define the mttcf M' as $(Q, \Sigma', \Delta, q_0, R')$ where

$$R' = \{\langle q, (\sigma, i_1, \ldots, i_m)(x_1, \ldots, x_m)\rangle(\vec{y}) \to r' \mid r \in R_{q,\sigma} \text{ such that for all top-level calls } \langle q', x_p\rangle \text{ in } r, p \in \{i_1, \ldots, i_m\}, \text{ and } r' \text{ is obtained by replacing } \langle q', x_{i_j}\rangle \text{ in } r \text{ with } \langle q', x_j\rangle \text{ and } \langle q', x_p\rangle \text{ with } \theta \text{ for } p \notin \{i_1, \ldots, i_m\}\}.$$

The transducer M' has basically the same rules as M, except that state calls on 'deleted' children are replaced by θ (or, if it is at the top-level then the rule is removed, to preserve canonicity). It should be easy to see that M' is canonical and non-erasing, and preserves the path-linearity of M. The correctness of the construction is proved by taking as s' the minimal substructure of s that contains all nodes used for calculating t. ∎

Skipping The third and last kind of deletion is "skipping". A computation tree u is *skipping* if there is a node $v \in pos(s)$ labeled by a rank-1 symbol such that no node in u is labeled $\underline{v}$. For a canonical, non-erasing, and path-linear mttcf, skipping is caused by either one of the following two forms of rules. One type is of the form $\langle q, \sigma(x_1)\rangle(y_1, \ldots, y_m) \to \langle q', x_1\rangle(u_1, \ldots, u_v)$ where $u_i \in T_{Y \cup C}$, and such rules are called *skipping*. The others are rules which are not skipping but are of the form $\langle q, \sigma(x_1)\rangle(y_1, \ldots, y_m) \to \langle q', x_1\rangle(u_1, \ldots, u_v)$ where $u_i \in T_{\Delta \cup Y \cup C}$, and such rules are called *quasi-skipping*. Note that, since the mttcf is path-linear, there are no nested state calls in right-hand sides of rules for input symbols of rank 1. Also note that if the root node of the right-hand side of a rule is not a state call, then it must be a Δ-node since the mttcf is canonical and non-erasing. So an application of such a rule generates a Δ-node and thus a $\underline{v} \in \overline{pos(s)}$ node for the current input node. Therefore, it is sufficient to consider only skipping and quasi-skipping rules.

Quasi-skipping rules may cause skipping computations due to parameter deletion: for example, consider the quasi-skipping rule $\langle q, \sigma(x_1)\rangle(y_1) \to \langle q', x_1\rangle(\delta(y_1))$; if there is a q'-rule with a right-hand side not using y_1, then the σ-node may be skipped. For total deterministic mtts [17], there is a "parameter non-deleting" normal form, i.e., every total deterministic mtt is equivalent to one that uses all parameters in the right-hand sides of its rules, and thus only skipping rules (without choice nodes) were considered there. Unfortunately, as for non-erasure, we could not find such a normal form for nondeterministic mtts. Instead, we add some auxiliary skipping rules to mttcfs, so that we only need to consider skipping rules. Note that quasi-skipping rules cause skipping computations only when parameters are deleted. The idea is, if a parameter in some rule is never used for a computation, then replacing the parameter by a failure symbol θ does not change the translation, and moreover, such replacement changes a quasi-skipping rule into a skipping rule.

LEMMA 4. *Let M be an canonical, non-erasing, and path-linear mttcf. There effectively exists a linear tt S and a canonical, non-erasing, and path-linear mttcf M' such that (1) $\tau_S ; \tau_{M'} = \tau_M$ and (2) for every input tree s and non-input-deleting computation tree $u \in COMP(M, s)$, there exists a tree s' and a computation tree u' such that $s' \in \tau_S(s)$, $u' \in COMP(M', s')$, $delpos(u') = delpos(u)$, and u' is both non-input-deleting and non-skipping.*

PROOF. First, we construct a new set $\overline{R}$ of skipping rules from quasi-skipping rules of M, by replacing all Δ nodes in each quasi-skipping rule by the failure symbol θ. We then prove that adding rules in $\overline{R}$ to M does not change the translation, and moreover, the addition implies that all skipping computations of M have a derivation that does not apply quasi-skipping rules to skipped nodes. Thus we may assume that all skipping computations are caused by skipping rules, and hence we can straightforwardly extend the proofs for total deterministic mtts [17] and nondeterministic tts [3]. ∎

LEMMA 5. *Let $M = (Q, \Sigma, \Delta, q_0, R)$ be an mttcf, s an input tree, and u a non-input-deleting, non-skipping computation tree in $COMP(M, s)$ with $delpos(u) = t$. Then $|s| \leq 2|t|$.*

PROOF. Since u is non-input-deleting and non-skipping, for all nodes $v \in pos(s)$ of rank zero or one, there exists a node labeled $\underline{v}$ in u, and by definition of computation trees, its child node is labeled by a symbol in Δ. Thus, $leaves(s) + rank1nodes(s) \leq |t|$ where $leaves(s)$ is the number of leaf nodes of s and $rank1nodes(s)$ is the number of nodes of s labeled by rank-1 symbols. Since $|s| \leq 2 \times leaves(s) + rank1nodes(s)$ (this holds for any tree s), we have $|s| \leq 2|t|$ as desired. ∎

Main Results

LEMMA 6. *Let $\mathcal{K} \in \{\mathrm{NSPACE}(n), \mathrm{NP}\}$ and F a class of $\mathcal{K}$ languages effectively closed under* LT. *Then* $\mathrm{LMT}(F)$ *and* $\mathrm{T}(F)$ *are also in $\mathcal{K}$.*

PROOF. Let M be a linear mtt or a tt. Note that in both cases, M is path-linear. First, we make it non-erasing; by Lemma 2, there exist a linear tt E and a canonical, non-erasing, and path-linear mttcf M_1 such that $\tau_E ; \tau_{M_1} = \tau_M$. Next, we make each computation non-input-deleting; by Lemma 3, there exist a linear tt I and a canonical, non-erasing, and path-linear mttcf M_2 such that $\tau_I ; \tau_{M_2} = \tau_{M_1}$. For every $(s_1, t) \in \tau_{M_1}$, there is an intermediate tree s_2 and a non-input-deleting computation $u \in COMP(M_2, s_2)$ such that $(s_1, s_2) \in \tau_I$ and $delpos(u) = t$. Then, we make each computation non-skipping; by Lemma 4, there exist a linear tt S and a canonical, non-erasing, and path-linear mttcf M_3 such that $\tau_S ; \tau_{M_3} = \tau_{M_2}$. For every non-input-deleting computation $u \in COMP(M_2, s_2)$, there is an intermediate tree s_3 and a non-input-deleting, non-skipping computation $u' \in COMP(M_3, s_3)$ such that $(s_2, s_3) \in \tau_S$ and $delpos(u') = delpos(u)$. Altogether, we have $\tau_E ; \tau_I ; \tau_S ; \tau_{M_3} = \tau_M$, and for every $(s, t) \in \tau_M$ there exists a tree s_3 such that $(s, s_3) \in \tau_E ; \tau_I ; \tau_S$ and a non-input-deleting, non-skipping computation $u' \in COMP(M_3, s_3)$ such that $delpos(u') = t$. By Lemma 5, $|s_3| \leq 2|t|$.

Let L be a language in F. To check whether $t \in \tau_M(L)$, we nondeterministically generate every tree s' of size $|s'| \leq 2|t|$ and for each of them, test whether $(s', t) \in \tau_{M_3}$ and $s' \in (\tau_E ; \tau_I ; \tau_S)(L)$. By Theorem 1, the former test can be done nondeterministically in $O(|s'| + |t|) = O(|t|)$ space and polynomial time with respect to $|t|$. By the assumption that F is closed under LT, the language $(\tau_E ; \tau_I ; \tau_S)(L)$ is also in $\mathcal{K}$. Thus the latter test is in complexity $\mathcal{K}$ with respect to $|s'| = O(|t|)$. ∎

Note that, for T, the result is known to hold also for $\mathcal{K} = \mathrm{DSPACE}(n)$ (Theorem 1 of [3]).

LEMMA 7. *Let $\mathcal{K} \in \{\mathrm{NSPACE}(n), \mathrm{NP}\}$ and F a class of $\mathcal{K}$ languages effectively closed under* LT. *Then* $\mathrm{MT}(F)$ *is also in $\mathcal{K}$ and effectively closed under* LT.

PROOF. The closure under LT immediately follows from the following known results: $\mathrm{MT} = \mathrm{D_tMT} ; \mathrm{T}$ (Corollary 6.12 of [12]), $\mathrm{T} ; \mathrm{LT} = \mathrm{D_tQREL} ; \mathrm{T}$ (Lemma 2.11 of [9]), and $\mathrm{D_tMT} ; \mathrm{D_tQREL} \subseteq \mathrm{D_tMT}$ (Lemma 11 of [11]). By Lemma 2.11 of [9] and Theorem 2.9 of [8], $\mathrm{T} ; \mathrm{LT} \subseteq \mathrm{LT} ; \mathrm{T}$, which implies that $\mathrm{T}(F)$ is also closed under LT. By the decomposition $\mathrm{MT} = \mathrm{D_tT} ; \mathrm{LMT}$ (page 138 of [12]), $\mathrm{MT}(F) \subseteq \mathrm{LMT}(\mathrm{T}(F))$. By applying Lemma 6 twice, $\mathrm{LMT}(\mathrm{T}(F))$ is in $\mathcal{K}$. ∎

By REGT, we denote the class of *regular tree languages* [15].

THEOREM 8. $\mathrm{MT}^*(\mathrm{REGT}) \subseteq \mathrm{NSPACE}(n) \cap \mathrm{NP}$-complete.

PROOF. The class REGT is closed under LT (Propositions 16.5 and 20.2 of [15]) and is in $\mathrm{NSPACE}(n) \cap \mathrm{NP}$ (see, e.g., [15]). By induction on $k \geq 1$ it follows from Lemma 7 that $\mathrm{MT}^k(\mathrm{REGT})$ is in $\mathrm{NSPACE}(n)$ and NP. As noted in the Introduction, NP-hardness follows from [22] and the fact that the indexed languages, which are equivalent to the yields of context-free-tree languages under OI-derivation, are in $\mathrm{MT}^2(\mathrm{REGT})$. ∎

Although we only have considered outside-in evaluation order up to here, the previous result holds for compositions of mtts in *inside-out* evaluation order. This is because $\mathrm{MT}^*_{\mathrm{IO}} = \mathrm{MT}^*$ by Theorem 7.3 of [12], where $\mathrm{MT}_{\mathrm{IO}}$ denotes the class of translations realized by mtts in inside-out evaluation order. The *yield* translation, which translates a tree into its string of leaf labels from left to right (seen as a monadic tree), is in $\mathrm{D_tMT}$. Therefore the output string languages $\mathit{yield}(\mathrm{MT}^*(\mathrm{REGT}))$ of mtts are also in the same complexity class as Theorem 8. Especially, this class contains the IO- and OI- hierarchies [6]. Note that the IO-hierarchy is in $\mathrm{D_tMT}^*(\mathrm{REGT})$ and hence in $\mathrm{DSPACE}(n)$ by Corollary 17 of [17]. The first level of the OI-hierarchy are the indexed languages [13] which are NP-complete [22].

COROLLARY 9. *The OI-hierarchy is in* $\mathrm{NSPACE}(n) \cap \mathrm{NP}$-complete.

Thanks This work was partly supported by the Japan Society for the Promotion of Science.

References

[1] A. V. Aho. Indexed grammars—an extension of context-free grammars. *J. ACM*, 15:647–671, 1968.
[2] P. R. J. Asveld. Time and space complexity of inside-out macro languages. *Int. J. Comp. Math.*, 10:3–14, 1981.
[3] B. S. Baker. Generalized syntax directed translation, tree transducers, and linear space. *SIAM J. Comp.*, 7:376–391, 1978.
[4] G. Busatto, M. Lohrey, and S. Maneth. Efficient memory representation of XML document trees. *Inf. Syst.*, 33:456–474, 2008.
[5] B. Courcelle. Monadic second-order definable graph transductions: A survey. *TCS*, 126:53–75, 1994.
[6] W. Damm. The IO- and OI-hierarchies. *TCS*, 20:95–207, 1982.
[7] F. Drewes and J. Engelfriet. Decidability of the finiteness of ranges of tree transductions. *Inf. and Comp.*, 145:1–50, 1998.
[8] J. Engelfriet. Bottom-up and top-down tree transformations – a comparison. *Math. Sys. Th.*, 9:198–231, 1975.
[9] J. Engelfriet. Top-down tree transducers with regular look-ahead. *Math. Sys. Th.*, 10:289–303, 1977.
[10] J. Engelfriet. The complexity of languages generated by attribute grammars. *SIAM J. Comp.*, 15:70–86, 1986.
[11] J. Engelfriet and S. Maneth. Output string languages of compositions of deterministic macro tree transducers. *J. Comp. Sys. Sci.*, 64:350–395, 2002.
[12] J. Engelfriet and H. Vogler. Macro tree transducers. *J. Comp. Sys. Sci.*, 31:71–146, 1985.
[13] M. J. Fischer. *Grammars with Macro-Like Productions*. PhD thesis, Harvard University, Cambridge, 1968.
[14] Z. Fülöp and H. Vogler. *Syntax-Directed Semantics: Formal Models Based on Tree Transducers*. Springer-Verlag, 1998.
[15] F. Gécseg and M. Steinby. Tree languages. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages, Vol 3: Beyond Words*, pages 1–68. Springer-Verlag, 1997.
[16] B. Leguy. Grammars without erasing rules. the OI case. In *Trees in Algebra and Programming*, 1981.
[17] S. Maneth. The complexity of compositions of deterministic tree transducers. In *FSTTCS*, 2002.
[18] S. Maneth, A. Berlea, T. Perst, and H. Seidl. XML type checking with macro tree transducers. In *PODS*, 2005.
[19] S. Maneth and G. Busatto. Tree transducers and tree compressions. In *FoSSaCS*, 2004.
[20] T. Milo, D. Suciu, and V. Vianu. Typechecking for XML transformers. In *PODS*, 2000.
[21] T. Perst and H. Seidl. Macro forest transducers. *Information Processing Letters*, 89:141–149, 2004.
[22] W. C. Rounds. Complexity of recognition in intermediate-level languages. In *FOCS*, 1973.

Foundations of Software Technology and Theoretical Computer Science (Bangalore) 2008.
Editors: R. Hariharan, M. Mukund, V. Vinay; pp 256-267

STCON in Directed Unique-Path Graphs

Sampath Kannan[*], **Sanjeev Khanna**[†], **Sudeepa Roy**[‡]
University of Pennsylvania
Philadelphia, PA, USA
{kannan,sanjeev,sudeepa}@cis.upenn.edu

ABSTRACT. We study the problem of space-efficient polynomial-time algorithms for *directed st-connectivity* (STCON). Given a directed graph G, and a pair of vertices s, t, the STCON problem is to decide if there exists a path from s to t in G. For general graphs, the best polynomial-time algorithm for STCON uses space that is only slightly sublinear. However, for special classes of directed graphs, polynomial-time poly-logarithmic-space algorithms are known for STCON. In this paper, we continue this thread of research and study a class of graphs called *unique-path graphs with respect to source* s, where there is at most one simple path from s to any vertex in the graph. For these graphs, we give a polynomial-time algorithm that uses $\tilde{O}(n^{\varepsilon})$ space for any constant $\varepsilon \in (0, 1]$. We also give a polynomial-time, $\tilde{O}(n^{\varepsilon})$-space algorithm to *recognize* unique-path graphs. Unique-path graphs are related to configuration graphs of unambiguous log-space computations, but they can have some directed cycles. Our results may be viewed along the continuum of sublinear-space polynomial-time algorithms for STCON in different classes of directed graphs - from slightly sublinear-space algorithms for general graphs to $O(\log n)$ space algorithms for trees.

1 Introduction

We study the *directed st-connectivity (STCON)* problem, where given a directed graph G, a source vertex s and a terminal vertex t, we are interested in finding whether there is a path from s to t in G. The STCON problem can be solved in polynomial time using standard search algorithms (for eg. Depth First Search (DFS) or Breadth First Search (BFS)). These algorithms run in $O(m + n)$ time and use $O(n \log n)$ space on a graph with n vertices and m edges. Improving the space complexity of STCON is a well-studied and fundamental problem. The best known deterministic upper bound is given by Savitch's theorem [18], which solves STCON in $O(\log^2 n)$ space. On the other hand, STCON is known to be NL-complete [13]; i.e. giving an $O(\log n)$ space algorithm will imply $L = NL$. A comprehensive survey on the complexity of STCON can be found in [19].

An interesting related question is the time-space trade-off involved in solving STCON [8]. Savitch's theorem uses $O(\log^2 n)$ space, but takes super-polynomial ($n^{O(\log n)}$) time. DFS or BFS takes linear time but its standard implementation requires $O(n \log n)$ space. The only algorithm known till date that breaks the linear space barrier but takes polynomial time is due to Barnes *et al* [6]. This algorithm uses $n/2^{\Theta(\sqrt{\log n})}$ space to solve STCON in any directed graph.

[*]Supported by NSF Award CT-ISG 0716172.
[†]Supported in part by a Guggenheim Fellowship, an IBM Faculty Award, and by NSF Award CCF-0635084.
[‡]Supported by NSF Award IIS-0803524.

FSTTCS 2008
IARCS Annual Conference on
Foundations of Software Technology and Theoretical Computer Science
http://drops.dagstuhl.de/opus/volltexte/2008/1758

On the other hand, the space complexity of the undirected counterpart of STCON, namely USTCON, has been recently resolved by Reingold [15], who showed that USTCON $\in L$. USTCON can also be solved in randomized $O(\log n)$ space and $O(mn)$ time using random walks [1].

STCON has been studied in more restricted models of computation than the Turing machine model. For example in the *JAG (Jumping Automaton for Graphs)* model proposed by Cook and Rackoff, it has been shown that STCON has a lower bound of $\Omega(\log^2 n/\log\log n)$ on space complexity [10]. The same lower bound has also been shown by Berman and Simon [7] for the Randomized JAG model. Poon further defined a stronger *Node Named JAG (NNJAG)* model [14] and showed a time-space lower bound of $T = 2^{\Omega(\frac{\log^2(n\log n/S)}{\log\log n})} \times \sqrt{nS/\log n}$ on both the JAG and the NNJAG model [12] (where T denotes the time and S denotes the space), underscoring the difficulty in designing polynomial-time sublinear-space algorithms for STCON.

One important complexity class in the study of STCON and reachability problems is *Unambiguous Log-Space (UL or USPACE($\log n$))* [5]. This is a subclass of NL and characterizes the class of problems accepted by logarithmic-space-bounded non-deterministic Turing Machines with at most one accepting computation path for each input. Though this appears to be a strong restriction on the computation power of non-deterministic log-space machines, UL/poly has been shown to be identical to NL/poly in [17]. Further subclasses of UL with stronger requirements in terms of uniqueness of the computation path between two configurations have been defined [9]. The complexity class *Reach Unambiguous Log-Space (RUSPACE($\log n$))* requires any two configurations reachable from the start configuration to have a unique computation path in between them and in *Strong Unambiguous Log-Space (StUSPACE($\log n$))* any two configurations should have at most one path between them. Clearly StUSPACE($\log n$) $\subseteq$ RUSPACE($\log n$) $\subseteq$ UL. In RUSPACE($\log n$) the set of vertices reachable from the *source* vertex forms a tree whereas in StUSPACE($\log n$) the set of vertices reachable from *any* vertex forms a tree. Configuration graphs for StUSPACE($\log n$) have also been described as *Mangroves* in [4].

Though both the space complexity and the time-space tradeoff for STCON have not been resolved till date for general directed graphs, these problems have been studied extensively on interesting subclasses of directed graphs. Given an oracle to access the set of incoming edges and outgoing edges for a node in the graph, the STCON problem can be easily solved in polynomial time using $O(\log n)$ space on a tree. Allender *et al* have given a polynomial-time $O(\log^2 n/\log\log n)$ space algorithm to solve the STCON problem on StUSPACE($\log n$) that also works for RUSPACE($\log n$) [4]. On planar DAGs with single source STCON has been shown to be solvable using $O(\log n)$ space [3]. But none of these graph families allow the presence of cycles. A recent survey of Allender [2] highlights the results on the complexity of reachability in UL and its subclasses and on other special subclasses of directed graphs. Also Reingold's technique has been generalized in [16] to solve STCON in $O(\log n)$ space for *regular directed graphs*, where there is a value d such that the in-degrees and out-degrees of all vertices are d.

In this paper we define a subclass of directed graphs that we call *unique-path graphs with respect to source vertex s*. These graphs are defined by the existence of at most one

simple path from s to any vertex in the graph. We will show later that this class of graphs is characterized by the absence of forward or cross edges with respect to any DFS-tree rooted at a vertex reachable from the source vertex. But some back edges may be present, i.e., we allow the presence of some cycles in these graphs.

Configuration graphs for UL and its subclasses are closely related to unique-path graphs. Unique-path graphs strictly contain trees and configuration graphs for StUSPACE($\log n$) and RUSPACE($\log n$), since we allow some cycles. But, a configuration graph for UL is not necessarily a unique-path graph, and vice-versa. Figure 1 shows examples of unique-path graphs and configuration graphs of different subclasses of UL.

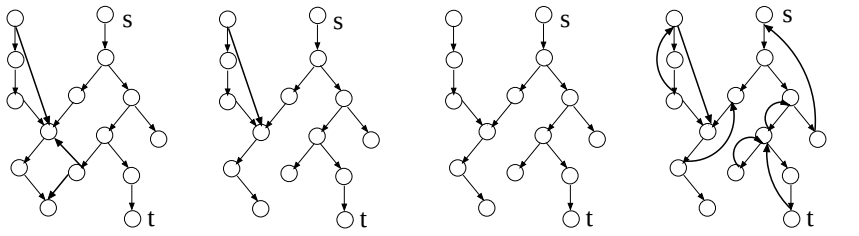


Figure 1: Examples of unique-path graphs and configuration graphs of UL and its subclasses

Our Results. As noted above, upper bounds on the space complexity of polynomial-time algorithms for STCON ranges from $O(\log n)$ (on trees), $O(\log^2 n/\log\log n)$ (on mangroves) to $n/2^{\Theta(\sqrt{\log n})}$ (on general directed graphs). In this paper, we show that for any $\varepsilon \in (0,1]$, the STCON problem can be solved in unique-path graphs in $n^{O(\frac{1}{\varepsilon})}$ time using $\tilde{O}(\frac{n^\varepsilon}{\varepsilon})$ space [§]; this gives a polynomial time algorithm that uses $\tilde{O}(n^\varepsilon)$ space for any constant ε. We also show that we can *recognize* unique-path graphs in $\tilde{O}(\frac{n^\varepsilon}{\varepsilon})$ space and $n^{O(\frac{1}{\varepsilon})}$ time.

Our algorithm uses a sublinear-space implementation of DFS in unique-path graphs. The standard implementation of DFS uses linear space for two purposes: (i) to maintain a stack for backtracking from a vertex v after exploring all vertices reachable from it, and (ii) to keep track of all vertices already visited and avoid rediscovering them. We show that, in unique-path graphs, these purposes can be served by maintaining a sublinear-space data structure which we call *landmark vertices*. We first give an $\tilde{O}(\sqrt{n})$-space polynomial-time algorithm for STCON in unique-path graphs. Extending our techniques further, we obtain an algorithm which improves the space requirement to $\tilde{O}(\frac{n^\varepsilon}{\varepsilon})$.

Organization. The rest of the paper is organized as follows. In Section 2 we define a unique-path graph and discuss some useful properties of unique-path graphs. In Section 3 we give an $\tilde{O}(\frac{n^\varepsilon}{\varepsilon})$-space $n^{O(\frac{1}{\varepsilon})}$-time algorithm for any $\varepsilon \in (0,1]$ to solve STCON in unique-path graphs. In Section 4 we show how to decide if an input directed graph is a unique-path graph in $\tilde{O}(\frac{n^\varepsilon}{\varepsilon})$ space and $n^{O(\frac{1}{\varepsilon})}$ time. Section 5 contains conclusions and some directions for future work.

[§] $\tilde{O}(f(n))$ denotes $O(f(n)\log^k n)$, for some constant k.

2 Preliminaries

Given a directed graph G, we will use $V(G)$ and $E(G)$ to denote the set of vertices and edges, respectively, in G. We assume that there are no self loops or parallel edges in the graph. A path where no intermediate vertices is repeated is called a *simple path*; a *simple cycle* is defined similarly. Two simple paths p_1, p_2 are called *distinct* if they differ in at least one edge. Next we define a *unique-path graph*.

DEFINITION 1. *A directed graph G is a* unique-path graph with respect to a source vertex s *if there is at most one simple path from s to any vertex $v \in V(G)$.*

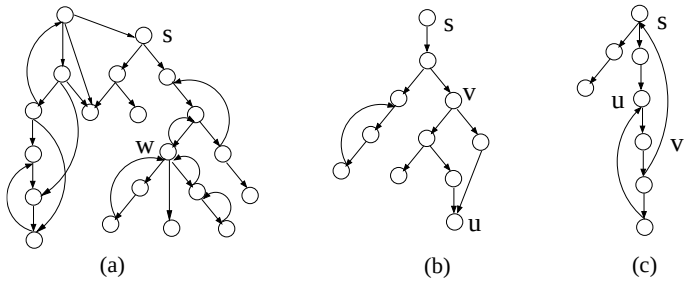


Figure 2: Examples of unique-path graphs ((a)) and non-unique-path graphs ((b) and (c))

It is easy to see that a directed graph G is a unique-path graph with source vertex s iff there is at most one simple path between any two distinct vertices $u, v \in V(G)$, where both u and v are reachable from s. If a directed graph is not a unique-path graph, we call it a *non-unique-path graph*. Examples of unique-path and non-unique-path graphs are shown in Figure 2. The graph in Figure 2(a) is a unique-path graph with source vertex s, but the graphs in Figure 2(b) and (c) are non-unique-path graphs - in both cases there are two distinct simple paths from s to u. The definition of unique-path graphs does not put any restriction on the vertices which are not reachable from the source vertex s; they can have arbitrary number of simple paths between them. Even there can be multiple simple paths from a vertex u to a vertex v where v is reachable from s but u is not. Also note that while there is at most one simple path between any pair of *distinct* vertices reachable from s, a vertex reachable from s (such as w in Figure 2(a)) can lie on many different simple cycles.

For any vertex x in a graph G, we denote by $N^-(x)$ (resp. $N^+(x)$) the set of vertices that have an out-going edge to (in-coming edge from) x in G. We assume that the input graph G is represented in an adjacency-list format, where for each vertex $x \in V(G)$, $N^+(x)$ and $N^-(x)$ are specified as lists. Given $u, v \in N^+(x)$ (or $N^-(x)$) u is called a *successor* of v if u immediately follows v in the list. We assume access to the incoming and the outgoing edges of a node v and therefore the neighbors of v via queries to an *oracle* that answers as follows: given vertices v and w, the oracle can answer if $w \in N^+(v)$ (or in $N^-(v)$), i.e., we can check if (v, w) (or (w, v)) $\in E(G)$. Also we can query the oracle to return the successor (if any) of w in $N^+(v)$ or in $N^-(v)$.

2.1 Properties of Unique-Path Graphs

The algorithm we present to solve STCON in unique-path graphs relies on depth first search (DFS) from the source vertex s. We therefore begin by making a few observations about DFS in unique-path graphs. DFS from a vertex $v \in V(G)$ generates a tree called the *DFS-tree* with v as the *root* of the tree. The edges used in the tree are called *tree edges*. Apart from tree edges, DFS on general directed graphs yield three other types of edges: *back edges, forward edges and cross edges* (see, for instance [11]). The *parent* of a vertex v is the vertex $u \in N^-(v)$ such that (u, v) is a tree edge and will be denoted by $\pi(v)$. Lemma 2 states a necessary and sufficient condition for a directed graph G to be a unique-path graph with respect to a vertex s and Lemma 3 describes the structure of back edges in a unique-path graph. The proofs are easy and are omitted due to space constraint.

LEMMA 2. *A directed graph G is a unique-path graph with respect to $s \in V(G)$ iff DFS invoked from any vertex reachable from s does not produce any forward or cross edges.*

For a back edge (u, v) in a DFS-tree, let $\text{SPAN}(u, v)$ denote the set of vertices on the path in the DFS-tree from v to u including v and u.

LEMMA 3. *Let G be a unique-path graph. Let $(u, v), (x, y) \in E(G)$ be back edges in the DFS-tree with w as the root, where w is reachable from s. Then $|\text{SPAN}(u, v) \cap \text{SPAN}(x, y)| \leq 1$.*

3 Algorithm for STCON in Unique-path Graphs

We assume that G is a directed unique-path graph with respect to source s in this section. We will solve STCON in unique-path graphs by implementing DFS from s in polynomial time using $O(n^\varepsilon)$ space. A typical implementation of DFS relies on remembering the set of vertices that have already been visited (to avoid *rediscovering* previously visited vertices), and remembering the current exploration path for *backtracking* from a vertex v after all vertices reachable from v have been visited (using a stack). Both these tasks can be accomplished using linear space. We show that, for unique-path graphs, these steps can be implemented in polynomial time using sublinear space by maintaining some sparse auxiliary information.

Our final aim is to design an $O(n^\varepsilon)$ space polynomial-time algorithm to implement STCON in unique-path graphs for a constant $\varepsilon \in (0, 1]$ [¶]. In Section 3.1 we give an $O(\sqrt{n})$ space polynomial-time algorithm to present our techniques. In Section 3.2 we will use our techniques recursively to get an $O(n^\varepsilon)$-space polynomial-time algorithm. We will assume that the oracle takes one unit of time to answer any query, though we get a polynomial-time algorithm as long as the time taken by the oracle is bounded by a polynomial. We will refer to the last vertex discovered by the DFS with an unfinished DFS call as the *current vertex*, and the path using the tree edges from s to the current vertex as the *active path*.

3.1 An $O(\sqrt{n})$-Space Algorithm

We prove the following theorem in this section.

[¶]From now on, "$O(f(n))$ space" will refer to the space needed to store $O(f(n))$ words; the bit complexity will be $O(f(n) \log n) = \tilde{O}(f(n))$.

Algorithm 1 An $O(\sqrt{n})$-space, polynomial-time algorithm for STCON on a unique-path graph G with source vertex s and terminal vertex t

CURRENT:
– Suppose the control of the DFS is at the current vertex x (initially $x = s$).
– Either DFS has backtracked to x from some vertex $v \in N^+(x)$, or x is a newly discovered vertex.
if DFS has backtracked to x from v **then**
  – Ask the oracle to return the successor of v in $N^+(x)$.
else {x is a newly discovered vertex}
  – Ask the oracle to return the first vertex in $N^+(x)$.
end if
NEXT:
if the oracle returns that there are no more vertices in $N^+(x)$ **then** {either the DFS has backtracked from the last child of x in $N^+(x)$ or $N^+(x)$ is empty)}
  if x is same as the source vertex s **then** {the search from s is complete}
    – Exit with the answer 'there are no paths from s to t in G'.
  end if
  – Perform the *backtrack step* for x to reach $u = \pi(x)$.
  – Set $x = u$, pass the control to (new) x and jump to Step CURRENT.
else
  – The oracle returns y as the next vertex in $N^+(x)$.
  – Perform the *discovery step* for the edge (x, y).
  if (x, y) is a back edge **then** {y has been visited before,}
    – Ask the oracle to return the successor of y in $N^+(x)$ and jump to Step NEXT.
  else {y is a newly discovered vertex}
    if y is same as the terminal vertex t **then**
      – Exit with the answer 'there is a path from s to t in G'.
    end if
    – Set $x = y$, pass the control to (new) x and jump to Step CURRENT.
  end if
end if

THEOREM 4. *STCON is solvable in $O(mn + m^2\sqrt{n})$ time with $O(\sqrt{n})$ space in unique-path graphs.*

Overview of the Algorithm Algorithm 1 describes how STCON in unique-path graphs can be implemented in $O(\sqrt{n})$ space and polynomial time. It relies on a sublinear-space implementation of two key subroutines. The first subroutine is to *backtrack from a vertex x*, i.e., to return the control to the parent $\pi(x)$ once the DFS finishes at x. The second subroutine is the *discovery step for an edge (x, y)*, which is called from a current vertex x to determine if an edge (x, y) being considered by the DFS is a back edge. Now in order to complete the description of the algorithm, it suffices to describe how we implement the backtrack and the discovery steps. Note that Algorithm 1 and later the procedures for the backtrack and the discovery steps always start the search on G from a vertex reachable from s. Thus we only

need the unique-path property of the vertices reachable from the source vertex s and do not have any restriction on the rest. First we introduce the notion of L-bounded DFS, which is used as a subroutine in the procedures for the backtrack and the discovery steps.

L-bounded DFS

DEFINITION 5. *A DFS search is called* L-bounded *if it backtracks whenever the length of the active path exceeds* L.

For a unique-path graph, if we store the entire active path, the backtrack and the discovery steps can be easily implemented. The next lemma follows from the above observation; we omit the proof due to space constraint.

LEMMA 6. *Given a unique-path graph* G *with respect to source* s, *a vertex* $v \in V(G)$ *reachable from* s *and and an integer* L, *an* L*-bounded DFS from* v *can be implemented in* $O(L)$ *space and* $O(n + mL)$ *time. Moreover, it visits every vertex within distance* L *from* v, *and does not visit any vertex at distance greater than* L *from* v.

We note that an $O(\sqrt{n})$ space, polynomial-time algorithm for STCON in unique-path graphs can be obtained from the above lemma with the approach of [6]. But there is no obvious way of improving the space complexity beyond $O(\sqrt{n})$ using this approach. We present here another approach for solving STCON in unique-path graphs in $O(\sqrt{n})$ space. This will be the starting point to obtain an algorithm that reduces the space requirement to $O(n^{\varepsilon})$.

Next we describe the implementation of the backtrack and discovery steps. The main idea in implementing these steps is maintaining *landmark vertices* which are a few evenly spaced vertices on the active path from s to the current vertex x. The landmark vertices will be denoted by z_i, $i = 0, 1, \cdots$, where the landmark vertex z_i is at distance $i\lfloor\sqrt{n}\rfloor$ from s along the current active path ($s = z_0$); i is called the *index* of the landmark vertex z_i. We will consider the current vertex x as an additional landmark vertex z_p, where $z_0, z_1, \cdots, z_{p-1}$ is the the set of landmark vertices maintained along the active path to x. Since it is easy to maintain the length of the active path from s to the current vertex x in Algorithm 1, the landmark vertices can be maintained by a simple modification of the algorithm. The space needed to maintain the landmark vertices is $O(\sqrt{n})$, because the number of landmark vertices is $O(\sqrt{n})$. As in standard DFS, Algorithm 1 performs $O(n)$ backtrack and $O(m)$ discovery steps; thus to implement the whole algorithm in $O(\sqrt{n})$ space and polynomial time it suffices to show that the backtrack and the discovery steps can be implemented in $O(\sqrt{n})$ space and polynomial time.

Backtrack Step

Let x be the current vertex and let $v_1, v_2, ..., v_q$ be the vertices in $N^-(x)$. Suppose $v_i = \pi(x)$ in the DFS-tree. Since G is a unique-path graph with respect to source s, the unique simple path from s to x is through the edge (v_i, x). Recall that if $z_0, \cdots, z_p$ are the landmark vertices then current vertex $x = z_p$ and z_{p-1} is the previous landmark vertex.

Procedure 2 Procedure to implement the backtrack step from the current vertex x

1: – Let $v_1, v_2, ..., v_q$ be the vertices in $N^-(x)$.
2: **for** each $v_j \in N^-(x)$ **do**
3: – Perform a $\sqrt{n}$-bounded DFS from z_{p-1} in the graph $G - (v_j, x)$.
4: **if** $z_p = x$ is not reached **then**
5: – Return v_j as $\pi(x)$.
6: **end if**
7: **end for**

LEMMA 7. *In the graph $G - (v_i, x)$ the current vertex $z_p = x$ is not discovered by a $\sqrt{n}$-bounded DFS from z_{p-1} iff v_i is the parent of x in the original DFS-tree.*

PROOF. (if) Assume $v_i = \pi(x)$ and there is a path from z_{p-1} to x in the graph $G - (v_i, x)$. Then there are two distinct paths from z_{p-1} to x, one uses the tree edge (v_i, x) and the other does not. Thus there are two distinct paths from s to x - this contradicts that G is a unique-path graph. (only if) Let $v_j \in N^-(x)$ and $v_j \neq \pi(x)$. As the landmark vertices are placed $\sqrt{n}$ distance apart along the active path, by Lemma 6, x will be discovered by a $\sqrt{n}$-bounded DFS from the last landmark vertex z_{p-1} in the graph $G - (v_j, x)$. ∎

The number of $\sqrt{n}$-bounded DFS to implement the backtrack step from x is at most $|N^-(x)|$. From Lemma 6, each $\sqrt{n}$-bounded DFS takes time $O(n + m\sqrt{n})$. Hence all the backtrack steps performed in G can be implemented in $O(mn + m^2\sqrt{n})$ time and $O(\sqrt{n})$ space.

Discovery Step

The goal of the discovery step at a current vertex x is to check if a vertex $y \in N^+(x)$ has already been visited by the DFS. By Lemma 3, a DFS from s in the unique-path graph G cannot produce any forward or cross edges; hence this is equivalent to checking if the edge (x, y) is a back edge.

Procedure 3 gives the implementation of the discovery step. If y is one of the landmark vertices then clearly (x, y) is a back edge, i.e., Case 1 in Procedure 3 returns the correct output. Otherwise let $Z = Z(y)$ be the set of landmark vertices reachable from y by a $\sqrt{n}$-bounded DFS. Consider any back edge (x, y) such that y lies between the landmark vertices z_{j-1} and z_j ($j \geq 1$); then at least $z_j \in Z$. Hence if Z is empty, we know that (x, y) is not a back edge. Let $z_k \in Z$ be the landmark vertex with the highest index k in Z. Note that if (x, y) is a back edge, then $k \geq 1$. So the outputs of Case 2 and Case 3 are correct. But a $\sqrt{n}$-bounded DFS from y can discover more than one landmark vertex, since there can be successive back edges. The relation between z_j and z_k is described by the following lemma when (x, y) is a back edge.

LEMMA 8. *(a) If $j < p - 1$ or $j = p$, then z_j is the landmark vertex with the highest index j in Z. (b) If $j = p - 1$, then z_{p-1} or z_p is the landmark vertex with the highest index j in Z.*

PROOF. As distance of z_j from y is $\leq \sqrt{n}$, by Lemma 6, z_j will be discovered by a $\sqrt{n}$-bounded DFS from y, i.e., $z_j \in Z$. (a) If $j = p$, then z_p has the highest index p in Z since z_p is the last landmark vertex. If $j < p - 1$, the distance between y and z_ℓ is $> \sqrt{n}$ for any $\ell > j$.

Procedure 3 Procedure to implement the discovery step for the edge (x, y)

– Let $z_0 = s, z_1, \cdots, z_p = x$ be the current set of landmark vertices.
if $y \in \{z_0, \cdots, z_{p-1}\}$ **then** {**(Case 1)**: y is one of the landmark vertices}
  – Return '(x, y) is a back edge' ($y \neq z_p$, since there are no self loops).
end if
– Perform a $\sqrt{n}$-bounded DFS from y and let Z be the set of landmark vertices reached by this DFS.
if Z is empty **then** {**(Case 2)**: no landmark vertex is reached}
  – Return 'y has not been visited'.
else
  – Let $z_j \in Z$ be the landmark vertex with the highest index j.
  if $j = 0$ **then** {**(Case 3)**: $Z = \{z_0(= s)\}$}
    –Return 'y has not been visited'.
  else if $(j < p)$ **then** {**(Case 4)**}
    – Perform a second $\sqrt{n}$-bounded DFS from z_{j-1}, and terminate the DFS as soon as one of z_j or y is discovered.
    if y is discovered **then**
      – Return '(x, y) is a back edge'.
    else
      – Return 'y has not been visited'.
    end if
  else {**(Case 5)**: $z_p(= x)$ is the landmark vertex z_j with highest index j}
    – Perform a $2\sqrt{n}$-bounded DFS from z_{p-2} and terminate the DFS as soon as one of x or y is discovered. (if $p = j = 1$, perform a $\sqrt{n}$-bounded DFS from z_0).
    if y is discovered **then**
      – Return '(x, y) is a back edge'.
    else
      – Return 'y has not been visited'.
    end if
  end if
end if

Hence by Lemma 6 the $\sqrt{n}$-bounded DFS from y cannot discover z_ℓ. (b) If $j = p - 1$, then $z_{p-1} \in Z$, but depending on the distance of the current vertex $z_p = x$ from z_{p-1}, z_p may or may not belong to Z. ∎

The following lemma proves the correctness of Case 4; the correctness of Case 5 can be proved similarly.

LEMMA 9. *An edge (x, y) is a back edge iff we terminate with the discovery of vertex y by a $\sqrt{n}$-bounded DFS from z_{j-1} (i.e. y is discovered before z_j).*

PROOF. (only if) Suppose (x, y) is a back edge. Thus y is an ancestor of x. A $\sqrt{n}$-bounded DFS from y discovers exactly one landmark vertex that is a descendant of y since landmark vertices are spaced $\sqrt{n}$ apart. Since the highest-indexed landmark vertex discovered from

y is z_j, z_j is a descendant of y and hence y is on the unique path from z_{j-1} to z_j. Therefore y will be discovered before z_j by the $\sqrt{n}$-bounded DFS from z_{j-1}. (if) Suppose y is a newly discovered vertex. Then no landmark vertex is a descendant of y and the unique path from z_{j-1} to y is through z_j. So the $\sqrt{n}$-bounded DFS from z_{j-1} cannot discover y before z_{j-1}. ∎

Thus the discovery step involves at most two $\sqrt{n}$-bounded (or $2\sqrt{n}$-bounded) DFS. From Lemma 6, each discovery step takes time $O(n + m\sqrt{n})$. So all the discovery steps in the graph G can be performed in time $O(mn + m^2\sqrt{n})$ time and $O(\sqrt{n})$ space. This completes the proof of Theorem 4.

3.2 Improving to $O(n^\varepsilon)$ Space

Applying the ideas above recursively, we can improve the space bound to n^ε for any $\varepsilon \in (0, 1]$ while still achieving a polynomial time bound. We will prove the following theorem in this section.

THEOREM 10. *For any $\varepsilon \in (0, 1]$, STCON in unique-path graphs is solvable with $O(\frac{n^\varepsilon}{\varepsilon})$ space in $n^{O(\frac{1}{\varepsilon})}$ time.*

We first modify the definition of the landmark vertices. Now the landmark vertices will be spaced $n^{1-\varepsilon}$ distance apart on the current search path, so that they can be stored using $O(n^\varepsilon)$ space. Note that the immediate problem in increasing the spacing of the landmark vertices is that, in both the backtrack and discovery steps, landmark vertices may not be reachable by a n^ε-bounded DFS. So we need to apply the ideas of the previous section recursively.

We define the procedure D-REACH(u, U, H, d), where H is a subgraph of the unique-path graph G with source s[‖], $u \in V(H)$ and u is reachable from s, $U \subseteq V(H), 1 \leq d \leq |V(H)| - 1$. This procedure decides if there exists a vertex $v \in U$ within distance d from u in H. If such a $v \in U$ exists, then the procedure returns the first such vertex v and terminates; otherwise it outputs that no such vertex in U exists. A variant called D-REACH-ALL(u, U, H, d) determines *all* vertices in U reachable in distance d from u. This variant has the same time and space complexity as D-REACH. The STCON problem is same as D-REACH$(s, \{t\}, G, n - 1)$.

LEMMA 11. *If $|U| \leq n^\varepsilon$, the procedure* D-REACH(u, U, H, n^ε)*, can be implemented in $O(n + mn^\varepsilon)$ time using $O(n^\varepsilon)$ space.*

PROOF. The set U, $|U| \leq n^\varepsilon$, can be stored in $O(n^\varepsilon)$ space. Since u is reachable from s, D-REACH(u, U, H, n^ε) can be implemented like an L-bounded DFS by storing the entire active path; thus the backtrack step takes $O(1)$ time for each vertex. For each edge (x, y), the discovery step is performed by checking (i) if y belongs to the active path (of $\leq n^\varepsilon$ length) and (ii) if y is a new vertex, then whether it belongs to U, where $|U| \leq n^\varepsilon$ (and in that case the procedure returns with output y). Clearly the discovery step can be performed in $O(n^\varepsilon)$ time. Hence $O(n + mn^\varepsilon)$ time suffices to implement D-REACH(u, U, H, n^ε). ∎

[‖] G_1 is a *subgraph* of G_2 if $V(G_1) \subseteq V(G_2)$ and $E(G_1) \subseteq E(G_2)$.

Suppose we are at the current vertex x and $z_0 = s, z_1, \cdots, z_p = x$ is the set of landmark vertices stored at the top-most level of the recursion. In the backtrack step, similar to Procedure 2, for each $u \in N^-(x)$ we need to check if the landmark vertex z_{p-1} can reach the current vertex $z_p = x$ in the graph $G - (u, x)$. As the distance of x from z_{p-1} can be at most $n^{1-\varepsilon}$, for each $u \in N^-(x)$, we recursively call D-REACH$(z_{p-1}, \{x\}, G - (u, x), n^{1-\varepsilon})$. For the entire graph G, we have to make at most m such calls. Similarly for each discovery step, we have to call a $n^{1-\varepsilon}$-bounded D-REACH-ALL procedures first, and then we may have to call either a $n^{1-\varepsilon}$-bounded or a $2n^{1-\varepsilon}$-bounded D-REACH procedure (depending on the cases in Procedure 3). Using the same notations as in Section 3.1, these recursive procedures are: (i) D-REACH-ALL$(y, Z_{cur}, G, n^{1-\varepsilon})$ (where Z_{cur} is the current set of landmark vertices at the top-most level), (ii) if z_j is the highest indexed landmark vertex found, a second call is made either to D-REACH$(z_{j-1}, \{z_j, y\}, G, n^{1-\varepsilon})$ (in Case 4) or to D-REACH$(z_{p-2}, \{z_p, y\}, G, 2n^{1-\varepsilon})$ (in Case 5). Hence, there are at most $2m$ calls to $n^{1-\varepsilon}$-bounded D-REACH procedures (invoked by the backtrack and discovery steps) and at most m calls to $n^{1-\varepsilon}$-bounded or $2n^{1-\varepsilon}$-bounded D-REACH procedures (invoked by the discovery step).

Let $T(m, n, d)$ denote the running time of D-REACH(u, U, H, d) ($|U| \leq n^\varepsilon$), when H has at most n nodes and m edges. Note that, in each call to the D-REACH and D-REACH-ALL procedures used by our algorithm, $|U| \leq n^\varepsilon$ (since U is a subset of the landmark vertices). Hence we have the following recursion. $T(m, n, n) \leq 2mT(m, n, n^{1-\varepsilon}) + m \max(T(m, n, n^{1-\varepsilon}), T(m, n, 2n^{1-\varepsilon})) + O(m + n)$, i.e. $T(m, n, n) \leq 3mT(m, n, 2n^{1-\varepsilon}) + O(m + n)$. As the base case we have, $T(m, n, n^\varepsilon) = O(n + mn^\varepsilon) = (m + n)^{O(1)}$. The first two parameters in the recurrence relation are not changed at any step and they do not play active role in the solution of the recurrence. The solution to this recurrence is $(m + n)^{O(1+\frac{1}{\varepsilon})} = n^{O(\frac{1}{\varepsilon})}$, which is a polynomial when ε is a constant.

Next we analyze the increased space requirement due to these recursive calls. Note that, at any point of time we have to remember the landmark vertices at all levels of the recursion. But we can reuse the space allocated to landmark vertices in successive DFS calls at the same recursion level. The recursion depth is at most $\frac{1}{\varepsilon}$. Hence we have to remember at most $O(\frac{n^\varepsilon}{\varepsilon})$ vertices. So the overall space complexity of this recursive algorithm is $O(\frac{n^\varepsilon}{\varepsilon})$. This proves Theorem 10.

4 Recognition of Unique-Path Graphs

We prove the following theorem in this section. But due to space constraint, we omit the proof of the theorem.

THEOREM 12. *Given a directed graph G and a vertex $s \in V(G)$, there is an $O(\frac{n^\varepsilon}{\varepsilon})$-space, $n^{O(\frac{1}{\varepsilon})}$-time algorithm to decide whether G is a unique-path graph with respect to source s.*

5 Conclusions

An interesting open question is whether there are polynomial-time, polylog-space algorithms for STCON in unique-path graphs. It would also be interesting to see if our ideas can be extended to obtain an $O(n^\varepsilon)$-space polynomial-time algorithm for STCON in a more general family of graphs.

References

[1] R. ALELIUNAS, R. M. KARP, R. J. LIPTON, L. LOVASZ, AND C. RACKOFF. Random walks, universal traversal sequences, and the complexity of maze problems, *FOCS*, (1979) 218–223.

[2] E. ALLENDER. Reachability Problems: An Update, *CiE*, (2007) 25–27.

[3] E. ALLENDER, D. A. M. BARRINGTON, T. CHAKRABORTY, S. DATTA, AND S. ROY. Grid Graph Reachability Problems, *CCC*, (2006) 299–313.

[4] E. ALLENDER AND K.-J. LANGE. RUSPACE($\log n$) $\subseteq$ DSPACE ($\log^2 n / \log\log n$), *Theory Comput. Syst.*, 31(5), (1998) 539–550.

[5] C. ÁLVAREZ AND B. JENNER. A very hard log-space counting class, *Theor. Comput. Sci.*, 107(1), (1993) 3–30.

[6] G. BARNES, J. F. BUSS, W. L. RUZZO, AND B. SCHIEBER. A Sublinear Space, Polynomial Time Algorithm for Directed s-t Connectivity, *SIAM J. Comput.*, 27(5), (1998) 1273–1282.

[7] P. BERMAN AND J. SIMON. Lower Bounds on Graph Threading by Probabilistic Machines (Preliminary Version), *FOCS*, (1983) 304–311.

[8] A. BORODIN. Time Space Tradeoffs (Getting Closer to the Barrier?), *ISAAC*, (1993) 209–220.

[9] G. BUNTROCK, B. JENNER, K.-J. LANGE, AND P. ROSSMANITH. Unambiguity and Fewness for Logarithmic Space, *FCT*, (1991) 168–179.

[10] S. A. COOK AND C. RACKOFF. Space Lower Bounds for Maze Threadability on Restricted Machines, *SIAM J. Comput.*, 9(3), (1980) 636–652.

[11] T. H. CORMEN, C. E. LEISERSON, R. L. RIVEST, AND C. STEIN. Introduction to Algorithms, Second Edition, *The MIT Press*, (2001).

[12] J. EDMONDS AND C. K. POON. A nearly optimal time-space lower bound for directed *st*-connectivity on the NNJAG model, *STOC*, (1995) 147–156.

[13] C. M. PAPADIMITRIOU. *Computational complexity*, Addison-Wesley, (1994).

[14] C. K. POON. Space Bounds for Graph Connectivity Problems on Node-named JAGs and Node-ordered JAGs, *FOCS*, (1993) 218–227.

[15] O. REINGOLD. Undirected ST-connectivity in log-space, *STOC*, (2005) 376–385.

[16] O. REINGOLD, L. TREVISAN, AND S. P. VADHAN. Pseudorandom walks on regular digraphs and the RL vs. L problem, *STOC*, (2006) 457–466.

[17] K. REINHARDT AND E. ALLENDER. Making Nondeterminism Unambiguous, *FOCS*, (1997) 244–253.

[18] W. J. SAVITCH. Relationships between nondeterministic and deterministic tape Relationships Between Nondeterministic and Deterministic Tape Complexities, *J. Comput. Syst. Sci.*, 4(2), (1970) 177–192.

[19] A. WIGDERSON. The Complexity of Graph Connectivity, *MFCS*, (1992) 112–132.

Foundations of Software Technology and Theoretical Computer Science (Bangalore) 2008.
Editors: R. Hariharan, M. Mukund, V. Vinay; pp 268-279

Dynamic matrix rank with partial lookahead

Telikepalli Kavitha
Indian Institute of Science, Bangalore, India
kavitha@csa.iisc.ernet.in

ABSTRACT. We consider the problem of maintaining information about the rank of a matrix M under changes to its entries. For an $n \times n$ matrix M, we show an amortized upper bound of $O(n^{\omega-1})$ arithmetic operations per change for this problem, where $\omega < 2.376$ is the exponent for matrix multiplication, under the assumption that there is a *lookahead* of up to $\Theta(n)$ locations. That is, we know up to the next $\Theta(n)$ locations $(i_1, j_1), (i_2, j_2), \ldots$, whose entries are going to change, in advance; however we do not know the new entries in these locations in advance. We get the new entries in these locations in a dynamic manner.

1 Introduction

The dynamic matrix rank problem is that of computing the rank of an $n \times n$ matrix $M = \{m_{ij}\}$ under changes to the entries of M. The rank of a matrix M is the maximum number of linearly independent rows (or equivalently, columns) in M. The entries of M come from a field F, and the operation $change_{ij}(v)$ changes the value of the (i,j)-th entry of M to v, where $i, j \in \{1, \ldots, n\}$ and $v \in F$. We have a sequence of $change_{ij}(v)$ operations and the dynamic matrix rank problem is that of designing an efficient algorithm to return the rank of M under every *change* operation.

Here we consider a simpler variant of the above problem, where we assume that we can *lookahead* up to $\Theta(n)$ operations in advance so that we know location indices (i,j) of the entries of M that the next $\Theta(n)$ operations $change_{ij}$ are going to change. Note that we get to know the new value v of m_{ij} only when the operation $change_{ij}(v)$ actually happens, the assumption of lookahead is only regarding the location indices.

1.1 Earlier Work

The dynamic matrix rank problem was first studied by Frandsen and Frandsen [2] in 2006. They showed an upper bound of $O(n^{1.575})$ and a lower bound of $\Omega(n)$ for this problem (the lower bound is valid for algebraically closed fields). Frandsen and Frandsen present two algorithms for the dynamic matrix rank problem - the first algorithm is quite elementary and finds the rank by recomputing a reduced row echelon form of M for every change. This takes $O(n^2)$ time per change. This bound is valid also when a change alters arbitrarily many entries in a single column of the matrix. The second algorithm uses an implicit representation of the reduced row echelon form and this implicit representation is kept sufficiently compact by using fast rectangular matrix multiplication for global rebuilding. This yields a complexity of $O(n^{1.575})$ arithmetic operations per change, and this bound is valid when a change alters up to $O(n^{0.575})$ entries in a single column of M.

FSTTCS 2008
IARCS Annual Conference on
Foundations of Software Technology and Theoretical Computer Science
http://drops.dagstuhl.de/opus/volltexte/2008/1759

Sankowski [7] in 2004 gave several dynamic algorithms for computing matrix inverse, matrix determinant and solving systems of linear equations. The best of these algorithms obtains a worst case time of $O(n^{1.495})$ per change/query. These algorithms assume that the matrix M remains non-singular during the changes. In 2007 Sankowski [8] showed a randomized (there is a small probability of error here) reduction from the dynamic matrix rank problem to the dynamic matrix inverse problem: this yields a randomized upper bound of $O(n^{1.495})$ for the dynamic matrix rank problem.

Dynamic problems with lookahead. Dynamic graph problems with lookahead were considered by Khanna et al. in [5]. However their results have been superseded by dynamic algorithms without lookahead. Very recently, Sankowski and Mucha [9] worked on the dynamic transitive closure problem with lookahead. They present a randomized one-sided error algorithm with changes and queries in $O(n^{\omega(1,1,\epsilon)-\epsilon})$ time given a lookahead of n^ϵ operations, where $\omega(1,1,\epsilon)$ is the exponent of multiplication of an $n \times n$ matrix by an $n \times n^\epsilon$ matrix. For $\epsilon \leq 0.294$, this yields an algorithm with queries and changes in $O(n^{2-\epsilon})$ time, whereas for $\epsilon = 1$, the time is $O(n^{\omega-1})$, where $\omega < 2.376$ is the exponent for matrix multiplication. Their algorithm is based on a dynamic algorithm with lookahead for matrix inverse. This algorithm also assumes that the matrix M remains non-singular during the changes, which need not be true for the dynamic matrix rank problem. However using the randomized reduction of Sankowski [8] mentioned above, this algorithm for dynamic matrix inverse implies a Monte Carlo algorithm with the same bounds for the dynamic matrix rank problem with lookahead.

In this paper we use a direct approach for solving the dynamic matrix rank problem rather than routing through the dynamic matrix inverse problem. The dynamic matrix rank problem was also originally motivated by its application to the maximum rank matrix completion problem. The maximum rank matrix completion problem is that of assigning values to the undefined entries in a mixed matrix (this is a matrix where some entries are undefined) such that the rank of the resulting fully defined matrix is maximized. Geelen [3] gave a simple $O(n^9)$ time algorithm for the maximum rank completion problem that uses a data structure for dynamic matrix rank. However this application has been superseded by newer results: Berdan [1] reduced this complexity to $O(n^4)$ and Harvey et al. [4] gave an $O(n^3 \log n)$ algorithm for the maximum rank completion problem using a different technique.

Here we show a *deterministic* upper bound of $O(n^{\omega-1})$ for the dynamic matrix rank problem assuming that we are given a lookahead of $O(n)$ location indices. The trade-off between the number of locations that we can lookahead and the running time of our algorithm is: if we are allowed a lookahead of $s \leq n$ locations, then our amortized time per change is $O(n^\omega/s + ns^{\omega-2})$. Taking $s = \Theta(n)$ balances the two terms. Our algorithm relies on the idea of maintaining some "important entries" of certain matrices related to M. We describe this in more detail in the next section.

Organization of the paper. We discuss preliminaries in Section 2. Our algorithm for dynamic matrix rank is presented in Section 3. Our update subroutine is described and analyzed in Section 4. Due to lack of space, we omit some proofs from this version of the paper.

2 Preliminaries

We are given an $n \times n$ matrix M with entries from a field F. Our problem is that of computing the rank of M as elements of M change under the *change*() operations.

As a preprocessing step, in $O(n^\omega)$ time, where $\omega < 2.376$, we compute matrices U and E such that $UM = E$ where E has a special form, similar to reduced row-echelon form. Every row in E that is not the all-zeros row has a special element, whose value is non-zero; let us call this element the "leading element" of its row - such an element is the unique non-zero element in its column. We call such columns *clean* columns (a clean column is a vector with exactly one non-zero coordinate, which is the leading element of its row) and the remaining columns of E are the *dirty* columns. In E we have $rank(M)$ many clean columns, $n - rank(M)$ many dirty columns, $n - rank(M)$ many rows that are all-zeros and $rank(M)$ many non-zero rows. As an example, let us consider the matrix M_0 below to be our starting matrix. To the right we have $U_0 M_0 = E_0$ where U_0 is our transformation matrix and E_0 is in our special form. In the preprocessing step we compute U_0 and E_0.

$$M_0 = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 2 & 4 & 6 & 8 \\ 1 & 1 & 0 & 1 \\ 2 & 0 & 0 & 5 \end{bmatrix}; \quad \begin{bmatrix} 0 & 0 & 0 & 0.5 \\ -2 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0.5 \\ 2 & 0 & -4 & 1 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 & 4 \\ 2 & 4 & 6 & 8 \\ 1 & 1 & 0 & 1 \\ 2 & 0 & 0 & 5 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 2.5 \\ 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 1.5 \\ 0 & 0 & 6 & 9 \end{bmatrix}$$

In the matrix E_0 above, the first, second, and third columns are clean, while the fourth is a dirty column. Let us use the following terminology - right after the preprocessing step, we will define a function $\pi_0 : [n] \to \{0, 1, 2, \ldots, n\}$ that tells us which coordinate of a clean column of E_0 contains the unique non-zero element of that column. That is, only the $\pi_0(i)$-th coordinate of a clean column i is non-zero. So in the example above, we have $\pi_0(1) = 1$, $\pi_0(2) = 3$, and $\pi_0(3) = 4$. We define $\pi_0(i) = 0$ if column i is a dirty column. We will also use the function $\mathsf{Zero}_0 : [n] \to \{true, false\}$ where $\mathsf{Zero}_0(i) = true$ if and only if row i of E_0 is an all-zeros row. In our example, $\mathsf{Zero}_0(2) = true$ while $\mathsf{Zero}_0(w) = false$ for $w \in \{1, 3, 4\}$.

At the beginning of Step t we have the matrix M_{t-1} and let us see what we need to do in Step t, when the operation $change_{ij}(v)$ happens: here we are given the new value v of m_{ij} and say, the previous value of m_{ij} is u. So the new matrix $M_t = M_{t-1} + Z$, where Z is the all-zeros matrix except for its (i, j)-th coordinate that is $v - u$. Let E'_{t-1} denote the matrix $U_{t-1}(M_{t-1} + Z)$.

Let the symbols $r(X, s)$ and $c(X, \ell)$ denote the s-th row and ℓ-th column of matrix X, respectively. E'_{t-1} is the same as E_{t-1}, except for its j-th column which is $c(E_{t-1}, j) + (v - u)c(U_{t-1}, i)$. Even if we assume that we had the matrix E_{t-1} with us at the beginning of Step t, we now need to "clean up" E'_{t-1} by elementary row operations to obtain E_t in our special form and this could take as much as $\Theta(n^2)$ time. (Repeating these row operations on U_{t-1} yields U_t.) Thus we will not be able to maintain the matrices U_k, E_k at the end of Step k, for each k.

Thus at the beginning of Step t we in fact do not know the matrices U_{t-1} and E_{t-1}. However we will know certain important entries of E_{t-1} and U_{t-1}. For this purpose, we need to recall the functions π and Zero defined earlier; $\pi_{t-1}(i)$ is 0 if column i is a dirty column of E_{t-1}, else it gives the row coordinate of the unique non-zero element of column

i; $\mathsf{Zero}_{t-1}(i)$ is *true* if row i is an all-zeros row in E_{t-1}, else $\mathsf{Zero}_{t-1}(i)$ is $false$. What we will ensure at the beginning of Step t (to process $change_{ij}(\cdot)$) is the following:

- we will know the entire $\pi_{t-1}(j)$-th row of E_{t-1}: this is $r(E_{t-1}, \pi_{t-1}(j))$
- we will know the i-th column of U_{t-1} in its $\pi_{t-1}(j)$-th coordinate and all those coordinates w such that $Zero_{t-1}(w) = true$; we call this the *sub-column* $\tilde{c}(U_{t-1}, i)$.

We describe our algorithm in Section 3 and show that these entries suffice to determine the rank of E'_{t-1}. Now in Step t we also compute certain rows of E_t and certain entries of the matrix U_t. We postpone the work of computing the other rows of E_t and the other entries of U_t for now and this postponed work will be executed in batches at various points in the algorithm. The technique of postponing work for a while and executing this postponed work in batches at some other time in the algorithm has been used for other problems too (for instance, for computing maximum matchings in [6]). The novelty in our paper is that when we run our update step (to compute certain entries of E_t and U_t), we do not update entire columns since that is too expensive. Instead, we are able to identify these columns in their "critical" coordinates and update columns only in their critical coordinates. We are able to identify these critical coordinates in advance due to the function π on column indices of E and the function Zero on row indices of E that we maintain throughout our algorithm and also due to the lookahead on location indices that our problem assumes.

3 The algorithm for dynamic matrix rank

Let us assume that we have a lookahead of up to s locations; for convenience let s be a power of 2. After the preprocessing step, our algorithm can process $2s$ change operations as follows: let $change_{ij}(v)$ be the t-th change operation, where $1 \leq t \leq 2s$. When this change operation occurs, we do the following:

- $*$ first call $rank(i, j, v, r(E_{t-1}, \pi_{t-1}(j)), \tilde{c}(U_{t-1}, i))$; this returns the rank of the matrix M_t.
- $*$ if $t < 2s$ then call $update(\{t+1, \ldots, t+k\})$ where k is the largest power of 2 that divides t. This subroutine computes the rows $\pi_t(j_{t+1}), \ldots, \pi_t(j_{t+k})$ of E_t and the *sub-columns* $i_{t+1}, \ldots, i_{t+k}$ of U_t, where $(i_{t+1}, j_{t+1}), \ldots, (i_{t+k}, j_{t+k})$ are the location indices of the change operations in Steps $t+1, \ldots, t+k$.

We consider processing $2s$ change operations as described above as one *phase*. Each phase starts with the preprocessing step of computing the matrices U and E corresponding to the current M so that $UM = E$. Then we process $2s$ change operations. This finishes one phase. We will show in Section 4 that the $update(\{t+1, \ldots, t+k\})$ subroutine takes $O(nk^{\omega-1})$ time. Thus the total running time, $T(2s)$, for all the $update$ subroutines in a phase is given by: $T(2s) = O(ns^{\omega-1} + 2n(s/2)^{\omega-1} + 4n(s/4)^{\omega-1} + \ldots + sn(s/s)^{\omega-1})$, which is $O(ns^{\omega-1})$.

In Section 3.1 we describe the $rank()$ subroutine and show that its running time is $O(n)$. Hence the time for processing $2s$ change operations in a phase, after the initialization step, is $O(ns^{\omega-1})$. We incur a cost of $O(n^\omega)$ per phase for the initialization step and for every phase other than the first, let us distribute the cost of the initialization step of that phase among the $2s$ change operations of the previous phase. Thus the amortized cost of processing each change is $O(n^\omega/s + ns^{\omega-2})$. With $s = \Theta(n)$, our algorithm has a cost of $O(n^{\omega-1})$ per change, which proves the following theorem.

THEOREM 1. *Dynamic matrix rank over an arbitrary field can be solved using amortized $O(n^{\omega-1})$ arithmetic operations per change (where $\omega < 2.376$) with a preprocessing cost of $O(n^{\omega})$, provided that we are allowed a lookahead of up to $\Theta(n)$ locations.*

3.1 The subroutine $rank(i, j, v, r(E_{t-1}, \pi_{t-1}(j)), \tilde{c}(U_{t-1}, i))$

The *rank* subroutine is called after every change operation. Let $change_{ij}(v)$ be the current change operation - this changes the matrix M_{t-1} to M_t. The input to $rank()$ consists of i, j, v, the $\pi_{t-1}(j)$-th row of E_{t-1}, and the i-th column of U_{t-1} restricted to certain critical coordinates. Let u be the value of (i, j)-th coordinate of M_{t-1}. We can assume that the new value $v \neq u$ since $M_t = M_{t-1}$ if $v = u$.

Recall that we defined E'_{t-1} to be the matrix $U_{t-1}M_t$. Below we determine the rank of E'_{t-1} and it is easy to see that $rank(M_t) = rank(E'_{t-1})$ since U_{t-1} is just a transformation matrix that is a product of elementary row operations (adding a scalar multiple of one row to another row).

Let the rank of M_{t-1} be ρ. Then the rank of M_t is one of $\rho - 1, \rho, \rho + 1$. To decide which of these 3 numbers is the rank of M_t, we need to read only those entries of E'_{t-1} as given by checks (1), (2), and (3) below. It can be shown that these entries suffice to determine the rank of E'_{t-1}.

(1) We first check if there exists any row index w such that $\mathsf{Zero}_{t-1}(w) = true$ and the w-th coordinate of $c(U_{t-1}, i)$ is non-zero.

CLAIM 2. *If j is a dirty column in E_{t-1}, then $rank(E'_{t-1}) = \rho + 1$ if there is a w such that $\mathsf{Zero}_{t-1}(w) = true$ and the w-th coordinate of $c(U_{t-1}, i)$ is non-zero; else $rank(E'_{t-1}) = \rho$.*

Thus the case when j is a dirty column in E_{t-1} (i.e., $\pi_{t-1}(j) = 0$) is easy. Just knowing those coordinates w of column $c(U_{t-1}, i)$ such that $\mathsf{Zero}_{t-1}(w) = true$ suffices to determine the rank of E_t. The case when j is a clean column is only a little more difficult. If $\pi_{t-1}(j) \neq 0$, then we also do the checks as given by (2) and (3).

(2) We check if there exists any index d such that $\pi_{t-1}(d) = 0$ and the d-th coordinate of $r(E_{t-1}, \pi_{t-1}(j))$ is non-zero.

(3) If there is neither a d of check (2) nor a w of check (1), then we check if $E'_{t-1}[\pi_{t-1}(j), j]$ is non-zero or 0. This tells us if the $\pi_{t-1}(j)$-th row of E_t will be all-zeros or not.

CLAIM 3. *If j is a clean column in E_{t-1}, then we have the following cases:*

(i) If there is a w with $\mathsf{Zero}_{t-1}(w) = true$ and $U_{t-1}[w, i] \neq 0$ and if there is a d with $\pi_{t-1}(d) = 0$ and $E_{t-1}[\pi_{t-1}(j), d] \neq 0$, then the rank of E'_{t-1} is $\rho + 1$.

(ii) If there is no w with $\mathsf{Zero}_{t-1}(w) = true$ and $U_{t-1}[w, i] \neq 0$ and the row $\pi_{t-1}(j)$ in E'_{t-1} is all 0's, then the rank of E'_{t-1} is $\rho - 1$.

(iii) Else the rank of E'_{t-1} is ρ.

Thus at the end of this step we know the rank of M_t. In summary, note that we did not really need to know the entire column $c(U_{t-1}, i)$ here - its entries in coordinates w such that $\mathsf{Zero}_{t-1}(w) = true$ and in its $\pi_{t-1}(j)$-th coordinate are what we needed; also we needed to know the row $r(E_{t-1}, \pi_{t-1}(j))$ in the dirty column coordinates and in its $\pi_{t-1}(j)$-th coordinate in order to know if this row remains a non-zero row or if it becomes the all-zeros row in

E_t. Thus the two vectors: $r(E_{t-1}, \pi_{t-1}(j))$ and $\tilde{c}(U_{t-1}, i)$ were sufficient for us to determine the rank of M_t.

Now we compute the functions π_t and Zero_t from the functions π_{t-1} and Zero_{t-1}. The only values w for which $\pi_{t-1}(w)$ and $\pi_t(w)$ might be possibly different are $w = j$, $w = d$ (where d is a dirty column in E_{t-1} but will be a clean column in E_t). The only rows r for which $\mathsf{Zero}_t(r)$ and $\mathsf{Zero}_{t-1}(r)$ might be possibly different are row $\pi_{t-1}(j)$ and row w (where w is a zero row in E_{t-1} but will be a non-zero row in E_t). We omit the details of computing the functions π_t and Zero_t here.

It is easy to see that the time taken by the *rank* subroutine is $O(n)$. Hence the following lemma can be concluded.

LEMMA 4. *The subroutine $rank(i, j, v, r(E_{t-1}, \pi_{t-1}(j)), \tilde{c}(U_{t-1}, i))$ computes the rank of the matrix M_t in $O(n)$ time. It also maintains the functions π_t (on column indices) and Zero_t (on row indices).*

4 The $update()$ subroutine

In each update subroutine, we will compute certain rows of E_t and certain columns of U_t in "critical coordinates" that will be useful in the next few *change* operations. In particular, we will compute k rows of E_t and k sub-columns of U_t, where $k \geq 1$ is the largest power of 2 that is a divisor of t (recall that the current change operation is the t-th change operation in this phase).

Let the change operations that will occur in in the next k steps be in the locations $(x_1, y_1), \ldots, (x_k, y_k)$, respectively. Note that we know $(x_1, y_1), \ldots, (x_k, y_k)$ due to the lookahead allowed to us. Define the set $\mathcal{S}_t = \{\pi_t(y_1), \ldots, \pi_t(y_k), o_1, \ldots, o_h\}$ where $o_1, \ldots, o_h$ are all the row indices o such that $\mathsf{Zero}_t(o) = true$. The set $\mathcal{S}_t$ is the set of critical coordinates for $update(\{t+1, \ldots, t+k\})$.

In $update(\{t+1, \ldots, t+k\})$, we will compute the k rows $\pi_t(y_1), \ldots, \pi_t(y_k)$ of E_t (note that this implies that we know all those rows s of E_t, for $s \in \mathcal{S}_t$ since rows $o_1, \ldots, o_h$ are all-zeros in E_t) and the k columns $x_1, \ldots, x_k$ of U_t in the coordinates s for $s \in \mathcal{S}_t$. Once we compute the above rows and sub-columns, we will store them so that we can use/reuse them at later steps of this phase. In our current *update* subroutine, we will be reusing the rows and sub-columns that we computed in the *update* subroutine of Step $t-k$.

Let us see what rows and sub-columns were computed in the *update* subroutine of Step $t-k$. Let us use the symbol γ to denote $t-k$. Let $change_{i_\ell j_\ell}(v_\ell)$ be the change operation in Step ℓ, for $\gamma + 1 \leq \ell \leq t$. Since the number k is the largest power of 2 that is a divisor of t, it follows that $\gamma = t - k$ is a multiple of $2k$. Hence the set of critical coordinates for Step γ, call it $\mathcal{S}_\gamma$, contains $\{\pi_\gamma(j_{\gamma+1}), \ldots, \pi_\gamma(j_t), \pi_\gamma(y_1), \ldots, \pi_\gamma(y_k), z_1, \ldots, z_g\}$ where $z_1, \ldots, z_g$ are the row indices for which Zero_γ is true. We have the following claim stated as Proposition 5. Its proof is omitted here.

PROPOSITION 5. $\mathcal{S}_t \subseteq \mathcal{S}_\gamma$.

Since the *update* subroutine of Step t computes the rows s of E_t for $s \in \mathcal{S}_t$, it follows that the *update* subroutine of Step γ computed the rows s' of E_γ for $s' \in \mathcal{S}_\gamma$. Lemma 6 follows from this fact and Proposition 5.

LEMMA 6. *The update subroutine of Step γ computes the rows $\pi_t(y_1), \ldots, \pi_t(y_k)$ of E_γ and the columns $x_1, \ldots, x_k$ of U_γ restricted to the coordinates of $\mathcal{S}_t$.*

Now we are in a position to specify what tasks have to be performed in our current update subroutine, i.e., $update(\{t+1, \ldots, t+k\})$. Here we have to perform the following tasks:

(i) update the row $r(E_\gamma, \pi_t(y_h))$ to $r(E_t, \pi_t(y_h))$, for $1 \leq h \leq k$

(ii) update the sub-column $\tilde{c}(U_\gamma, x_h)$ to $\tilde{c}(U_t, x_h)$, for $1 \leq h \leq k$ (all these sub-columns are restricted to the coordinates $s \in \mathcal{S}_t$)

Theorem 7 is our main tool here to perform the updates given by **(i)** and **(ii)** above. During each Step ℓ, where $\gamma + 1 \leq \ell \leq t$, recall that the matrix $M_{\ell-1}$ gets changed to M_ℓ by a $change_{i_\ell j_\ell}(v_\ell)$ operation; we have $U_{\ell-1}M_\ell = E'_{\ell-1}$. In order to "clean" the matrix $E'_{\ell-1}$ we might need to clean up to 2 columns (column j_ℓ and a dirty column d_ℓ) of $E'_{\ell-1}$. The cleaning of column j_ℓ will be done by the row $\pi_\ell(j_\ell)$ and the cleaning of column d_ℓ will be done by the row $\pi_{\ell-1}(j_\ell)$. Let us use the symbols a_ℓ and b_ℓ for $\pi_{\ell-1}(j_\ell)$ and for $\pi_\ell(j_\ell)$, respectively. The row operations performed on $E'_{\ell-1}$ have to be then performed on $U_{\ell-1}$ and this yields U_ℓ.

Let us use the symbol $\tilde{E'}_{\ell-1}$ (similarly, $\tilde{U}_{\ell-1}$) to denote the matrix $E'_{\ell-1}$ (resp., $U_{\ell-1}$) *after* the "cleaning" of column j_ℓ and *before* the "cleaning" of column d_ℓ. Let e_{j_ℓ} denote the unit vector with a 1 in its j_ℓ-th coordinate.

Note that the equalities given in Theorem 7 hold for all row indices $s \in \{1, \ldots, n\}$, however we focus only on row indices $s \in \mathcal{S}_t$ here. (For simplicity of exposition, we will not qualify statements on a row $\pi_\ell(s)$ of E with "if $\pi_\ell(s) \neq 0$", thus we might refer to a row r' where $r' = 0$ - such a row will be the all 0's row.) The proof of Theorem 7 is omitted here.

THEOREM 7. *For each $s \in \mathcal{S}_t$, we have the following relation between row s of E_t and row s of E_γ:*

$$r(E_t, s) = r(E_\gamma, s) - \sum_{\ell=\gamma+1}^{t} \delta_{s,\ell} \cdot e_{j_\ell} - \sum_{\ell=\gamma+1}^{t} \alpha_{s,\ell} \cdot r(\tilde{E'}_{\ell-1}, a_\ell)$$

and the following relation between row s of U_t and row s of U_γ:

$$r(U_t, s) = r(U_\gamma, s) - \sum_{\ell=\gamma+1}^{t} \beta_{s,\ell} \cdot r(U_\ell, b_\ell) - \sum_{\ell=\gamma+1}^{t} \alpha_{s,\ell} \cdot r(\tilde{U}_{\ell-1}, a_\ell)$$

where the scalars $\delta_{s,\ell}, \alpha_{s,\ell}$ and $\beta_{s,\ell}$ are defined as follows:

* *If $b_\ell = 0$, then $\delta_{s,\ell} = (u_\ell - v_\ell) \cdot U_{\ell-1}[s, i_\ell]$.*

$$Else \quad \delta_{s,\ell} = \begin{cases} (u_\ell - v_\ell) \cdot U_{\ell-1}[b_\ell, i_\ell] & \text{if } s = b_\ell \\ E_{\ell-1}[s, j_\ell] & \text{otherwise} \end{cases}$$

* *If $b_\ell = 0$ then $\beta_{s,\ell} = 0$.*

$$Else \quad \beta_{s,\ell} = \begin{cases} 0 & \text{if } s = b_\ell \\ E'_{\ell-1}[s, j_\ell] / E'_{\ell-1}[b_\ell, j_\ell] & \text{otherwise} \end{cases}$$

∗ *If $a_\ell = 0$ or $a_\ell = b_\ell$ then $\alpha_{s,\ell} = 0$.*
Else let d_ℓ be the leading element of row a_ℓ in E_ℓ, i.e., $\pi_\ell(d_\ell) = a_\ell$. We have:

$$\alpha_{s,\ell} = \begin{cases} 0 & \text{if } s = a_\ell \\ E'_{\ell-1}[s, d_\ell] / E'_{\ell-1}[a_\ell, d_\ell] & \text{otherwise} \end{cases}$$

The above theorem can be written in matrix form as follows. For simplicity let us call the elements of $\mathcal{S}_t$ as $s_1, \ldots, s_p$, and the $\delta_{s,\ell}$, $\beta_{s,\ell}$ and $\alpha_{s,\ell}$ values for $s \in \mathcal{S}_t$ and $1 \leq \ell \leq k$ as $\delta_{1,1}, \ldots, \delta_{p,k}$, $\beta_{1,1}, \ldots, \beta_{p,k}$ and $\alpha_{1,1}, \ldots, \alpha_{p,k}$, respectively.

$$\begin{bmatrix} r(E_t, s_1) \\ \vdots \\ r(E_t, s_p) \end{bmatrix} = \begin{bmatrix} r(E_\gamma, s_1) \\ \vdots \\ r(E_\gamma, s_p) \end{bmatrix} - \begin{bmatrix} \delta_{1,1} & \alpha_{1,1} & \ldots & \delta_{1,k} & \alpha_{1,k} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \delta_{p,1} & \alpha_{p,1} & \ldots & \delta_{p,1} & \alpha_{p,k} \end{bmatrix} \begin{bmatrix} e_{j_{\gamma+1}} \\ \vdots \\ r(\tilde{E'}_{t-1}, a_t) \end{bmatrix} \quad (1)$$

$$\begin{bmatrix} r(U_t, s_1) \\ \vdots \\ r(U_t, s_p) \end{bmatrix} = \begin{bmatrix} r(U_\gamma, s_1) \\ \vdots \\ r(U_\gamma, s_p) \end{bmatrix} - \begin{bmatrix} \beta_{1,1} & \alpha_{1,1} & \ldots & \beta_{1,k} & \alpha_{1,k} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \beta_{p,1} & \alpha_{p,1} & \ldots & \beta_{p,k} & \alpha_{p,k} \end{bmatrix} \begin{bmatrix} r(U_\gamma, b_{\gamma+1}) \\ \vdots \\ r(\tilde{U}_{t-1}, a_t) \end{bmatrix} \quad (2)$$

Our goal is to determine the matrices on the left of Eqns. (1) and (2). However notice that these matrices can be quite large. Each matrix on the left is a $p \times n$ matrix, where $p = |\mathcal{S}_t|$. The value of $|\mathcal{S}_t|$ could be $\Theta(n)$ and then we would spend $\Theta(n^2)$ time only to just write down all the entries of such a matrix. We certainly do not want to spend $\Theta(n^2)$ time for $update(\{t+1, \ldots, t+k\})$. Recall that we promised to show that $update(\{t+1, \ldots, t+k\})$ takes $O(nk^{\omega-1})$ time. Hence we do not perform all the matrix arithmetic as specified by Eqns. (1) and (2).

Instead, to compute the relevant rows of E_t, we restrict the matrices of Eqn. (1) solely to the row indices $\pi_t(y_1), \ldots, \pi_t(y_k)$ since job (i) only needs these rows of E_t. This involves multiplying a $k \times 2k$ matrix (of α's and δ's) with a $2k \times n$ matrix which takes $O(nk^{\omega-1})$ time, once we know the matrices involved. To compute the sub-columns of U_t, we restrict the matrices of Eqn. (2) only to the column indices $x_1, \ldots, x_k$, since job (ii) only needs columns $x_1, \ldots, x_k$ of U_t, restricted to coordinates in $\mathcal{S}_t$. This involves multiplying a $p \times 2k$ matrix (of α's and β's) with a $2k \times k$ matrix of sub-rows, which again takes $O(nk^{\omega-1})$ time. Observe that we need the entire $p \times 2k$ matrix of β's and α's written above for this matrix multiplication. Determining the β's, α's, and δ's is, in fact, the crux of the *update* subroutine. We will show the following lemma here. Section 4.1 contains the algorithm that proves this lemma.

LEMMA 8. *The $\delta_{s,\ell}, \alpha_{s,\ell}, \beta_{s,\ell}$ values for all $s \in \mathcal{S}_t$ and $\gamma + 1 \leq \ell \leq t$ can be computed in time $O(nk^{\omega-1})$.*

Once we compute the matrix of α's and δ's, task (i) of $update(\{t+1, \ldots, t+k\})$ is essentially done since we now know all the matrices on the right of Eqn. (1): the matrix whose rows are $r(E_\gamma, s)$ for $s \in \{\pi_t(y_1), \ldots, \pi_t(y_k)\}$ is known to us (by Lemma 6) and each *odd* indexed row in the rightmost matrix is a unit vector (e_{j_ℓ} is the $(2(\ell - \gamma) - 1)$-th row). Regarding the *even* indexed rows, the vector $r(E'_{\ell-1}, a_\ell)$ was a part of the input to the *rank* subroutine of Step ℓ (recall that $a_\ell = \pi_{\ell-1}(j_\ell)$) and we have $r(\tilde{E'}_{\ell-1}, a_\ell) = r(E'_{\ell-1}, a_\ell) - \beta_{a_\ell,\ell} \cdot e_{j_\ell}$.

Completing task (ii) of $update(\{t+1,\ldots,t+k\})$ is more difficult, the rightmost matrix of Eqn. (2) is unknown to us. We describe how we compute this matrix after we present the proof of Lemma 8.

4.1 Proof of Lemma 8

We compute the $\delta_{s,\ell}, \alpha_{s,\ell}, \beta_{s,\ell}$ values for all $s \in \mathcal{S}_t$ and $\gamma+1 \le \ell \le t$ using a recursive subroutine UpdateColumns$(\{\gamma+1,\ldots,t\})$. This subroutine computes these scalars by determining the sub-columns $\tilde{c}(U_{\ell-1}, i_\ell)$, $\tilde{c}(E_{\ell-1}, j_\ell)$, $\tilde{c}(E_{\ell-1}, d_\ell)$ in the coordinates of $\mathcal{S}_t$ for $\gamma+1 \le \ell \le t$; recall that these sub-columns determine the $\delta_{s,\ell}, \alpha_{s,\ell}, \beta_{s,\ell}$ values by Theorem 7.

Here we describe a generic subroutine UpdateColumns$(\{w+1,\ldots,w+q\})$ (this is either the original subroutine UpdateColumns$(\{\gamma+1,\ldots,t\})$ or a subroutine invoked in a recursive call). This subroutine will compute $\delta_{s,\ell}, \alpha_{s,\ell}, \beta_{s,\ell}$ values for all $s \in \mathcal{S}_t$ and $w+1 \le \ell \le w+q$. We will maintain the invariant that we know (restricted to the coordinates of $\mathcal{S}_t$) the sub-columns j_ℓ, d_ℓ of E_w and the sub-columns i_ℓ of U_w, for $w+1 \le \ell \le w+q$, at the time UpdateColumns$(\{w+1,\ldots,w+q\})$ is called. Observe that this invariant is true at the onset when UpdateColumns$(\{\gamma+1,\ldots,t\})$ is called, since we know the vectors $\tilde{c}(E_\gamma, j_\ell)$, $\tilde{c}(E_\gamma, d_\ell)$, $\tilde{c}(U_\gamma, i_\ell)$ in the coordinates of $\mathcal{S}_t$ for $\gamma+1 \le \ell \le t$ (by the *update* subroutine of Step γ and because $\mathcal{S}_\gamma \supseteq \mathcal{S}_t$).

UpdateColumns$(\{w+1,\ldots,w+q\})$**:**

- If $q = 1$ then it is the base case: we compute the values $\delta_{s,w+1}$, $\beta_{s,w+1}$ and $\alpha_{s,w+1}$ for all $s \in \mathcal{S}_t$ from $\tilde{c}(E_w, j_{w+1})$, $\tilde{c}(E_w, d_{w+1})$, $\tilde{c}(U_w, i_{w+1})$, and scalars v_{w+1}, u_{w+1} using Theorem 7.
- Else
  (1) Call UpdateColumns$(\{w+1,\ldots,w+q/2\})$ recursively.
  (2) Perform a *bulk update step* to update
    (I) the sub-columns j_ℓ, d_ℓ of E_w to $E_{w+q/2}$ for $w+q/2+1 \le \ell \le w+q$
    (II) the sub-columns i_ℓ of U_w to $U_{w+q/2}$ for $w+q/2+1 \le \ell \le w+q$
  (3) Call UpdateColumns$(\{w+q/2+1,\ldots,w+q\})$ recursively.

Remark. Observe that Step (2) enables us to maintain the following invariant that is necessary when UpdateColumns$(\{w+q/2+1,\ldots,w+q\})$ is called in Step (3): we know the columns $\tilde{c}(E_{w+q/2}, j_\ell)$, $\tilde{c}(E_{w+q/2}, d_\ell)$, $\tilde{c}(U_{w+q/2}, i_\ell)$ in the coordinates of $\mathcal{S}_t$ for $w+q/2+1 \le \ell \le w+q$.

The base case is easy since we had maintained the invariant that we know the sub-columns $\tilde{c}(E_w, j_{w+1})$, $\tilde{c}(E_w, d_{w+1})$, $\tilde{c}(U_w, i_{w+1})$ in the coordinates s, where $s \in \mathcal{S}_t$, when the subroutine UpdateColumns$(\{w+1\})$ is called. The base case takes $O(n)$ time. Thus we have the following recurrence for the running time $T'(q)$ of UpdateColumns$(\{w+1,\ldots,w+q\})$:

$$T'(q) = \begin{cases} 2T'(q/2) + \text{time for the bulk update step} & \text{if } q > 1 \\ O(n) & \text{if } q = 1 \end{cases} \tag{3}$$

Now we need to describe the bulk update step. The bulk update step has to perform the updates given by (I) and (II) of Step (2) in the algorithm UpdateColumns$(\{w+1,\ldots,w+q\})$ described above. We describe first how we do (I) and then (II).

(I): Update columns j_ℓ, d_ℓ of E_w to $E_{w+q/2}$. Here we need to update the columns j_ℓ, d_ℓ of E_w to $E_{w+q/2}$, for $w+q/2+1 \leq \ell \leq w+q$. This amounts to updating the coordinates $j_{w+q/2+1}, \ldots, j_{w+q}, d_{w+q/2+1}, \ldots, d_{w+q}$ of certain *rows* of E_w to $E_{w+q/2}$. These row indices are the numbers s in $\mathcal{S}_t$. The updates on these rows of E_w are given by equations analogous to the ones in Theorem 7. We have for each $s \in \mathcal{S}_t$ the following equation:

$$r(E_{w+q/2}, s) = r(E_w, s) - \sum_{\ell=w+1}^{w+q/2} \delta_{s,\ell} \cdot e_{j_\ell} - \sum_{\ell=w+1}^{w+q/2} \alpha_{s,\ell} \cdot r(\tilde{E'}_{\ell-1}, a_\ell)$$

where the $\delta_{s,\ell}$'s and $\alpha_{s,\ell}$'s are defined in Theorem 7. Note that we already know all $\delta_{s,\ell}, \alpha_{s,\ell}, \beta_{s,\ell}$ values for $w+1 \leq \ell \leq w+q/2$, since we computed these values in the UpdateColumns$(\{w+1, \ldots, w+q/2\})$ subroutine, that was called in Step 1 of UpdateColumns$(\{w+1, \ldots, w+q\})$. The above equations in matrix form become:

$$\begin{bmatrix} r(E_{w+q/2}, s_1) \\ \vdots \\ r(E_{w+q/2}, s_p) \end{bmatrix} = \begin{bmatrix} r(E_w, s_1) \\ \vdots \\ r(E_w, s_p) \end{bmatrix} - \begin{bmatrix} \delta_{1,1} & \ldots & \alpha_{1,q/2} \\ \vdots & \vdots & \vdots \\ \delta_{p,1} & \ldots & \alpha_{p,q/2} \end{bmatrix} \begin{bmatrix} e_{j_{w+1}} \\ \vdots \\ r(\tilde{E'}_{w+q/2-1}, a_{w+q/2}) \end{bmatrix} \quad (4)$$

where the rows of the E matrices are restricted to the coordinates $j_{w+q/2+1}, \ldots, j_{w+q}, d_{w+q/2+1}, \ldots, d_{w+q}$. Recall that $s_1, \ldots, s_p$ are the elements of $\mathcal{S}_t$ and for convenience, we called the $\delta_{s,\ell}$ and $\alpha_{s,\ell}$ values that we computed in UpdateColumns$(\{w+1, \ldots, w+q/2\})$ as $\delta_{1,1}, \ldots, \delta_{p,q/2}$ and $\alpha_{1,1}, \ldots, \alpha_{p,q/2}$.

We know all the matrices on the right in Eqn. (4) (regarding the rows of the rightmost matrix, refer to the paragraph after the statement of Lemma 8). Since we multiply a $p \times q$ matrix with a $q \times q$ matrix in Eqn. (4), the time taken for this matrix multiplication is $O((p/q)q^\omega)$, which is $O(nq^{\omega-1})$. We thus determine for each $s \in \mathcal{S}_t$, the row s of $E_{w+q/2}$ restricted to coordinates j_ℓ, d_ℓ, for $w+q/2+1 \leq \ell \leq w+q$, in $O(nq^{\omega-1})$ time; in other words, we know the columns j_ℓ, d_ℓ, for $w+q/2+1 \leq \ell \leq w+q$, of $E_{w+q/2}$, in the coordinates of elements in $\mathcal{S}_t$ in $O(nq^{\omega-1})$ time.

(II): Update columns i_ℓ of U_w to $U_{w+q/2}$. Here we need to update columns i_ℓ, for $w+q/2+1 \leq \ell \leq w+q$ of U_w to $U_{w+q/2}$. We follow the same method that we used to update the columns of E_w to $E_{w+q/2}$. For each $s \in \mathcal{S}_t$ we have the following equation:

$$r(U_{w+q/2+1}, s) = r(U_w, s) - \sum_{\ell=w+1}^{w+q/2} \beta_{s,\ell} \cdot r(U_{\ell-1}, b_\ell) - \sum_{\ell=w+1}^{w+q/2} \alpha_{s,\ell} \cdot r(\tilde{U}_{\ell-1}, a_\ell)$$

where the $\beta_{s,\ell}$'s and $\alpha_{s,\ell}$'s are defined in Theorem 7. The bulk update step for the rows of U (written in matrix form) is analogous to Eqn. (4) - note that here these rows are restricted to the coordinates i_ℓ for $w+q/2+1 \leq \ell \leq w+q$. However, here we cannot claim that we know all the matrices on the right of the analogous equation of Eqn. (4) for U. The rightmost matrix, whose rows are the sub-rows $r(U_w, b_{w+1}), r(\tilde{U}_w, a_{w+1}), \ldots, r(\tilde{U}_{w+q/2-1}, a_{w+q/2})$ is unknown to us and we have to determine it now. We will compute the rows of this matrix using the same recursive strategy as was used in the UpdateColumns algorithm. The subroutine UpdateRowsU$(\{w+1, \ldots, w+q/2\})$, described below, computes these rows.

We present a generic subroutine UpdateRowsU($\{z+1,\ldots,z+l\}$) (this is either the original subroutine UpdateRowsU($\{w+1,\ldots,w+q/2\}$) or one invoked in a recursive call). Note that $w+1 \leq z+1 \leq z+l \leq w+q/2$. We maintain the invariant that at the time of calling UpdateRowsU($\{z+1,\ldots,z+l\}$), we have the rows $b_{z+1},\ldots,a_{z+l}$ of U_z in the coordinates of $i_{w+q/2+1},\ldots,i_{w+q}$; in this subroutine we update these sub-rows to the sub-rows $r(U_z,b_{z+1}),\ldots,r(\tilde{U}_{z+l-1},a_{z+l}))$, respectively. Observe that this invariant is true at the onset when we call UpdateRowsU($\{w+1,\ldots,w+q/2\}$) by the invariant that we had maintained when UpdateRowsU($\{w+1,\ldots,w+q\}$) was called.

UpdateRowsU($\{z+1,\ldots,z+l\}$):

- If $l=1$ then it is the base case: by our invariant, we have the rows $r(U_z,b_{z+1})$ and $r(U_z,a_{z+1})$ in the coordinates $i_{w+q/2+1},\ldots,i_{w+q}$. We need to update $r(U_z,a_{z+1})$ to $r(\tilde{U}_z,a_{z+1})$. This is easily done.
- Else
  - Call UpdateRowsU($\{z+1,\ldots,z+l/2\}$).
  - Simultaneously update the sub-rows $b_{z+l/2+1},\ldots,a_{z+l}$ (call these row indices $s'_1,\ldots,s'_l$ for convenience) of U_z to $U_{z+l/2}$ as follows:

$$\begin{bmatrix} r(U_{z+l/2},s'_1) \\ \vdots \\ r(U_{z+l/2},s'_l) \end{bmatrix} = \begin{bmatrix} r(U_z,s'_1) \\ \vdots \\ r(U_z,s'_l) \end{bmatrix} - \begin{bmatrix} \beta_{1,1} & \ldots & \alpha_{1,l/2} \\ \vdots & \vdots & \vdots \\ \beta_{l,1} & \ldots & \alpha_{l,l/2} \end{bmatrix} \begin{bmatrix} r(U_z,(b_{z+1}) \\ \vdots \\ r(\tilde{U}_{z+l/2-1},a_{z+l/2}) \end{bmatrix}$$

    We know all the matrices on the right side above, since we have already computed the matrix of α's and β's corresponding to $\{z+1,\ldots,z+l\}$ during the subroutine UpdateColumns($\{w+1,\ldots,w+q/2\}$) and the rightmost matrix was computed in the earlier recursive call UpdateRowsU($\{z+1,\ldots,z+l/2\}$).
  - Call UpdateRowsU($\{z+l/2+1,\ldots,z+l\}$). [Note that the update of the above step ensures that our invariant is maintained for this call of UpdateRowsU.]

It is easy to see that the recurrence relation for the running time $T''(l)$ of the above algorithm UpdateRowsU($\{z+1,\ldots,z+l\}$) is:

$$T''(l) = \begin{cases} 2T''(l/2)+O(ql^{\omega-1}) & \text{if } l>1 \\ O(q) & \text{if } l=1 \end{cases}$$

$T''(l)$ solves to $O(ql^{\omega-1})$. Thus $T''(q)$ is $O(q^\omega)$. This is the time taken to compute the rightmost matrix in the equation analogous to Eqn. (4) for U. Once this matrix is determined, the matrix multiplication is performed in $O(nq^{\omega-1})$ time. Thus we determine the sub-rows s of $U_{w+q/2}$ for $s \in \mathcal{S}_t$ in the coordinates $i_{w+q/2+1},\ldots,i_{w+q}$. In other words, we computed the columns $i_{w+q/2+1},\ldots,i_{w+q}$ of $U_{w+q/2}$ in the coordinates of $\mathcal{S}_t$. This completes the description of the bulk update step of the subroutine UpdateColumns($\{w+1,\ldots,w+q\}$).

We can now analyse the running time $T'(q)$ of UpdateColumns($\{w+1,\ldots,w+q\}$) (see Eqn. (3)). We have $T'(q)=2T'(q/2)+O(nq^{\omega-1})$ and $T'(1)=O(n)$. Thus $T'(q)$ solves to $O(nq^{\omega-1})$. Hence $T'(k)$, the running time of UpdateColumns($\{\gamma+1,\ldots,t\}$), is $O(nk^{\omega-1})$. This proves Lemma 8. ∎

Completing task (ii) of $update(\{t+1,\ldots,t+k\})$**.** We have to determine all the matrices on the right hand side of Eqn. (2) - we currently know two of these matrices: the matrix whose rows are $r(U_\gamma, s)$ restricted to columns $i_{\gamma+1},\ldots,i_t$ (by Lemma 6) and the matrix of α's and β's that we just computed. The rightmost matrix of Eqn. (2) is currently unknown to us; however we know the matrix whose rows are $b_1, a_1, \ldots, b_k, a_k$ of U_γ. We need to update this matrix to the desired matrix. This is done by calling the subroutine UpdateRowsU$(\{\gamma+1,\ldots,t\})$ described in the previous section. This takes $O(nk^{\omega-1})$ time by our analysis given there. Now we know all the matrices on the right of Eqn. (2). Thus we can compute the matrix on the left of Eqn. (2) in time $O(nk^{\omega-1})$. This completes the description of the update subroutine. We have thus shown the following theorem.

THEOREM 9. *The* $update(\{t+1,\ldots,t+k\})$ *subroutine obtains the rows* $r(E_t, \pi_t(j_h))$ *and the sub-columns* $\tilde{c}(U_t, i_h)$ *in the coordinates* s *for* $s \in \mathcal{S}_t$*, for* $\gamma+1 \le h \le t$*, in time* $O(nk^{\omega-1})$.

Conclusions

We showed that the dynamic matrix rank problem for an $n \times n$ matrix with entries from any field can be solved using amortized $O(n^{\omega-1})$ arithmetic operations per change (where $\omega < 2.376$) with a preprocessing cost of $O(n^\omega)$, provided that we are allowed a lookahead of up to $\Theta(n)$ locations. An open problem is to show such a bound without lookahead.

References

[1] M. Berdan. A matrix rank problem. Master's thesis, University of Waterloo, December 2003.

[2] G. S. Frandsen and P. F. Frandsen. Dynamic Matrix Rank. In *Proc. of the 33rd ICALP*, LNCS 4051: 395-406, 2006.

[3] J. F. Geelen. Maximum rank matrix completion. *Linear Algebra Appl.*, 288(1-3): 211-217, 1999.

[4] N. J. A. Harvey, D. R. Karger, and K. Murota. Deterministic network coding by matrix completion. In *Proc. of the 16th Annual ACM-SODA*, 489-498, 2005.

[5] S. Khanna, R. Motwani, and R. H. Wilson. On certificates and lookahead on dynamic graph problems. In *Proc. of the 7th Annual ACM-SIAM SODA*: 222-231, 1996.

[6] M. Mucha and P. Sankowski. Maximum Matchings via Gaussian Elimination. In *Proc. of the 45th Annual IEEE FOCS*: 248-255, 2004.

[7] P. Sankowski. Dynamic Transitive Closure via Dynamic Matrix Inverse. In *Proc. of the 45th Annual IEEE FOCS*: 509-517, 2004.

[8] P. Sankowski. Faster Dynamic Matchings and Vertex Connectivity. In *Proc. of the 18th Annual ACM-SIAM SODA*: 118-126, 2007.

[9] P. Sankowski and M. Mucha. Fast Dynamic Transitive Closure with Lookahead. To appear in *Algorithmica*. Online version available at `http://www.springerlink.com/content/1q86442762411268/`

Foundations of Software Technology and Theoretical Computer Science (Bangalore) 2008.
Editors: R. Hariharan, M. Mukund, V. Vinay; pp 280-291

A Cubic-Vertex Kernel for Flip Consensus Tree

Christian Komusiewicz* **and Johannes Uhlmann†**
Institut für Informatik,
Friedrich-Schiller-Universität Jena,
Ernst-Abbe-Platz 2,
D-07743 Jena, Germany.
{ckomus,uhlmann}@minet.uni-jena.de

ABSTRACT. Given a bipartite graph $G = (V_c, V_t, E)$ and a non-negative integer k, the NP-complete MINIMUM-FLIP CONSENSUS TREE problem asks whether G can be transformed, using up to k edge insertions and deletions, into a graph that does not contain an induced P_5 with its first vertex in V_t (a so-called M-graph or Σ-graph). This problem plays an important role in computational phylogenetics, V_c standing for the characters and V_t standing for taxa. Chen et al. [IEEE/ACM TCBB 2006] showed that MINIMUM-FLIP CONSENSUS TREE is NP-complete and presented a parameterized algorithm with running time $O(6^k \cdot |V_t| \cdot |V_c|)$. Recently, Böcker et al. [IWPEC '08] presented a refined search tree algorithm with running time $O(4.83^k(|V_t| + |V_c|) + |V_t| \cdot |V_c|)$. We complement these results by polynomial-time executable data reduction rules yielding a problem kernel with $O(k^3)$ vertices.

1 Introduction

The MINIMUM-FLIP CONSENSUS TREE problem arises in computational phylogenetics in the context of supertree construction. Given a binary matrix, the task is to "flip" a minimum number of entries of the matrix in order to obtain a binary matrix that admits what is called a *perfect phylogeny*. These are matrices from which a rooted phylogenetic tree can be inferred [15, 21].

In this work, we employ a graph-theoretic formulation of the problem, which was introduced by Chen et al. [4]: the binary input matrix A is represented by a bipartite graph $G = (V_c, V_t, E)$ where an edge between two vertices $i \in V_c$ and $j \in V_t$ is drawn iff $A_{i,j} = 1$. The matrix then admits a perfect phylogeny iff the graph does not contain an *M-graph* as an induced subgraph. An M-graph is a path of five vertices with the first vertex belonging to V_t. An example of such an M-graph is depicted in Fig. 1. Then, the flipping of a matrix entry $A_{i,j}$ from 0 to 1 corresponds to the insertion of the edge $\{i, j\}$, and from 1 to 0 corresponds to the deletion of the edge $\{i, j\}$. The MINIMUM-FLIP CONSENSUS TREE problem is then defined as follows.

Instance: A bipartite graph $G = (V_c, V_t, E)$ and an integer $k \geq 0$.
Question: Can G be changed by up to k edge modifications into an M-free graph, that is, a graph without an induced M-graph?

*Supported by a PhD fellowship of the Carl-Zeiss-Stiftung.
†Supported by the DFG, research project PABI, NI 369/7.

FSTTCS 2008
IARCS Annual Conference on
Foundations of Software Technology and Theoretical Computer Science
http://drops.dagstuhl.de/opus/volltexte/2008/1760

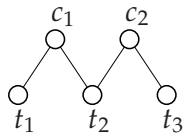


Figure 1: An M-subgraph with $t_1, t_2, t_3 \in V_t$ and $c_1, c_2 \in V_c$.

Chen et al. [4] showed that MINIMUM-FLIP CONSENSUS TREE is NP-complete, which motivates the study of the MINIMUM-FLIP CONSENSUS TREE problem in the context of parameterized algorithmics [19]. Other than previous work [1, 4] on parameterized algorithms for MINIMUM-FLIP CONSENSUS TREE, which mainly dealt with the development of depth-bounded search trees, here we deal with polynomial-time data reduction with provable performance guarantee, that is, kernelization. Kernelization is considered as one of the theoretically and practically most interesting algorithmic methods of parameterized algorithmics [6, 14, 17, 19]. Roughly speaking, the goal is to derive a problem kernel which is an instance "equivalent" to the original one but with (hopefully) much smaller size; in particular, the size of the problem kernel shall only be a function of the parameter k. Moreover, the problem kernel needs to be computable in polynomial-time—so this is closely related to polynomial-time preprocessing.

Known results and previous work. The MINIMUM-FLIP CONSENSUS TREE was introduced by Chen et al. [4] who also proved its NP-completeness and described a factor-$2d$ approximation algorithm for graphs with maximum degree d. Furthermore, they showed fixed-parameter tractability with respect to the number of flips k by describing a simple $O(6^k \cdot mn)$ search tree algorithm that is based on the forbidden induced subgraph characterization with M-graphs. Recently, Böcker et al. [1] improved the running time to $O(4.83^k(|V_c| + |V_t|) + |V_c| \cdot |V_t|)$ by employing a refined branching strategy that leads to a search tree of size $O(4.83^k)$. This theoretically proven running time acceleration was also practically confirmed by computational experiments [1].

From a graph-theoretic point of view, MINIMUM-FLIP CONSENSUS TREE belongs to the class of so-called Π-EDGE MODIFICATION problems: Given a graph G, a graph property Π, and an integer $k \geq 0$, the question is whether G can be transformed by at most k edge modifications into a graph with property Π. A lot of work has been put into classifying Π-EDGE MODIFICATION problems with respect to their classical complexity [3, 18, 24]. Recently, parameterized algorithmics —in particular kernelizations—for Π-EDGE MODIFICATION problems have attracted special attention. For instance, there is a series of papers studying the kernelizability of CLUSTER EDITING and some of its variations [7, 9, 11, 13, 22]. Also vertex deletion problems such as UNDIRECTED FEEDBACK VERTEX SET with its cubic-size problem kernel [2]—very recently improved to a quadratic-vertex problem kernel [23]—have been studied, underpinning the importance of kernelization in the wide area of graph modification problems. Furthermore, even exponential-size kernels such as those for CLIQUE COVER [10] and BICLIQUE COVER [8] are of importance, since they often provide the only known way to show that a problem is fixed-parameter tractable. Damaschke [5] investigated kernelization in the context of enumerating all inclusion-minimal solutions of size

at most k. In this scenario, when designing reduction rules one has to guarantee that all inclusion-minimal solutions of size at most k are preserved. Kernels that fulfill these additional constraints are called *full kernels*. In this setting, Damaschke [5] presents a full kernel consisting of $O(6^k)$ matrix entries for the following problem closely related to MINIMUM-FLIP CONSENSUS TREE: Given a binary matrix and a non-negative integer k, enumerate all inclusion-minimal sets of at most k flips that transform the matrix into a matrix that admits an unrooted perfect phylogeny.

Our contributions. In this work, we provide several polynomial-time data reduction rules for MINIMUM-FLIP CONSENSUS TREE that lead to a problem kernel containing $O(k^3)$ vertices. This is the first non-trivial kernelization result for MINIMUM-FLIP CONSENSUS TREE. Combining our kernelization algorithm with the search tree by Böcker et al. [1], we achieve a running time of $O(4.83^k + \text{poly}(|V_c|, |V_t|))$ instead of the previous $O(4.83^k \cdot \text{poly}(|V_c|, |V_t|))$. Furthermore, we describe one of the data reduction rules in a fairly abstract and general way, making it applicable to a wide range of Π-EDGE MODIFICATION problems. Due to the lack of space, several details are deferred to a full version of the paper.

2 Preliminaries

The open *neighborhood* $N_G(v)$ of a vertex $v \in V$ is the set of vertices that are adjacent to v in $G = (V, E)$. For a set of vertices $V' \subseteq V$, the induced subgraph $G[V']$ is the graph over the vertex set V' with edge set $\{\{v, w\} \in E \mid v, w \in V'\}$. For $V' \subseteq V$ we use $G - V'$ as abbreviation for $G[V \setminus V']$ and for a vertex $v \in V$ let $G - v$ denote $G - \{v\}$. For two sets X and Y with $X \cap Y = \emptyset$, let $E_{X,Y}$ denote the set $\{\{x, y\} \mid x \in X \wedge y \in Y\}$. As an abbreviation for $E_{\{x\},Y}$ we write $E_{x,Y}$. For two sets E and F, define $E \Delta F := (E \setminus F) \cup (F \setminus E)$ (the symmetric difference). Further, for a bipartite graph $G = (V_c, V_t, E)$ and a set $F \subseteq E_{V_c,V_t}$ define $G \Delta F := (V_c, V_t, E \Delta F)$. Sometimes we refer to a vertex $c \in V_c$ as c-vertex, and to a vertex $t \in V_t$ as t-vertex. A graph property Π is called *hereditary* if it holds for all induced subgraphs of a graph G with Π. That is, the class of graphs with a hereditary graph property Π is closed under vertex deletion. Clearly, all graph properties that can be described by a (possibly non-finite) set of forbidden induced subgraphs (such as M-freeness for example) are hereditary. Two c-vertices c_1 and c_2 are said to be *in conflict* if there exists an induced M-graph containing both of them. It is not hard to see that two vertices $c_1, c_2 \in V_c$ are in conflict iff

$$(N_G(c_1) \setminus N_G(c_2) \neq \emptyset) \wedge (N_G(c_1) \cap N_G(c_2) \neq \emptyset) \wedge (N_G(c_2) \setminus N_G(c_1) \neq \emptyset).$$

For our data reduction we crucially use a structure called *critical independent set*.

DEFINITION 1. *Given an undirected graph $G = (V, E)$, a set $I \subseteq V$ is called a* critical independent set *if for any two vertices $v, w \in I$ it holds that v and w are non-adjacent, $N_G(v) = N_G(w)$, and I is maximal with respect to this property.*

All critical independent sets of a graph can be found in linear time [16]. Given a graph $G = (V, E)$ and the collection $\mathcal{I} = \{I_1, I_2, \ldots, I_q\}$ of its critical independent sets,

where $q \leq n$, the *critical independent set graph* of G is the undirected graph $(\mathcal{I}, \mathcal{E})$ with $\{I_i, I_j\} \in \mathcal{E}$ iff $\forall u \in I_i, v \in I_j : \{u, v\} \in E$.

A bipartite graph $G = (X, Y, E)$ is called a *chain graph* if the neighborhoods of the vertices in X form a chain [24]. That is, there is an ordering of the vertices in X, say $x_1, x_2, \ldots, x_{|X|}$, such that $N_G(x_1) \subseteq N_G(x_2) \subseteq \ldots \subseteq N_G(x_{|X|})$. It is easy to see that the neighborhoods of Y also form a chain if G is a chain graph. Moreover, a bipartite graph is a chain graph iff it is $2K_2$-free [24] (herein, a $2K_2$ is the graph that consists of two independent edges). Since every M-graph contains an induced $2K_2$, the set of chain graphs is contained in the class of M-free graphs. One of our data reduction rules is based on identifying and reducing the size of subgraphs of the input graphs that are chain graphs and additionally have a special neighborhood structure.

Parameterized algorithmics [19] aims at a multivariate complexity analysis of problems. This is done by studying relevant problem parameters and their influence on the computational complexity. The decisive question is whether a given parameterized problem is *fixed-parameter tractable (FPT)* with respect to the parameter k. In other words, here we ask for the existence of a solving algorithm with running time $f(k) \cdot \text{poly}(n)$ for some computable function f. A core tool in the development of parameterized algorithms that has been recognized as one of the most important contribution of parameterized algorithmics to practical computing [6, 14, 17, 19] is polynomial-time preprocessing by *data reduction rules*, often yielding a *problem kernel*. Herein, the goal is, given any problem instance G with parameter k, to transform it in polynomial time into a new instance G' with parameter k' such that the size of G' is bounded from above by some function only depending on k, $k' \leq k$, and (G, k) is a yes-instance iff (G', k') is a yes-instance. We call a data reduction rule *correct* if the new instance after an application of this rule is a yes-instance iff the original instance is a yes-instance. An instance is called *reduced* with respect to some data reduction rule if the data reduction rule has been exhaustively applied.

3 A Universal Rule for Critical Independent Sets

In this section, we describe a polynomial-time data reduction rule for parameterized graph modification problems that applies to a certain kind of hereditary graph property and is a generalization of a rule that was developed for BICLUSTER EDITING [22]. Here, we prove the new result that this reduction rule can be applied to a wide range of Π-EDGE MODIFICATION problems, including MINIMUM-FLIP CONSENSUS TREE.

The basic idea of the data reduction is to show that, for some graph properties, vertices that belong to the same critical independent set are subject to the "same" edge modifications. Therefore, large critical independent sets can be reduced. First, we give a description of these graph properties. Let Π be a hereditary graph property. We call Π *critical independent set preserving* (*cisp*) whenever for all forbidden induced subgraphs F of Π, there are no two vertices $u, v \in V(F)$ that form a critical independent set in F (that is, all critical independent sets of F have size one). Note that M-freeness is a cisp graph property: all vertices in an induced M-graph have different neighborhoods. Therefore, the following lemmas and reduction rule apply directly to MINIMUM-FLIP CONSENSUS TREE. First, we can show that cisp graph properties are closed under a certain vertex-addition operation.

LEMMA 2. *Let $G = (V, E)$ be a graph fulfilling a cisp graph property Π. Let G' be the graph that results by adding to G a new vertex $x \notin V$ and making it adjacent to $N_G(v)$ for an arbitrary vertex $v \in V$. Then, G' also fulfills Π.*

Using Lemma 2, we can show that for graph modification problems for cisp properties there is an optimal solution that treats the vertices of a critical independent set equally.

LEMMA 3. *Let $I \subseteq V$ be a critical independent set in $G = (V, E)$, and let Π be a cisp graph property. Then there exists a minimum-cardinality edge modification set S such that $G' := G\Delta S$ fulfills Π and I is part of a critical independent set in G'.*

With Lemma 3 at hand, the following data reduction rule is not hard to see.

REDUCTION RULE 1. *Let $I \subseteq V$ be a critical independent set. If $|I| > k+1$, then delete $|I| - (k+1)$ arbitrary vertices from I.*

LEMMA 4. *Reduction Rule 1 is correct and can be exhaustively applied in $O(|V| + |E|)$ time.*

This general data reduction rule also applies to the COMPLETION and DELETION version of a Π-EDGE MODIFICATION problem for a cisp graph property Π. Examples for graph modification problems to which this rule can be applied are CHAIN DELETION and COTRIVIALLY PERFECT DELETION.[‡]

4 Specific Data Reduction Rules for Minimum-Flip Consensus Tree

In this section, we present three further polynomial-time data reduction rules that together with Reduction Rule 1 produce an $O(k^3)$-vertex kernel. The first reduction rule is obvious.

REDUCTION RULE 2. *Remove M-free connected components from the input graph.*

The next reduction rule removes c-vertices from G that do not appear in an M-graph.

REDUCTION RULE 3. *Let $G = (V_c, V_t, E)$ be a bipartite graph. If there exists a vertex $c \in V_c$ that is not in conflict with any other vertex in V_c, then remove c.*

LEMMA 5. *Reduction Rule 3 is correct and can be exhaustively applied in $O(|V_c|^2 \cdot |V_t|)$ time.*

PROOF. Let G be the original graph and let $G' := G - c$, where $c \in V_c$ is not in conflict with any other c-vertex. First, we prove the correctness of Reduction Rule 3. To this end, we show the following.

Claim: (G, k) is a yes-instance iff (G', k) is a yes-instance.

"$\Rightarrow$:" Follows directly because M-freeness is a hereditary graph property.

"$\Leftarrow$:" This direction is based on the observation that graph G' can be decomposed into two edge disjoint subgraphs G_1 and G_2 that can be solved independently from each other,

[‡]Definitions and kernelization results for these problems have been obtained by Guo [12].

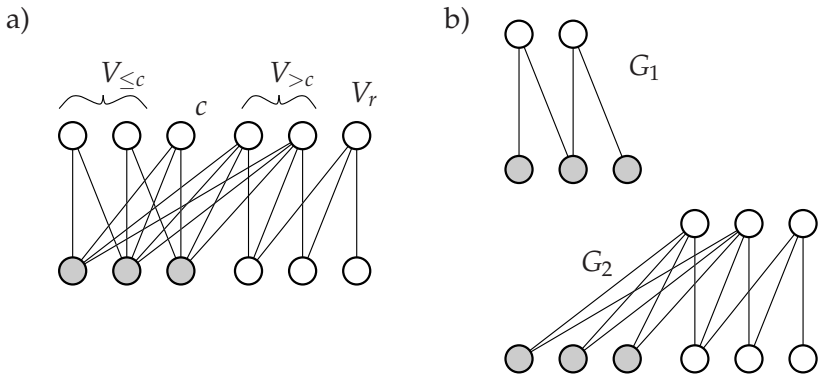


Figure 2: Correctness of Reduction Rule 3. a) Partition of the vertices in V_c depending on their relation to c. The neighbors of c are colored gray. b) The graphs G_1 (induced by $V_{\leq c}$ and $N(c)$) and G_2 (induced by $V_{>c}$,V_r, and V_t) .

without creating a new conflict containing c. We need the following notation.

$$V_{>c} := \{c' \in V_c \mid N(c) \subsetneq N(c')\},$$
$$V_{\leq c} := \{c' \in (V_c \setminus \{c\}) \mid N(c') \subseteq N(c)\}, \text{ and}$$
$$V_r := V_c \setminus (V_{\leq c} \cup V_{>c}).$$

See Fig. 2 a) for an example. Note that, since c is not in conflict with any other vertex $c' \in V_c - c$, either $N_G(c) \cap N_G(c') = \emptyset$ or $c' \in (V_{\leq c} \cup V_{>c})$. In particular, this implies that for every vertex $c' \in V_r$ it holds that $N_G(c) \cap N_G(c') = \emptyset$.

Let F' be a solution for (G', k). We show that from F' we can compute a solution F for (G, k). Let $V_2 := V_r \cup V_{>c}$. Consider the two graphs $G_1 := G[V_{\leq c} \cup N_G(c)]$ and $G_2 := G[V_2 \cup V_t]$. See Fig. 2 b) for an example. Observe that $F_1 := F' \cap E_{V_{\leq c}, N_G(c)}$ is a solution for G_1 and $F_2 := F' \cap E_{V_2, V_t}$ is a solution for G_2, since G_1 and G_2 are induced subgraphs of G'. Furthermore, note that $F_1 \cap F_2 = \emptyset$ since $V_{\leq c} \cap V_2 = \emptyset$. As a consequence, $|F_1| + |F_2| \leq |F'| \leq k$.

Consider the graph G_2. It is easy to observe that $N_G(c)$ is contained in a critical independent set in G_2. This can be seen as follows: since $N_G(c) \subset N_G(c')$ for every vertex $c' \in V_{>c}$ and $N_G(c) \cap N_G(c'') = \emptyset$ for every vertex $c'' \in V_r$, every vertex $t \in N_G(c)$ is adjacent in G_2 to exactly the vertices in $V_{>c}$. Since $N_G(c)$ is a critical independent set in G_2, according to Lemma 3 there exists a minimum-cardinality solution F_2' for G_2 such that $N_G(c)$ is contained in a critical independent set in $G_2 \Delta F_2'$. Clearly, $|F_2'| \leq |F_2|$.

Based on these facts, we show that $F := F_1 \cup F_2'$ is a solution for (G, k). First of all, note that by the discussion above $|F| = |F_1| + |F_2'| \leq |F_1| + |F_2| \leq k$. Second, no two vertices in V_c are in conflict, and hence, $G\Delta F$ is M-free. This can be seen as follows. Since F_1 is a solution for G_1, any two vertices $c_1, c_2 \in V_{\leq c}$ are not in conflict in $G\Delta F$. The same holds true for any two vertices in V_2, since $G_2 \Delta F_2'$ is M-free. Moreover, since for every vertex $c' \in V_{\leq c}$ it holds that $N_{G\Delta F}(c') = N_{G_1 \Delta F_1}(c') \subseteq N_G(c) = N_{G\Delta F}(c)$, c is not in conflict with any vertex in $V_{\leq c}$. Finally, since $N_G(c)$ is a critical independent set in $G_2 \Delta F_2'$, we know that for every $c' \in V_2$ either $N_{G\Delta F}(c') \cap N_G(c) = \emptyset$ or $N_G(c) \subseteq N_{G\Delta F}(c')$ and hence c' is not in conflict with any

vertex $c'' \in V_{\leq c} \cup \{c\}$. Therefore, $G\Delta F$ is M-free.

For the running time consider the following. For each pair of vertices $c_1, c_2 \in V_c$, we can determine in $O(V_t)$ time whether they are in conflict by checking for each vertex $t \in V_t$, whether it is adjacent to c_1, c_2, or both. Each c-vertex that is in conflict with some other vertex is marked. Finally, unmarked vertices are removed from the graph. This can be performed in $O(|E|)$ time. The overall running time is thus $O(|V_c|^2 \cdot |V_t|)$. ∎

The structurally "deepest" reduction rule shrinks subgraphs of the input graph that resemble "local" chain graphs. We call such a subgraph *P-structure*:

DEFINITION 6. *Let $G = (V_c, V_t, E)$ be a bipartite graph. A tuple (C_P, T_P) of two subsets $C_P \subseteq V_c$ and $T_P \subseteq V_t$ forms a* P-structure *if the following three properties are fulfilled:*

1. *$G[C_P \cup T_P]$ is a chain graph,*
2. *for all $c', c'' \in C_P$ it holds that $N(c') \setminus T_P = N(c'') \setminus T_P$, and*
3. *for all $t', t'' \in T_P$ it holds that $N(t') \setminus C_P = N(t'') \setminus C_P$.*

It is easy to see that for a P-structure (C_P, T_P) of a bipartite graph G the neighborhoods in G of the vertices in C_P (and T_P) also form a chain (since "outside" of the P-structure they have the same neighbors). Moreover, note that the vertices of a P-structure form a subgraph that is M-free.

REDUCTION RULE 4. *Let (C_P, T_P) be a P-structure in a bipartite graph $G = (V_c, V_t, E)$. Let $T_P = \{t_1, t_2, \ldots, t_l\}$ such that $N(t_1) \subseteq N(t_2) \subseteq \ldots \subseteq N(t_l)$. If $l > 2(k+1)$, then remove $t_{k+2}, t_{k+3}, \ldots, t_{l-(k+1)}$ from G.*

LEMMA 7. *Reduction Rule 4 is correct and can be exhaustively applied in polynomial time.*

We can find P-structures in polynomial time by trying all possibilities for choosing the four "endpoints" t_1, t_l, c_1, c_q of the chain, where $N(t_1) \subseteq N(t_l)$ and $N(c_q) \subseteq N(c_1)$. It is not hard to see that in the case that t_1, t_l, c_1, c_q are indeed endpoints of a P-structure, we can reconstruct the corresponding P-structure as follows:

$$C_P = (N(t_l) \setminus N(t_1)) \cup \{c_1\} \cup \{c' \in V_v \mid N(c') = N(c_1)\}$$

and analogously

$$T_P = (N(c_1) \setminus N(c_q)) \cup \{t_l\} \cup \{t' \in V_t \mid N(t') = N(t_l)\}.$$

To recognize the cases that t_1, t_l, c_1, c_q are not the endpoints of a chain, we have to check whether the found vertex sets indeed form a P-structure. This approach works clearly in polynomial time, although there seems to be room for improving the efficiency, a task for future research.

5 Mathematical Analysis of the Problem Kernel Size

In this section, we bound the maximum number of vertices in a reduced instance. We need the following notation concerning rooted trees. We use *node* to refer to a vertex of a tree. For a rooted tree T let $L(T)$ denote the *leaves* of T (that is, the nodes of degree one). The nodes

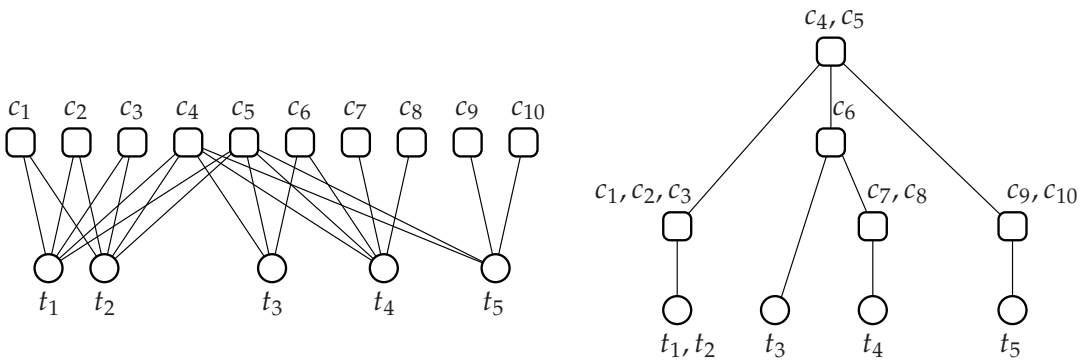


Figure 3: An M-free graph G and the corresponding tree $T_{cis}(G)$.

in $V(T) \setminus L(T)$ are denoted as *inner nodes*. The root of T is denoted by $r(T)$. Moreover, for a node $v \in V(T)$, the subtree rooted at v is denoted by T_v. We classify the children of a node as follows. We refer to a child of a node as its *leaf child* if it is a leaf , otherwise it is called its *non-leaf child*. We speak of the leaves (inner nodes) of a forest to refer to the union of the leaves (inner nodes) of the trees of the forest.

Given a connected and M-free graph $G = (V_c, V_t, E)$, one can construct a rooted tree T with node set $V_t \cup V_c$ and with $L(T) = V_t$ such that $t_i \in V_t$ is a descendant of $c_j \in V_c$ iff $t_i \in N_G(c_j)$, see [4, 15, 21] for details. Note that the critical independent set graph of an M-free graph is M-free. Hence, we can find a tree with the property that every leaf one-to-one corresponds to a critical independent set of the t-vertices and every inner vertex one-to-one corresponds to a critical independent set of the c-vertices. For an M-free graph G, this tree is denoted by $T_{cis}(G)$. Figure 3 shows an M-free graph G together with $T_{cis}(G)$.

The following easy observations are helpful in the analysis of the kernel size.

1. Every inner vertex of T_{cis} has at most one leaf child, and
2. every inner vertex with at most one non-leaf child has exactly one leaf child.

Now, we arrive at our main result.

THEOREM 8. MINIMUM-FLIP CONSENSUS TREE *admits an $O(k^3)$-vertex problem kernel .*

PROOF. Consider a reduced instance $(G = (V_c, V_t, E), k)$. We show that if (G, k) is a yes-instance, then the number of vertices in $V_c \cup V_t$ is bounded by $O(k^3)$.

If (G, k) is a yes-instance, then there exists an optimal solution S of size at most k. That is, the graph $G_S := G\Delta S$ is M-free. Vertices that are involved in an edge modification are called *affected* in the following. Let X_c denote the c-vertices that are affected by an edge modification in S and let Y_c denote the c-vertices that are not affected by any edge modification. Analogously, we define X_t and Y_t. Note that since every edge modification involves a c-vertex and a t-vertex, we have that $|X_c| \leq k$ and $|X_t| \leq k$.

Let $G_{S,1}, G_{S,2}, \ldots, G_{S,p}$ denote the connected components of G_S. Recall that for every connected component $T_i := T_{cis}(G_{S,i})$ denotes the rooted tree corresponding to the critical independent set graph of $G_{S,i}$. Moreover, let T denote the forest containing all T_i. Recall that the leaves of T one-to-one correspond to the critical independent sets of V_t in G_S and that

the inner nodes of T one-to-one correspond to the critical independent sets of V_c in G_S. For a node $z \in V(T)$, let $\mathcal{C}(z)$ denote the set of vertices contained in the critical independent set corresponding to z. Moreover, for $Z \subseteq V(T)$, we define $\mathcal{C}(Z) := \bigcup_{z \in Z} \mathcal{C}(z)$.

We partition the set of inner nodes into three sets A, B, and Q as follows. The set A contains all inner nodes z for which it holds that either $\mathcal{C}(z) \cap X_c \neq \emptyset$ or z has a leaf child w with $\mathcal{C}(w) \cap X_t \neq \emptyset$. Note that A has cardinality at most $2k$ since there are at most $2k$ affected vertices. Moreover, let B contain the inner nodes that are not contained in A and that have at least two non-leaf children. Finally, Q contains all inner nodes not contained in $A \cup B$.

Next, we bound the number of the vertices contained in the critical independent sets corresponding to the nodes in $A \cup B$ and their leaf children. To this end, we show the following.

1. For every inner node x not contained in A, there exists at least one node $y \in V(T_x)$ with $y \in A$.
2. The cardinality of B is at most $2k$.
3. Let $L_{A,B}$ denote the leaves adjacent to the nodes in $A \cup B$. The number of vertices contained in the critical independent sets corresponding to the nodes in $A \cup B \cup L_{A,B}$ is $O(k^2)$.

1.) Assume that there exists an inner node $x \in V(T) \setminus (L(T) \cup A)$ such that $V(T_x) \cap A = \emptyset$. That is, no vertex in $\mathcal{C}(V(T_x))$ is affected. Consider a vertex $c \in \mathcal{C}(x)$. We show that c is not contained in any conflict in G, contradicting the fact that G is reduced with respect to Reduction Rule 3. First, for every vertex $y \in \mathcal{C}(V(T_x))$, it holds that $N_G(y) \subseteq N_G(c)$ since $N_{G_S}(y) \subseteq N_{G_S}(c)$ and S does not affect c or y. Second, for every vertex $y \in \mathcal{C}(V(T) \setminus V(T_x))$, it holds that $N_{G_S}(c) \cap N_{G_S}(y) = \emptyset$ or $N_{G_S}(c) \subseteq N_{G_S}(y)$. But since neither c nor any vertex in $N_{G_S}(c)$ is modified, this implies that $N_G(c) \cap N_G(y) = \emptyset$ or $N_G(c) \subseteq N_G(y)$. This means that c is not contained in any conflict in G.

2.) Consider the forest T' that results from deleting all leaves of T. Note that B is a subset of the nodes from T' with at least two children. From 1) it follows directly that the leaves of T' are contained in A and, hence, their number is bounded by $2k$. Since the number of inner nodes with at least two children is bounded by the number of leaves, we get that $|B| \leq 2k$.

3.) First, note that $|A \cup B| \leq 4k$ since A and B each have cardinality at most $2k$. Moreover, $|L_{A,B}| \leq 4k$ since every inner node has at most one leaf child. For every node $y \in A \cup B \cup L_{A,B}$, define $\mathcal{C}'(y) := \mathcal{C}(y) \setminus (X_c \cup X_t)$. For every $y \in A \cup B \cup L_{A,B}$, since no vertex in $\mathcal{C}'(y)$ is affected, $\mathcal{C}'(y)$ forms a critical independent set in G and—since G is reduced with respect to Reduction Rule 1—we thus get that $|\mathcal{C}'(y)| \leq k + 1$. Putting all together, we obtain

$$|\mathcal{C}(A \cup B \cup L_{A,B})| \leq |X_c| + |X_t| + \sum_{y \in A \cup B \cup L_{A,B}} |\mathcal{C}'(y)| \leq 2k + 4k(k+1).$$

It remains to bound the number of the vertices contained in $\mathcal{C}(Q \cup L_Q)$, where L_Q denotes the leaves adjacent to the nodes in Q. Observe that each inner node contained in Q (and hence not contained in $A \cup B$) has exactly one leaf and one non-leaf child. That is, in the the forest $T' := T - L(T)$ these vertices have degree two. Recall that all leaves of T' (see 2.) above) are contained in A and hence $|L(T')| \leq 2k$. Consider a path $P =$

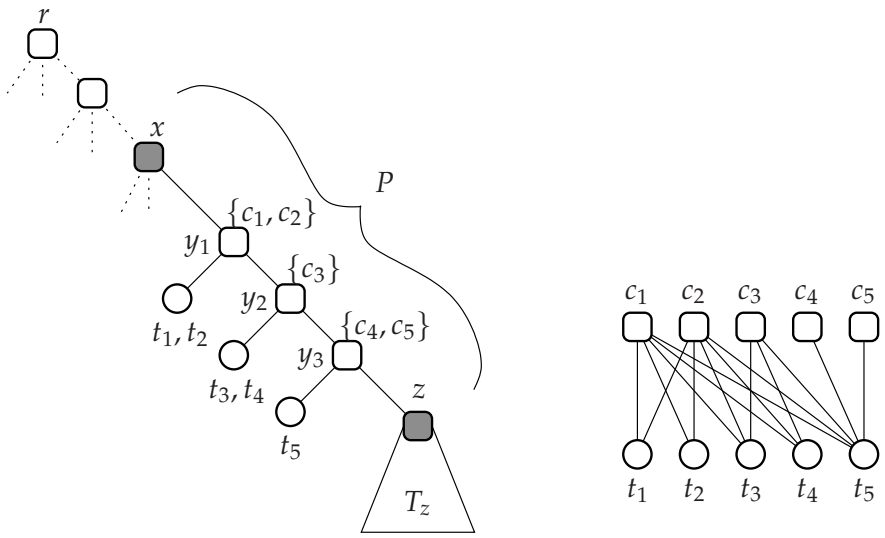


Figure 4: A degree-two-path P and the corresponding chain graph. Herein, $\mathcal{C}(y_1) = \{c_1, c_2\}$, $\mathcal{C}(y_2) = \{c_3\}$, $\mathcal{C}(y_3) = \{c_4, c_5\}$, and $\mathcal{C}(y_4) = \{c_6\}$.

$(\{x, y_1\}, \{y_1, y_2\}, \ldots, \{y_{l-1}, y_l\}, \{y_l, z\})$ in T' with $y_i \in Q$ for all $1 \leq i \leq l$ and $x, z \in A \cup B$. Such a path is called a *degree-two-path* in the following since by the above discussion $\deg_{T'}(y_i) = 2$ for all $1 \leq j \leq l$. Further, for every y_i, let w_i denote the leaf child of y_i in T. Note that in the forest T', there are at most $8k$ degree-two-paths since $L(T') \subseteq A$ and $|A \cup B| \leq 4k$. In the following, we bound the length of each degree-two-path by $2(k+1)$. Hence, for each such path we have

$$\sum_{i=1}^{l} (|\mathcal{C}(y_i)| + |\mathcal{C}(w_i)|) \leq l \cdot (2(k+1)) \leq (2(k+2)) \cdot 2(k+1)$$

vertices in G. Adding up over the at most $8k$ degree-two-paths, this amounts to $8k \cdot 2(k+1)(2(k+2)) \leq 32k(k+1)(k+2)$ vertices, yielding the bound of $O(k^3)$ vertices in total.

Next, we bound the length of each degree-two-path. To this end, consider such a degree-two-path $P = (\{x, y_1\}, \{y_1, y_2\}, \ldots, \{y_{l-1}, y_l\}, \{y_l, z\})$ in T', that is, $x, z \in A \cup B$ and $y_i \in Q$ for all $1 \leq i \leq l$. Without loss of generality, we assume that y_l is a descendent of y_1. See Fig. 4 for an example. Let $C_P := \bigcup_{i=1}^{l} \mathcal{C}(y_i)$ and $T_P := \bigcup_{i=1}^{l} \mathcal{C}(w_i)$.

We show that (C_P, T_P) forms a P-structure in G. First, note that $C_P \subseteq V_c$ and $T_P \subseteq V_t$. Next, note that $G[C_P \cup T_P]$ forms a chain graph. This can seen as follows. In G_S a vertex in $\mathcal{C}(y_1)$ is clearly adjacent to all vertices in T_P, a vertex in $\mathcal{C}(y_2)$ is adjacent to all vertices in $T_P \setminus \mathcal{C}(w_1)$, a vertex in $\mathcal{C}(y_3)$ is adjacent to all vertices in $T_P \setminus \mathcal{C}(\{w_1, w_2\})$, and so on. Hence, $G_S[C_P \cup T_P]$ is a chain graph and, since no vertex in C_P is involved in an edge modification, we have that $G[C_P \cup T_P]$ forms a chain graph, too (see Fig. 4). Next, we show that C_P and T_P fulfill the second and third property of a P-structure. On the one hand, every vertex in C_P is adjacent in G_S to all vertices contained in the critical independent sets corresponding to the leaves in T_z and, hence, for all $c, c' \in C_P$, we have $N_{G_S}(c) \setminus T_P = N_{G_S}(c') \setminus T_P$. Since no vertex in C_P is affected, this implies that $N_G(c) \setminus T_P = N_G(c') \setminus T_P$ for

all $c, c' \in C_P$. On the other hand, every vertex $t \in T_P$ is adjacent in G_S (and hence in G) to all c-vertices contained in a critical independent set on the path from the root r to z. Hence, for any two vertices $t, t' \in T_P$ it holds that $N_G(t) \setminus T_P = N_G(t') \setminus T_P$. In summary, (C_P, T_P) forms a P-structure.

Finally, we show that $l \leq 2(k+1)$. Assume towards a contradiction that $l > 2(k+1)$. This implies that $|T_P| > 2(k+1)$, too, since every y_i has exactly one leaf child that corresponds to a (non-empty) critical independent set of V_t. Hence, $|T_P| > 2(k+1)$ and thus all conditions to apply Reduction Rule 4 are fulfilled: a contradiction to the fact that G is reduced. ∎

Applying the technique of interleaving [20] to our kernelization and the search tree algorithm by Böcker et al. [1], we obtain an "additive FPT" algorithm for MINIMUM-FLIP CONSENSUS TREE.

COROLLARY 9. MINIMUM-FLIP CONSENSUS TREE *can be solved in running time* $O(4.83^k + \text{poly}(|V_c|, |V_t|))$.

6 Conclusion

As to future research, first of all, we want to implement and test the efficiency of our data reduction rules. Second, improving the polynomial running time of our data reduction rules is desirable. Obviously, obtaining data reduction rules that lead to a quadratic-vertex or linear-vertex kernel remains as an open question. Moreover, studying edge-weighted problem variants would be theoretically interesting. Finally, it would be interesting to adapt our data reduction to yield a full kernel (see [5]) for MINIMUM-FLIP CONSENSUS TREE.

References

[1] S. Böcker, Q. B. Bui, and A. Truss. An improved fixed-parameter algorithm for minimum-flip consensus trees. In *Proc. 3rd IWPEC*, volume 5018 of *LNCS*, pages 43–54. Springer, 2008.

[2] H. L. Bodlaender. A cubic kernel for feedback vertex set. In *Proc. 24nd STACS*, volume 4393 of *LNCS*, pages 320–331. Springer, 2007.

[3] P. Burzyn, F. Bonomo, and G. Durán. NP-completeness results for edge modification problems. *Discrete Applied Mathematics*, 154(13):1824–1844, 2006.

[4] D. Chen, O. Eulenstein, D. Fernández-Baca, and M. Sanderson. Minimum-flip supertrees: Complexity and algorithms. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 3(2):165–173, 2006. Preliminary version presented at *COCOON '02*.

[5] P. Damaschke. Parameterized enumeration, transversals, and imperfect phylogeny reconstruction. *Theoretical Computer Science*, 351(3):337–350, 2006.

[6] M. R. Fellows. The lost continent of polynomial time: Preprocessing and kernelization. In *Proc. 2nd IWPEC*, volume 4169 of *LNCS*, pages 276–277. Springer, 2006.

[7] M. R. Fellows, M. A. Langston, F. A. Rosamond, and P. Shaw. Efficient parameterized preprocessing for cluster editing. In *Proc. 16th FCT*, volume 4639 of *LNCS*, pages 312–321. Springer, 2007.

[8] H. Fleischner, E. Mujuni, D. Paulusma, and S. Szeider. Covering graphs with few complete bipartite subgraphs. In *Proc. 27th FSTTCS*, volume 4855 of *Lecture Notes in Computer Science*, pages 340–351. Springer, 2007.

[9] J. Gramm, J. Guo, F. Hüffner, and R. Niedermeier. Graph-modeled data clustering: Exact algorithms for clique generation. *Theory of Computing Systems*, 38(4):373–392, 2005.

[10] J. Gramm, J. Guo, F. Hüffner, and R. Niedermeier. Data reduction, exact, and heuristic algorithms for clique cover. In *Proc. 8th ALENEX*, pages 86–94. SIAM, 2006. Long version to appear in *ACM Journal of Experimental Algorithmics*.

[11] J. Guo. A more effective linear kernelization for Cluster Editing. In *Proc. 1st ESCAPE*, volume 4614 of *LNCS*, pages 36–47. Springer, 2007. Long version to appear in *Theoretical Computer Science*.

[12] J. Guo. Problem kernels for NP-complete edge deletion problems: split and related graphs. In *Proc. 18th ISAAC*, volume 4835 of *LNCS*, pages 915–926. Springer, 2007.

[13] J. Guo, F. Hüffner, C. Komusiewicz, and Y. Zhang. Improved algorithms for bicluster editing. In *Proc. 5th TAMC*, volume 4978 of *LNCS*. Springer, 2008.

[14] J. Guo and R. Niedermeier. Invitation to data reduction and problem kernelization. *ACM SIGACT News*, 38(1):31–45, 2007.

[15] D. Gusfield. Efficient algorithms for inferring evolutionary trees. *Networks*, 21:19–28, 1991.

[16] W. Hsu and T. Ma. Substitution decomposition on chordal graphs and applications. In *Proc. 2nd International Symposium on Algorithms*, volume 557 of *LNCS*, pages 52–60. Springer, 1991.

[17] F. Hüffner, R. Niedermeier, and S. Wernicke. Techniques for practical fixed-parameter algorithms. *The Computer Journal*, 51(1):7–25, 2008.

[18] A. Natanzon, R. Shamir, and R. Sharan. Complexity classification of some edge modification problems. *Discrete Applied Mathematics*, 113(1):109–128, 2001.

[19] R. Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Number 31 in Oxford Lecture Series in Mathematics and Its Applications. Oxford University Press, 2006.

[20] R. Niedermeier and P. Rossmanith. A general method to speed up fixed-parameter-tractable algorithms. *Information Processing Letters*, 73:125–129, 2000.

[21] I. Pe'er, T. Pupko, R. Shamir, and R. Sharan. Incomplete directed perfect phylogeny. *SIAM Journal on Computing*, 33(3):590–607, 2004.

[22] F. Protti, M. D. da Silva, and J. L. Szwarcfiter. Applying modular decomposition to parameterized cluster editing problems. *Theory of Computing Systems*. To appear.

[23] S. Thomasse. A quadratic kernel for feedback vertex set. In *Proc. 20th SODA*. SIAM, 2009. To appear.

[24] M. Yannakakis. Computing the minimum fill-in is NP-complete. *SIAM Journal on Algebraic Discrete Methods*, 2(1):77–79, 1981.

Foundations of Software Technology and Theoretical Computer Science (Bangalore) 2008.
Editors: R. Hariharan, M. Mukund, V. Vinay; pp 292-303

Leaf languages and string compression*

Markus Lohrey
Universität Leipzig, Institut für Informatik, Germany
lohrey@informatik.uni-leipzig.de

ABSTRACT. Tight connections between leafs languages and strings compressed via straight-line programs (SLPs) are established. It is shown that the compressed membership problem for a language L is complete for the leaf language class defined by L via logspace machines. A more difficult variant of the compressed membership problem for L is shown to be complete for the leaf language class defined by L via polynomial time machines. As a corollary, a fixed linear visibly pushdown language with a PSPACE-complete compressed membership problem is obtained. For XML languages, the compressed membership problem is shown to be coNP-complete.

1 Introduction

Leaf languages were introduced in [7, 25] and became an important concept in complexity theory. Let us consider a nondeterministic Turing machine M. For a given input x, one considers the yield string of the computation tree (i.e. the string obtained by listing all leafs from left to right), where accepting (resp. rejecting) leaf configurations yield the letter 1 (resp. 0). This string is called the *leaf string* corresponding to the input x. For a given language $K \subseteq \{0,1\}^*$ let $\mathrm{LEAF}(M,K)$ denote the set of all inputs for M such that the corresponding leaf string belongs to K. By fixing K and taking for M all nondeterministic polynomial time machines, one obtains the polynomial time leaf language class $\mathrm{LEAF}_a^P(K)$. The index a indicates that we allow Turing machines with arbitrary (non-balanced) computation trees. If we restrict to machines with balanced computation trees, we obtain the class $\mathrm{LEAF}_b^P(K)$, see [13, 16] for a discussion of the different shapes for computation trees.

Many complexity classes can be defined in a uniform way with this construction. For instance, $\mathrm{NP} = \mathrm{LEAF}_x^P(0^*1\{0,1\}^*)$ and $\mathrm{coNP} = \mathrm{LEAF}_x^P(1^*)$ for both $x = a$ and $x = b$. In [14], it was shown that $\mathrm{PSPACE} = \mathrm{LEAF}_b^P(K)$ for a fixed regular language K. In [16], logspace leaf language classes $\mathrm{LEAF}_a^L(K)$ and $\mathrm{LEAF}_b^L(K)$, where M varies over all (resp. all balanced) nondeterministic logspace machines, were investigated. Among other results, a fixed deterministic context-free language K with $\mathrm{PSPACE} = \mathrm{LEAF}_a^L(K)$ was presented. In [8], it was shown that in fact a fixed deterministic *one-counter* language K as well as a fixed *linear* deterministic context-free language [15] suffices in order to obtain PSPACE. Here "linear" means that the pushdown automaton makes only one turn.

In [6, 24], a tight connection between leaf languages and computational problems for succinct input representations was established. More precisely, it was shown that the membership problem for a language $K \subseteq \{0,1\}^*$ is complete (w.r.t. polynomial time reductions in [6] and projection reductions in [24]) for the leaf language class $\mathrm{LEAF}_b^P(K)$, if the input string x is represented by a Boolean circuit. A Boolean circuit $C(x_1,\ldots,x_n)$ with n inputs represents a string x of length 2^n in the natural way: the i-th position in x carries a 1 if

*This work is supported by the DFG research project ALKODA.

FSTTCS 2008
IARCS Annual Conference on
Foundations of Software Technology and Theoretical Computer Science
http://drops.dagstuhl.de/opus/volltexte/2008/1761

and only if $C(a_1, \ldots, a_n) = 1$, where $a_1 \cdots a_n$ is the n-bit binary representation of i. In this paper we consider another more practical compressed representation for strings, namely *straight-line programs* (SLPs) [23]. A straight-line program is a context-free grammar $\mathbb{A}$ that generates exactly one string $\mathrm{val}(\mathbb{A})$. In an SLP, repeated subpatterns in a string have to be represented only once by introducing a nonterminal for the pattern. An SLP with n productions can generate a string of length 2^n by repeated doubling. Hence, an SLP can be seen indeed as a compressed representation of the string it generates. Several other dictionary-based compressed representations, like for instance Lempel-Ziv (LZ) factorizations, can be converted in polynomial time into SLPs and vice versa [23]. This implies that complexity results can be transfered from SLP-encoded input strings to LZ-encoded input strings.

Algorithmic problems for SLP-compressed strings were studied e.g. in [5, 18, 19, 20, 22, 23]. A central problem in this context is the *compressed membership problem* for a language K: it is asked whether $\mathrm{val}(\mathbb{A}) \in K$ for a given SLP $\mathbb{A}$. In [19] it was shown that there exists a fixed linear deterministic context-free language with a PSPACE-complete compressed membership problem. A straightforward argument shows that for every language K, the compressed membership problem for K is complete for the logspace leaf language class $\mathrm{LEAF}_a^L(K)$ (Prop. 2). As a consequence, the existence of a linear deterministic context-free language with a PSPACE-complete compressed membership problem [19] can be deduced from the above mentioned LEAF_a^L-characterization of PSPACE from [8], and vice versa. For polynomial time leaf languages, we reveal a more subtle relationship to SLPs. Recall that the *convolution* $u \otimes v$ of two strings $u, v \in \Sigma^*$ is the string over the paired alphabet $\Sigma \times \Sigma$ that is obtained from gluing u and v in the natural way (we cut off the longer string to the length of the shorter one). We define a fixed projection homomorphism $\rho : \{0,1\} \times \{0,1\} \to \{0,1\}$ such that for every language K, the problem of checking $\rho(\mathrm{val}(\mathbb{A}) \otimes \mathrm{val}(\mathbb{B})) \in K$ for two given SLPs $\mathbb{A}$, $\mathbb{B}$ is complete for the class $\mathrm{LEAF}_b^P(K)$ (Cor. 4). By combining Cor. 4 with the main result from [14] (PSPACE $= \mathrm{LEAF}_b^P(K)$ for a certain regular language K), we obtain a regular language L for which it is PSPACE-complete to check whether the convolution of two SLP-compressed strings belongs to L (Cor. 6). Recently, the convolution of SLP-compressed strings was also studied in [5], where for every $n \geq 0$, SLPs $\mathbb{A}_n, \mathbb{B}_n$ of size $n^{O(1)}$ were constructed such that every SLP for $\mathrm{val}(\mathbb{A}_n) \otimes \mathrm{val}(\mathbb{B}_n)$ has size $\Omega(2^{n/2})$.

From Cor. 6 we obtain a strengthening of one of the above mentioned results from [8] (PSPACE $= \mathrm{LEAF}_a^L(K)$ for a linear deterministic context-free language K as well as a deterministic one-counter language K) to *visibly pushdown languages* [1]. The latter constitute a subclass of the deterministic context-free languages which received a lot of attention in recent years due to its nice closure and decidability properties. Visibly pushdown languages can be recognized by *deterministic* pushdown automata, where it depends only on the input symbol whether the automaton pushes or pops. Visibly pushdown languages were already introduced in [27] as input-driven languages. In [9] it was shown that every visibly pushdown language can be recognized in NC^1; thus the complexity is the same as for regular languages [2]. In contrast to this, there exist linear deterministic context-free languages as well as deterministic one-counter languages with an L-complete membership problem [15]. We show that there exists a linear visibly pushdown language with a PSPACE-complete compressed membership problem (Thm. 7). Together with Prop. 2, it follows that PSPACE $= \mathrm{LEAF}_a^L(K)$ for a linear visibly pushdown language K (Cor. 8).

In [21], nondeterministic finite automata (instead of polynomial time (resp. logspace) Turing-machines) were used as a device for generating leaf strings. This leads to the definition of the leaf language class $\mathrm{LEAF}^{\mathrm{FA}}(K)$. It was shown that $\mathrm{CFL} \subsetneq \mathrm{LEAF}^{\mathrm{FA}}(\mathrm{CFL}) \subseteq \mathrm{DSPACE}(n^2) \cap \mathrm{DTIME}(2^{O(n)})$, and the question for sharper upper and lower bounds was posed. Here we give a partial answer to this question. For the linear visibly pushdown language mentioned in the previous paragraph, the class $\mathrm{LEAF}^{\mathrm{FA}}(K)$ contains a PSPACE-complete language (Thm. 9).

Finally, in Sec. 5 we consider *XML-languages* [4], which constitute a subclass of the visibly pushdown languages. XML-languages are generated by a special kind of context-free grammars (XML-grammars), where every right-hand side of a production is enclosed by a matching pair of brackets. XML-grammars capture the syntactic features of XML document type definitions (DTDs), see [4]. We prove that, unlike for visibly pushdown languages, for every XML-language the compressed membership problem is in coNP and that there are coNP-complete instances.

Proofs that are omitted due to space restriction will appear in a long version.

2 Preliminaries

Let Γ be a finite alphabet. The *empty word* is denoted by ε. Let $s = a_1 \cdots a_n \in \Gamma^*$ be a word over Γ ($n \geq 0$, $a_1, \ldots, a_n \in \Gamma$). The *length* of s is $|s| = n$. For $1 \leq i \leq n$ let $s[i] = a_i$ and for $1 \leq i \leq j \leq n$ let $s[i,j] = a_i a_{i+1} \cdots a_j$. If $i > j$ we set $s[i,j] = \varepsilon$. We denote with $\overline{\Gamma} = \{\overline{a} \mid a \in \Gamma\}$ a disjoint copy of Γ. For $\overline{a} \in \overline{\Gamma}$ let $\overline{\overline{a}} = a$. For $w = a_1 \cdots a_n \in (\Gamma \cup \overline{\Gamma})^*$ let $\overline{w} = \overline{a_n} \cdots \overline{a_1}$. For two strings $u, v \in \Gamma^*$ we define the *convolution* $u \otimes v \in (\Gamma \times \Gamma)^*$ as the string of length $\ell = \min\{|u|, |v|\}$ with $(u \otimes v)[i] = (u[i], v[i])$ for all $1 \leq i \leq \ell$.

A sequence $(u_1, \ldots, u_n)$ of natural numbers is *superdecreasing* if $u_i > u_{i+1} + \cdots + u_n$ for all $1 \leq i \leq n$. An instance of the *subsetsum problem* is a tuple $(w_1, \ldots, w_k, t)$ of binary coded natural numbers. It is a positive instance if there are $x_1, \ldots, x_k \in \{0,1\}$ such that $t = x_1 w_1 + \cdots + x_k w_k$. Subsetsum is a classical NP-complete problem. The *superdecreasing subsetsum* problem is the restriction of subsetsum to instances $(w_1, \ldots, w_k, t)$, where $(w_1, \ldots, w_k)$ is superdecreasing. In [17] it was shown that superdecreasing subsetsum is P-complete ([17] deals with the *superincreasing* subsetsum problem; but the results from [17] can be easily transfered to superdecreasing subsetsum). In fact, something more general is shown in [17]: Let $C(x_1, \ldots, x_m)$ be a Boolean circuit with variable input gates $x_1, \ldots, x_m$ (and some additional input gates that are set to fixed Boolean values). Then from $C(x_1, \ldots, x_m)$ an instance $(t(x_1, \ldots, x_m), w_1, \ldots, w_k)$ of superdecreasing subsetsum is constructed. Here, $t(x_1, \ldots, x_m) = t_0 + x_1 t_1 + \cdots + x_m t_m$ is a linear expression such that:

- $t_1 > t_2 > \cdots > t_m$ and the t_i are pairwise distinct powers of 4. Hence also the sequence $(t_1, \ldots, t_m)$ is superdecreasing.
- For all $a_1, \ldots, a_m \in \{0,1\}$: $C(a_1, \ldots, a_m)$ evaluates to true if and only if $\exists b_1, \ldots, b_k \in \{0,1\} : t_0 + a_1 t_1 + \cdots + a_m t_m = b_1 w_1 + \cdots + b_k w_k$.
- $t_0 + t_1 + \cdots + t_m \leq w_1 + \cdots + w_k$

We encode a superdecreasing sequence $(w_1, \ldots, w_k)$ by the string $S(w_1, \ldots, w_k) \in \{0,1\}^*$ of

length $w_1 + \cdots + w_k + 1$ such that for all $0 \leq p \leq w_1 + \cdots + w_k$:

$$S(w_1, \ldots, w_k)[p+1] = \begin{cases} 1 & \text{if } \exists x_1, \ldots, x_k \in \{0,1\} : p = x_1 w_1 + \cdots + x_k w_k \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

Since $(w_1, \ldots, w_k)$ is superdecreasing, the number of 1's in $S(w_1, \ldots, w_k)$ is 2^k.

The lexicographic order on $\mathbb{N}^*$ is denoted by $\preceq$, i.e. $u \preceq v$ if either u is a prefix of v or there exist $w, x, y \in \mathbb{N}^*$ and $i, j \in \mathbb{N}$ such that $u = wix$, $v = wjy$, and $i < j$. A *finite ordered tree* is a finite set $T \subseteq \mathbb{N}^*$ such that for all $w \in \mathbb{N}^*$, $i \in \mathbb{N}$: if $wi \in T$ then $w, wj \in T$ for every $0 \leq j < i$. The set of *children* of $u \in T$ is $u\mathbb{N} \cap T$. A node $u \in T$ is a leaf of T if it has no children. We say that T is a *full binary tree* if (i) every node has at most two children, and (ii) every maximal path in T has the same number of branching nodes (i.e., nodes with exactly two children). A *left initial segment of a full binary tree* is a tree T such that there exists a full binary tree T' and a leaf $v \in T'$ such that $T = \{u \in T' \mid u \preceq v\}$.

2.1 Leaf languages

A nondeterministic Turing-machine (NTM) M is *adequate*, if (i) for every input $w \in \Sigma^*$, M does not have an infinite computation on input w and (ii) the set of finitely many transition tuples of M is linearly ordered. For an input w for M, we define the computation tree by unfolding the configuration graph of M from the initial configuration. By condition (i) and (ii), the computation tree can be identified with a finite ordered tree $T(w) \subseteq \mathbb{N}^*$. For $u \in T(w)$ let $q(u)$ be the M-state of the configuration that is associated with the tree node u. Then, the leaf string $\mathrm{leaf}(M, w)$ is the string $\alpha(q(v_1)) \cdots \alpha(q(v_k))$, where $v_1, \ldots, v_k$ are all leafs of $T(w)$ listed in lexicographic order, and $\alpha(q) = 1$ (resp. $\alpha(q) = 0$) if q is an accepting (resp. rejecting) state.

An adequate NTM M is *balanced*, if for every input $w \in \Sigma^*$, $T(w)$ is a left initial segment of a full binary tree. With a language $K \subseteq \{0,1\}^*$ we associate the language $\mathrm{LEAF}(M, K) = \{w \in \Sigma^* \mid \mathrm{leaf}(M, w) \in K\}$ and the following four complexity classes:

$$\begin{aligned}
\mathrm{LEAF}_a^P(K) &= \{\mathrm{LEAF}(M, K) \mid M \text{ is an adequate polynomial time NTM}\} \\
\mathrm{LEAF}_b^P(K) &= \{\mathrm{LEAF}(M, K) \mid M \text{ is a balanced polynomial time NTM}\} \\
\mathrm{LEAF}_a^L(K) &= \{\mathrm{LEAF}(M, K) \mid M \text{ is an adequate logarithmic space NTM}\} \\
\mathrm{LEAF}_b^L(K) &= \{\mathrm{LEAF}(M, K) \mid M \text{ is a balanced logarithmic space NTM}\}
\end{aligned}$$

The first two (resp. last two) classes are closed under polynomial time (resp. logspace) reductions. More details on leaf languages can be found in [7, 13, 14, 16].

2.2 Straight-line programs

Following [23], a *straight-line program (SLP)* over the terminal alphabet Γ is a context-free grammar $\mathbb{A} = (V, \Gamma, S, P)$ (V is the set of variables, Γ is the set of terminals, $S \in V$ is the initial variable, and $P \subseteq V \times (V \cup \Gamma)^*$ is the finite set of productions) such that: (i) for every $A \in V$ there exists exactly one production of the form $(A, \alpha) \in P$ for $\alpha \in (V \cup \Gamma)^*$, and (ii) the relation $\{(A, B) \in V \times V \mid (A, \alpha) \in P, B \text{ occurs in } \alpha\}$ is acyclic. Clearly, the

language generated by the SLP $\mathbb{A}$ consists of exactly one word that is denoted by $\mathrm{val}(\mathbb{A})$. The size of $\mathbb{A}$ is $|\mathbb{A}| = \sum_{(A,\alpha)\in P} |\alpha|$. Every SLP can be transformed in polynomial time into an equivalent SLP in *Chomsky normal form*, i.e. all productions have the form (A, a) with $a \in \Gamma$ or (A, BC) with $B, C \in V$.

As an example, consider the SLP $\mathbb{A}$ (in Chomsky normal form) that consists of the productions $A_1 \to b$, $A_2 \to a$, and $A_i \to A_{i-1}A_{i-2}$ for $3 \le i \le 7$. The start variable is A_7. Then $\mathrm{val}(\mathbb{A}) = abaababaabaab$, which is the 7-th Fibonacci word. We have $|\mathbb{A}| = 12$.

One may also allow exponential expressions of the form A^i for $A \in V$ and $i \in \mathbb{N}$ in right-hand sides of productions. Here the number i is coded binary. Such an expression can be replaced by a sequence of $\lceil \log(i) \rceil$ many ordinary productions.

Let us state some simple algorithmic problems that can be easily solved in polynomial time (but not in deterministic logspace under reasonable complexity theoretic assumptions: problem (a) is #L-complete, problems (b) and (c) are complete for functional P [18]):

(a) Given an SLP $\mathbb{A}$, calculate $|\mathrm{val}(\mathbb{A})|$.
(b) Given an SLP $\mathbb{A}$ and a number $i \in \{1, \ldots, |\mathrm{val}(\mathbb{A})|\}$, calculate $\mathrm{val}(\mathbb{A})[i]$.
(c) Given an SLP $\mathbb{A}$ and two positions $1 \le i \le j \le |\mathrm{val}(\mathbb{A})|$, calculate an SLP for the string $\mathrm{val}(\mathbb{A})[i, j]$.

In [22], Plandowski presented a polynomial time algorithm for testing whether $\mathrm{val}(\mathbb{A}) = \mathrm{val}(\mathbb{B})$ for two given SLPs $\mathbb{A}$ and $\mathbb{B}$. For a language $L \subseteq \Sigma^*$, we denote with $\mathrm{CMP}(L)$ (*compressed membership problem* for L) the following computational problem:

INPUT: An SLP $\mathbb{A}$ over the terminal alphabet Σ

QUESTION: $\mathrm{val}(\mathbb{A}) \in L$?

The following result was shown in [3, 16, 20]:

THEOREM 1. *For every regular language L, $\mathrm{CMP}(L)$ can be decided in polynomial time. Moreover, there exists a fixed regular language L such that $\mathrm{CMP}(L)$ is P-complete.*

In [18], we constructed in logspace from a given superdecreasing sequence $(w_1, \ldots, w_k)$ an SLP $\mathbb{A}$ over $\{0, 1\}$ such that $\mathrm{val}(\mathbb{A}) = S(w_1, \ldots, w_k)$, where $S(w_1, \ldots, w_k)$ is the string-encoding from (1). This construction was used in order to prove P-hardness of the problem (b) above. Let us briefly repeat the construction. For $1 \le i \le k$ let

$$d_i = \begin{cases} w_k - 1 & \text{if } i = k \\ w_i - (w_{i+1} + \cdots + w_k) - 1 & \text{if } 1 \le i \le k-1 \end{cases} \tag{2}$$

Moreover define strings $S_1, \ldots, S_k \in \{0, 1\}^*$ by the recursion

$$S_k = 10^{d_k}1 \qquad S_i = S_{i+1}0^{d_i}S_{i+1}\ (1 \le i \le k-1). \tag{3}$$

Then $S(w_1, \ldots, w_k) = S_1$. Note that the SLP that implements the recursion (3) can be constructed in logspace from the binary encoded sequence $(w_1, \ldots, w_k)$ (in [18] only the existence of an NC-construction is claimed). The only nontrivial step is the calculation of all suffix sums $w_{i+1} + \cdots + w_k$ for $1 \le i \le k-1$ in (2), see e.g. [26].

3 Straight-line programs versus leaf languages

In [6, 24], it was shown that the membership problem for a language $K \subseteq \{0,1\}^*$ is complete (w.r.t. polynomial time reductions in [6] and projection reductions in [24]) for the leaf language class $\mathrm{LEAF}_b^P(K)$, if the input string is represented by a Boolean circuit. For SLP-compressed strings, we obtain a similar result:

PROPOSITION 2. *For every language $K \subseteq \{0,1\}^*$, the problem $\mathrm{CMP}(K)$ is complete w.r.t. logspace reductions for the class $\mathrm{LEAF}_a^L(K)$.*

The proposition can be easily shown by translating configuration graphs of logspace machines into SLPs and vice versa. We now prove a more subtle relationship between SLP-compressed strings and polynomial time leaf languages. Let $\rho : (\{0,1\} \times \{0,1\})^* \to \{0,1\}^*$ be the morphism defined by

$$\rho(0,0) = \rho(0,1) = \varepsilon, \quad \rho(1,0) = 0, \quad \rho(1,1) = 1. \tag{4}$$

THEOREM 3. *Let M be a balanced polynomial time NTM. From a given input $w \in \Sigma^*$ for M we can construct in polynomial time two SLPs $\mathbb{A}$ and $\mathbb{B}$ such that $|\mathrm{val}(\mathbb{A})| = |\mathrm{val}(\mathbb{B})|$ and $\mathrm{leaf}(M,w) = \rho(\mathrm{val}(\mathbb{A}) \otimes \mathrm{val}(\mathbb{B}))$.*

PROOF. Let w be an input for M. Our construction consists of five steps:

Step 1. By simulating M e.g. along the right-most computation path, we can compute in polynomial time the number m of branching nodes along every maximal path in the computation tree $T(w)$. Thus, maximal paths in $T(w)$ can be represented by strings from $\{0,1\}^m$.

Step 2. Using the classical Cook-Levin construction, we compute in logspace a Boolean circuit $C_w(x_1,\ldots,x_m)$ from w such that for all $a_1,\ldots,a_m \in \{0,1\}$: $C_w(a_1,\ldots,a_m)$ evaluates to true if and only if the machine M accepts on the computation path that is specified by the bit string $a_1 \cdots a_m$. The circuit $C_w(x_1,\ldots,x_m)$ has input gates $x_1,\ldots,x_m$ together with some additional input gates that carry fixed input bits.

Step 3. The construction from [17] (see Sec. 2) allows us to compute from $C_w(x_1,\ldots,x_m)$ in logspace a superdecreasing subsetsum instance $(t(x_1,\ldots,x_m), w_1,\ldots,w_k)$ with $w_1,\ldots,w_k \in \mathbb{N}$ and $t(x_1,\ldots,x_m) = t_0 + x_1t_1 + \cdots + x_mt_m$ such that

- $t_1 > t_2 > \cdots > t_m$ and the sequence $(t_1,\ldots,t_m)$ is superdecreasing,
- for all $a_1,\ldots,a_m \in \{0,1\}$: $C_w(a_1,\ldots,a_m)$ evaluates to true if and only if $\exists b_1,\ldots,b_k \in \{0,1\} : t_0 + a_1t_1 + \cdots + a_mt_m = b_1w_1 + \cdots + b_kw_k$,
- $t_0 + t_1 + \cdots + t_m \leq w_1 + \cdots + w_k$.

Step 4. By [18] (see the end of Sec. 2.2), we can construct in logspace from the two superdecreasing sequences $(t_1,\ldots,t_m), (w_1,\ldots,w_k)$ SLPs $\mathbb{A}'$ and $\mathbb{B}$ over $\{0,1\}$ such that $\mathrm{val}(\mathbb{A}') = S(t_1,\ldots,t_m)$ and $\mathrm{val}(\mathbb{B}) = S(w_1,\ldots,w_k)$ (see (1)). Note that $|\mathrm{val}(\mathbb{A}')| = t_1 + \cdots + t_m + 1 \leq w_1 + \cdots + w_k + 1 = |\mathrm{val}(\mathbb{B})|$.

Step 5. Now, we compute in polynomial time the right-most path of the computation tree $T(w)$. Assume that this path is represented by the bit string $r = r_1 \cdots r_m \in \{0,1\}^m$. Let $p = r_1t_1 + \cdots + r_mt_m$. Thus, if r is the lexicographically n-th string in $\{0,1\}^m$, then $p+1$ is the position of the n-th 1 in $\mathrm{val}(\mathbb{A}')$. From the SLP $\mathbb{A}'$ we can finally compute in polynomial

time an SLP $\mathbb{A}$ with $\mathrm{val}(\mathbb{A}) = 0^{t_0}\ S(t_1, \ldots, t_m)[1, p+1]\ 0^{w_1+\cdots+w_k-t_0-p}$. Then $|\mathrm{val}(\mathbb{A})| = |\mathrm{val}(\mathbb{B})|$ and for all positions $q \in \{0, \ldots, |\mathrm{val}(\mathbb{A})| - 1\}$:

- $\mathrm{val}(\mathbb{A})[q+1] = 1$ if and only if $\exists a_1, \ldots, a_m \in \{0,1\} : q = t_0 + a_1 t_1 + \cdots + a_m t_m$
- $\mathrm{val}(\mathbb{B})[q+1] = 1$ if and only if $\exists b_1, \ldots, b_k \in \{0,1\} : q = b_1 w_1 + \cdots + b_k w_k$.

Due to the definition of the projection ρ in (4), we finally have

$$\rho(\mathrm{val}(\mathbb{A}) \otimes \mathrm{val}(\mathbb{B})) = \prod_{x \in \{0,1\}^m,\, x \preceq r} \alpha(x),$$

where $\alpha(x) \in \{0,1\}$ and $\alpha(x_1 \cdots x_m) = 1$ if and only if there exist $b_1, \ldots, b_k \in \{0,1\}$ such that $t_0 + x_1 t_1 + \cdots x_m t_m = b_1 w_1 + \cdots + b_k w_k$. Hence, $\alpha(x_1 \cdots x_m) = 1$ if and only if M accepts on the computation path specified by $x_1 \cdots x_m \preceq r$. Thus, $\mathrm{leaf}(M, w) = \rho(\mathrm{val}(\mathbb{A}) \otimes \mathrm{val}(\mathbb{B}))$. ∎

Thm. 3 implies the hardness part in the following corollary. The proof of the upper bound is not difficult and left to the reader.

COROLLARY 4. *For every language $K \subseteq \{0,1\}^*$, the following problem is complete for the class $\mathrm{LEAF}_b^p(K)$ w.r.t. polynomial time reductions:*

INPUT: Two SLPs $\mathbb{A}$ and $\mathbb{B}$ over $\{0,1\}$
QUESTION: $\rho(\mathrm{val}(\mathbb{A}) \otimes \mathrm{val}(\mathbb{B})) \in K$?

In order to get completeness results w.r.t. logspace reductions in the next section, we need a variant of Thm. 3. We say that an NTM is *fully balanced*, if for every input w, $T(w)$ is a full binary tree (and not just a left initial segment of a full binary tree).

THEOREM 5. *Let M be a fully balanced polynomial time NTM such that for some polynomial $p(n)$, every maximal path in a computation tree $T(w)$ has exactly $p(|w|)$ many branching nodes. From a given input $w \in \Sigma^*$ for M we can construct in logspace two SLPs $\mathbb{A}$ and $\mathbb{B}$ such that $\mathrm{leaf}(M, w) = \rho(\mathrm{val}(\mathbb{A}) \otimes \mathrm{val}(\mathbb{B}))$ and $|\mathrm{val}(\mathbb{A})| = |\mathrm{val}(\mathbb{B})|$.*

PROOF. Only step 1 and 5 in the proof of Thm. 3 cannot be done in logspace, unless L = P. Under the additional assumptions of Thm. 5, we have to compute in step 1 only $m = p(|w|)$, which is possible in logspace, since $p(n)$ is a fixed polynomial. In step 5, we just have to compute in logspace an SLP $\mathbb{A}$ with $\mathrm{val}(\mathbb{A}) = 0^{t_0}\ S(t_1, \ldots, t_m)\ 0^{w_1+\cdots+w_k-(t_0+\cdots+t_m)}$. ∎

4 Applications

COROLLARY 6. *There exists a fixed regular language $L \subseteq (\{0,1\} \times \{0,1\})^*$ such that the following problem is PSPACE-complete w.r.t. logspace reductions:*

INPUT: Two SLPs $\mathbb{A}$ and $\mathbb{B}$ over $\{0,1\}$
QUESTION: $\mathrm{val}(\mathbb{A}) \otimes \mathrm{val}(\mathbb{B}) \in L$?

PROOF. Membership in PSPACE is obvious. Let us prove the lower bound. By [14], there exists a regular language $K \subseteq \{0,1\}^*$ and a balanced polynomial time NTM M such that the language $\mathrm{LEAF}(M, K)$ is PSPACE-complete. Using the padding technique from [16, Prop. 2.3], we can even assume that M is fully balanced and that the number of branching nodes along every maximal path of $T(w)$ is exactly $p(|w|)$ for a polynomial $p(n)$. Let $L = \rho^{-1}(K)$, which is a fixed regular language, since ρ from (4) is a fixed morphism. Let w

be an input for M. By Thm. 5, we can construct in logspace two SLPs $\mathbb{A}$ and $\mathbb{B}$ such that $\rho(\mathrm{val}(\mathbb{A}) \otimes \mathrm{val}(\mathbb{B})) = \mathrm{leaf}(M, w)$. Hence, the corollary follows from $w \in \mathrm{LEAF}(M, K) \iff \mathrm{leaf}(M, w) = \rho(\mathrm{val}(\mathbb{A}) \otimes \mathrm{val}(\mathbb{B})) \in K \iff \mathrm{val}(\mathbb{A}) \otimes \mathrm{val}(\mathbb{B}) \in L$. ∎

From Thm. 5 it follows that that even the set of all SLP-pairs $\langle \mathbb{A}, \mathbb{B} \rangle$ with $\mathrm{val}(\mathbb{A}) \otimes \mathrm{val}(\mathbb{B}) \in L$ and $|\mathrm{val}(\mathbb{A})| = |\mathrm{val}(\mathbb{B})|$ (or $|\mathrm{val}(\mathbb{A})| \leq |\mathrm{val}(\mathbb{B})|$) is PSPACE-complete w.r.t. logspace reductions. We need this detail in the proof of the next theorem.

In [19] we constructed a linear deterministic context-free language with a PSPACE-complete compressed membership problem. As noted in the introduction, this result follows also from PSPACE $= \mathrm{LEAF}^L_a(K)$ for a linear deterministic context-free language K [8] together with Prop. 2. We now sharpen this result to linear visibly pushdown languages.

Let Σ_c and Σ_r be two disjoint finite alphabets (call symbols and return symbols) and let $\Sigma = \Sigma_c \cup \Sigma_r$. A *visibly pushdown automaton* (VPA) [1] over (Σ_c, Σ_r) is a tuple $V = (Q, q_0, \Gamma, \bot, \Delta, F)$, where Q is a finite set of states, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is the set of final states, Γ is the finite set of stack symbols, $\bot \in \Gamma$ is the initial stack symbol, and

$$\Delta \subseteq (Q \times \Sigma_c \times Q \times (\Gamma \setminus \{\bot\})) \cup (Q \times \Sigma_r \times \Gamma \times Q)$$

is the set of transitions. In [1], the input alphabet may also contain internal symbols, on which the automaton does not touch the stack at all. For our lower bound, we will not need internal symbols. A configuration of V is a triple from $Q \times \Sigma^* \times \Gamma^*$. For two configurations (p, au, v) and (q, u, w) (with $a \in \Sigma, u \in \Sigma^*$) we write $(p, au, v) \Rightarrow_V (q, u, w)$ if

- $a \in \Sigma_c$ and $w = \gamma v$ for some $\gamma \in \Gamma$ with $(p, a, q, \gamma) \in \Delta$, or
- $a \in \Sigma_r$ and $v = \gamma w$ for some $\gamma \in \Gamma$ with $(p, a, \gamma, q) \in \Delta$, or
- $a \in \Sigma_r$, $u = v = \bot$, and $(p, a, \bot, q) \in \Delta$.

The language $L(V)$ is defined as $L(V) = \{w \in \Sigma^* \mid \exists f \in F, u \in \Gamma^* : (q_0, w, \bot) \Rightarrow^*_V (f, \varepsilon, u)\}$. The VPA V is deterministic if for every $p \in Q$ and $a \in \Sigma$ the following hold:

- If $a \in \Sigma_c$, then there is at most one pair $(q, \gamma) \in Q \times \Gamma$ with $(p, a, q, \gamma) \in \Delta$.
- If $a \in \Sigma_r$, then for every $\gamma \in \Gamma$ there is at most one $q \in Q$ with $(p, a, \gamma, q) \in \Delta$.

For every VPA V there exists a deterministic VPA V' with $L(V) = L(V')$ [1]. A *1-turn* VPA is a VPA V with $L(V) \subseteq \Sigma_c^* \Sigma_r^*$. In this case $L(V)$ is called a *linear visibly pushdown language*.

By a classical result from [11], there exists a context-free language with a LOGCFL-complete membership problem. For visibly pushdown languages the complexity of the membership problem decreases to the circuit complexity class NC^1 [9] and is therefore of the same complexity as for regular languages [2]. In contrast to this, by the following theorem, compressed membership is in general PSPACE-complete even for linear visibly pushdown languages, whereas it is P-complete for regular languages (Thm. 1):

THEOREM 7. *There exists a linear visibly pushdown language K such that $\mathrm{CMP}(K)$ is PSPACE-complete w.r.t. logspace reductions.*

PROOF. Membership in PSPACE holds even for an arbitrary context-free language K [23]. For the lower bound, we reduce the problem from Cor. 6 to $\mathrm{CMP}(K)$ for some linear visibly pushdown language K. Let $L \subseteq (\{0,1\} \times \{0,1\})^*$ be the regular language from Cor. 6 and let $A = (Q, \{0,1\} \times \{0,1\}, \delta, q_0, F)$ be a deterministic finite automaton with $L(A) = L$. W.l.o.g. assume that the initial state q_0 has no incoming transitions.

From two given SLPs $\mathbb{A}$ and $\mathbb{B}$ over $\{0,1\}$ we can easily construct in logspace an SLP $\mathbb{C}$ over $\Sigma = \{0,1,\overline{0},\overline{1}\}$ with $\mathrm{val}(\mathbb{C}) = \overline{\mathrm{val}(\mathbb{B})}\,\mathrm{val}(\mathbb{A})$. Let $V = (Q, q_0, \{\bot, 0, 1\}, \bot, \Delta, F)$ be the 1-turn VPA over $(\{\overline{0},\overline{1}\}, \{0,1\})$ with the following transitions:

$$\Delta = \{(q_0, \overline{x}, q_0, x) \mid x \in \{0,1\}\} \cup \{(q, x, y, p) \mid x, y \in \{0,1\}, \delta(q,(x,y)) = p\}.$$

Thus, V can only read words of the form $\overline{v}u$ with $u, v \in \{0,1\}^*$ and $|v| \geq |u|$ (recall that q_0 has no incoming transitions). When reading such a word $\overline{v}u$, V first pushes the word v (reversed) on the stack and then simulates the automaton A on the string $u \otimes v$ and thereby pops from the stack. From the construction of V, we obtain

$$\mathrm{val}(\mathbb{C}) = \overline{\mathrm{val}(\mathbb{B})}\,\mathrm{val}(\mathbb{A}) \in L(V) \quad \Longleftrightarrow \quad \mathrm{val}(\mathbb{A}) \otimes \mathrm{val}(\mathbb{B}) \in L(A) \wedge |\mathrm{val}(\mathbb{A})| \leq |\mathrm{val}(\mathbb{B})|.$$

By Cor. 6 (and the remark after the proof), this concludes the proof. ∎

Prop. 2 and Thm. 7 imply:

COROLLARY 8. $\mathrm{PSPACE} = \mathrm{LEAF}_a^L(K)$ *for some linear visibly pushdown language* K.

In [21], a suitable variant of nondeterministic finite automata were used as leaf string generating devices. A *finite leaf automaton* (FLA) is a tuple $A = (Q, \Sigma, \Gamma, \delta, \rho, q_0)$, where Q is a finite set of states, Σ and Γ are finite alphabets, $\delta : Q \times \Sigma \to Q^+$ is the transition mapping, $\rho : Q \to \Gamma$ is the output mapping, and $q_0 \in Q$ is the initial state. For every state $q \in Q$ and every input word $w \in \Sigma^*$, we define by induction the string $\widehat{\delta}(q,w)$ as follows: $\widehat{\delta}(q,\varepsilon) = q$ and $\widehat{\delta}(q,au) = \widehat{\delta}(q_1,u)\cdots\widehat{\delta}(q_n,u)$ if $a \in \Sigma$ and $\delta(q,a) = q_1 \cdots q_n$. Let $\mathrm{leaf}(A,w) = \rho(\widehat{\delta}(q_0,w))$, where $\rho : Q \to \Gamma$ is extended to a morphism on Q^*. For $K \subseteq \Gamma^*$ let $\mathrm{LEAF}(A,K) = \{w \in \Sigma^* \mid \mathrm{leaf}(A,w) \in K\}$ and $\mathrm{LEAF}(K) = \{\mathrm{LEAF}(A,K) \mid A \text{ is an FLA }\}$.

THEOREM 9. *There exists a fixed linear visibly pushdown language* K *and an FLA* A *such that* $\mathrm{LEAF}(A,K)$ *is PSPACE-complete w.r.t. logspace reductions.*

PROOF. We use the linear visibly pushdown language K from the proof of Thm. 7. Notice that the question whether $\mathrm{val}(\mathbb{C}) \in K$ is already PSPACE-complete for a quite restricted class of SLPs. By tracing the construction of the SLP $\mathbb{C}$ (starting from the proof of Thm. 5), we see that it is already PSPACE-complete to check for a number t_0 and two superdecreasing sequences $(t_1,\ldots,t_m)$, $(w_1,\ldots,w_k)$ (all numbers are encoded binary) whether

$$\overline{S(w_1,\ldots,w_k)}\;0^{t_0}\;S(t_1,\ldots,t_m)\;0^{w_1+\cdots+w_k-(t_0+\cdots+t_m)} \;\in\; K. \tag{5}$$

Here we use again the encoding of superdecreasing sequences from (1). So, it remains to find an FLA A with the following property: from given input data $t_0, (t_1,\ldots,t_m), (w_1,\ldots,w_k)$ as above we can construct in logspace a string w such that $\mathrm{leaf}(A,w)$ is exactly the string in (5). We only present an FLA A and a logspace construction of a string w from a superdecreasing sequence $(w_1,\ldots,w_k)$ such that $\mathrm{leaf}(A,w) = S(w_1,\ldots,w_k)$. From this FLA, an FLA for producing the leaf string (5) can be easily derived. We use the following logspace-computable exponent-encoding of a natural number $d = 2^{e_1} + 2^{e_2} + \cdots + 2^{e_m}$ $(e_1 < e_2 < \cdots < e_m)$:

$$e(d) = a^{e_1}\$a^{e_2}\$\cdots a^{e_{m-1}}\$a^{e_m}\widetilde{\$} \in \{a,\$\}^*\widetilde{\$}.$$

Next, we derive in logspace from the superdecreasing sequence $(w_1, \ldots, w_k)$ the sequence $(d_1, \ldots, d_k)$ of differences as defined in (2) and encode it by the string

$$e(d_1, \ldots, d_k) = \left(\prod_{i=1}^{k-1} \#e(d_i)\right)\widetilde{\#}e(d_k) \in \{a, \$, \widetilde{\$}, \#, \widetilde{\#}\}^*$$

Our fixed FLA is $A = (\{q_0, p_r, p_\ell, r_0, r_1\}, \{a, \$, \widetilde{\$}, \#, \widetilde{\#}\}, \{0,1\}, \delta, \rho, q_0)$, where the transition function δ is defined as follows:

$$\begin{array}{lll}
\delta(q_0, \#) = q_0 p_r q_0 & \delta(p_r, a) = p_\ell p_r & \delta(p_\ell, a) = p_\ell p_\ell \\
\delta(q_0, x) = q_0 \text{ for } x \in \{a, \$, \widetilde{\$}\} & \delta(p_r, \$) = r_0 p_r & \delta(p_\ell, x) = r_0 \text{ for } x \in \{\$, \widetilde{\$}\} \\
\delta(q_0, \widetilde{\#}) = r_1 p_r r_1 & \delta(p_r, \widetilde{\$}) = r_0 & \delta(r_i, x) = r_i \text{ for } x \in \Sigma, i \in \{0,1\}
\end{array}$$

The δ-values that are not explicitly defined can be set arbitrarily. Finally, let $\rho(r_0) = 0$ and $\rho(r_1) = 1$; all other ρ-values can be defined arbitrarily. We claim that $\text{leaf}(A, e(d_1, \ldots, d_k)) = S(w_1, \ldots, w_k)$. First note that $\widehat{\delta}(p_r, a^e\$) = r_0^{2^e} p_r$ and $\widehat{\delta}(p_r, a^e\widetilde{\$}) = r_0^{2^e}$. Since $\delta(r_0, x) = r_0$ for all input symbols x, we have $\widehat{\delta}(p_r, e(d)) = r_0^d$ for every number d and therefore:

$$\begin{aligned}
\widehat{\delta}(q_0, \#e(d)) &= \widehat{\delta}(q_0, e(d))\, \widehat{\delta}(p_r, e(d))\, \widehat{\delta}(q_0, e(d)) = q_0 r_0^d q_0 \\
\widehat{\delta}(q_0, \widetilde{\#}e(d)) &= \widehat{\delta}(r_1, e(d))\, \widehat{\delta}(p_r, e(d))\, \widehat{\delta}(r_1, e(d)) = r_1 r_0^d r_1
\end{aligned}$$

Hence, the FLA A realizes the recurrence (3) when reading the input $e(d_1, \ldots, d_k)$. ∎

5 Compressed membership in XML languages

In this section, we consider a subclass of the visibly pushdown languages, which is motivated in connection with XML. Let B be a finite set of opening brackets and let $\overline{B}$ be the set of corresponding closing brackets. An *XML-grammar* [4] is a tuple $G = (B, (R_b)_{b \in B}, a)$ where $a \in B$ (the axiom) and R_b is a regular language over the alphabet $\{X_c \mid c \in B\}$. We identify G with the context-free grammar, where (i) $\{X_b \mid b \in B\}$ is the set of variables, (ii) $B \cup \overline{B}$ is the set of terminals, (iii) X_a is the start variable, and (iv) the (infinite) set of productions is $\{X_b \to b\, w\, \overline{b} \mid b \in B, w \in R_b\}$. Since R_b is regular, this set is equivalent to a finite set of productions. One can show that $L(G)$ is a visibly pushdown language [1]. XML-grammars capture the syntactic features of XML document type definitions (DTDs), see [4] for details.

THEOREM 10. *For every XML-grammar G,* $\text{CMP}(L(G))$ *belongs to coNP. Moreover, there is an XML-grammar G such that* $\text{CMP}(L(G))$ *is coNP-complete w.r.t. logspace reductions.*

For the proof of the upper bound in Thm. 10 we need a few definitions. Let us fix an XML-grammar $G = (B, (R_b)_{b \in B}, a)$ for the further considerations. The set $D_B \subseteq (B \cup \overline{B})^+$ of all *Dyck primes* over B is the set of all well-formed strings over $B \cup \overline{B}$ that do not have a non-empty proper prefix, which is well-formed as well. Formally, D_B is the smallest set such that $w_1, \ldots, w_n \in D_B$ ($n \geq 0$) implies $b w_1 \cdots w_n \overline{b} \in D_B$. For $b \in B$ let $D_b = D_B \cap b(B \cup \overline{B})^* \overline{b}$. The set of all *Dyck words* over $B \cup \overline{B}$ is D_B^*. Note that $L(G) \subseteq D_a$.

Let $w \in D_B^*$, and let $1 \leq i \leq |w|$ be a position with $w[i] \in B$, i.e. the i-th symbol in w is an opening bracket. Since $w \in D_B^*$, there exists a unique position $\gamma(w, i) > i$ with $w[i, \gamma(w, i)] \in$

D_B. The string $w[i+1, \gamma(w,i)-1]$ belongs to D_B^*. Since D_B is a code, there exists a unique factorization $w[i+1, \gamma(w,i)-1] = w_1 \cdots w_n$ with $n \geq 0$ and $w_1, \ldots, w_n \in D_B$. Moreover, for every $1 \leq i \leq n$ let b_i be the unique opening bracket such that $w_i \in D_{b_i}$. Finally, define $\text{surface}(w,i) = X_{b_1} X_{b_2} \cdots X_{b_n}$. The term "surface" is motivated by the surface of $b \in B$ from [4]. A straightforward induction shows:

LEMMA 11. *Let $w \in (B \cup \overline{B})^*$. Then $w \in L(G)$ if and only if (i) $w \in D_a$ and (ii) surface$(w,j) \in R_b$ for every position $1 \leq j \leq |w|$ such that $w[j] = b \in B$.*

The next lemma was shown in [19, Lemma 5.6]:

LEMMA 12. $\text{CMP}(D_B^*)$ *can be solved in polynomial time. Moreover, for a given SLP $\mathbb{A}$ such that $w := \text{val}(\mathbb{A}) \in D_B^*$ and a given (binary coded) position $1 \leq i \leq |w|$ with $w[i] \in B$ one can compute the position $\gamma(w,i)$ in polynomial time.*

Lemma 12 and the fact $w \in D_B \iff (w \in D_B^* \text{ and } \gamma(w,1) = |w|)$ implies:

PROPOSITION 13. $\text{CMP}(D_B)$ *can be solved in polynomial time.*

For the proof of Thm. 10 we need one more technical lemma, whose proof has to be omitted in this short version:

LEMMA 14. *For a given SLP $\mathbb{A}$ such that $w := \text{val}(\mathbb{A}) \in D_B^*$ and a given (binary coded) position $1 \leq i \leq |w|$ with $w[i] \in B$ one can compute an SLP for the string surface(w,i) in polynomial time.*

Now we can prove Thm. 10: For the coNP upper bound, let $G = (B, (R_b)_{b \in B}, a)$ be an XML grammar and let $\mathbb{A}$ be an SLP over the terminal alphabet $B \cup \overline{B}$ with $w = \text{val}(\mathbb{A})$. By Lemma 11 we have to check that (i) $w \in D_a = D_B \cap a(B \cup \overline{B})^* \overline{a}$ and (ii) $\text{surface}(w,j) \in R_b$ for all $1 \leq j \leq |w|$ with $w[j] = b \in B$. Condition (i) can be checked in deterministic polynomial time by Prop. 13; condition (ii) belongs to coNP by Lemma 14 and Thm. 1. The proof of the coNP lower bound is similar to the proof of [19, Thm. 5.2] and therefore omitted. ∎

References

[1] R. Alur and P. Madhusudan. Visibly pushdown languages. In *Proc. STOC 2004*, 202–211. ACM Press, 2004.

[2] D. A. M. Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1. *J. Comput. System Sci.*, 38:150–164, 1989.

[3] M. Beaudry, P. McKenzie, P. Péladeau, and D. Thérien. Finite monoids: From word to circuit evaluation. *SIAM J. Comput.*, 26(1):138–152, 1997.

[4] J. Berstel and L. Boasson. Formal properties of XML grammars and languages. *Acta Inform.*, 38(9):649–671, 2002.

[5] A. Bertoni, C. Choffrut, and R. Radicioni. Literal shuffle of compressed words. In *Proc. IFIP TCS 2008*, 87–100. Springer, 2008.

[6] B. Borchert and A. Lozano. Succinct circuit representations and leaf language classes are basically the same concept. *Inform. Process. Lett.*, 59(4):211–215, 1996.

[7] D. P. Bovet, P. Crescenzi, and R. Silvestri. A uniform approach to define complexity classes. *Theoret. Comput. Sci.*, 104(2):263–283, 1992.

[8] H. Caussinus, P. McKenzie, D. Thérien, and H. Vollmer. Nondeterministic NC^1 computation. *J. Comput. System Sci.*, 57(2):200–212, 1998.

[9] P. W. Dymond. Input-driven languages are in $\log n$ depth. *Inform. Process. Lett.*, 26(5):247–250, 1988.

[10] L. Gasieniec, M. Karpinski, W. Plandowski, and W. Rytter. Efficient algorithms for Lempel-Ziv encoding. In *Proc. SWAT 1996*, LNCS 1097, 392–403. Springer, 1996.

[11] S. Greibach. The hardest context-free language. *SIAM J. Comput.*, 2(4):304–310, 1973.

[12] C. Hagenah. *Gleichungen mit regulären Randbedingungen über freien Gruppen*. PhD thesis, University of Stuttgart, Institut für Informatik, 2000.

[13] U. Hertrampf. The shapes of trees. In *Proc. COCOON 1997*, LNCS 1276, 412–421. Springer, 1997.

[14] U. Hertrampf, C. Lautemann, T. Schwentick, H. Vollmer, and K. W. Wagner. On the power of polynomial time bit-reductions. In *Proc. Eighth Annual Structure in Complexity Theory Conference*, 200–207. IEEE Computer Society Press, 1993.

[15] M. Holzer and K.-J. Lange. On the complexities of linear LL(1) and LR(1) grammars. In *Proc. FCT 1993*, LNCS 710, 299–308. Springer, 1993.

[16] B. Jenner, P. McKenzie, and D. Thérien. Logspace and logtime leaf languages. *Inform. and Comput.*, 129(1):21–33, 1996.

[17] H. J. Karloff and W. L. Ruzzo. The iterated mod problem. *Inform. and Comput.*, 80(3):193–204, 1989.

[18] Y. Lifshits and M. Lohrey. Querying and embedding compressed texts. In *Proc. MFCS 2006*, LNCS 4162, 681–692. Springer, 2006.

[19] M. Lohrey. Word problems and membership problems on compressed words. *SIAM J. Comput.*, 35(5):1210 – 1240, 2006.

[20] N. Markey and P. Schnoebelen. A PTIME-complete matching problem for SLP-compressed words. *Inform. Process. Lett.*, 90(1):3–6, 2004.

[21] T. Peichl and H. Vollmer. Finite automata with generalized acceptance criteria. *Discrete Math. Theor. Comput. Sci.*, 4(2):179–192 (electronic), 2001.

[22] W. Plandowski. Testing equivalence of morphisms on context-free languages. In *Proc. ESA'94*, LNCS 855, 460–470. Springer, 1994.

[23] W. Plandowski and W. Rytter. Complexity of language recognition problems for compressed words. In J. Karhumäki, H. A. Maurer, G. Paun, and G. Rozenberg, editors, *Jewels are Forever, Contributions on Theoretical Computer Science in Honor of Arto Salomaa*, 262–272. Springer, 1999.

[24] H. Veith. Succinct representation, leaf languages, and projection reductions. *Inform. and Comput.*, 142(2):207–236, 1998.

[25] N. K. Vereshchagin. Relativizable and nonrelativizable theorems in the polynomial theory of algorithms. *Izv. Ross. Akad. Nauk Ser. Mat.*, 57(2):51–90, 1993.

[26] H. Vollmer. *Introduction to Circuit Complexity*. Springer, 1999.

[27] B. von Braunmühl and R. Verbeek. Input-driven languages are recognized in $\log n$ space. In *Proc. FCT 1983*, LNCS 158, 40–51. Springer, 1983.

Foundations of Software Technology and Theoretical Computer Science (Bangalore) 2008.
Editors: R. Hariharan, M. Mukund, V. Vinay; pp 304-315

Complexity Analysis of Term Rewriting Based on Matrix and Context Dependent Interpretations*

Georg Moser[1], **Andreas Schnabl**[1] **and Johannes Waldmann**[2]

[1] Institut für Informatik, Universität Innsbruck, Austria;
{georg.moser,andreas.schnabl}@uibk.ac.at

[2] Fachbereich Informatik, Mathematik und Naturwissenschaften, Hochschule für Technik, Wirtschaft und Kultur (FH) Leipzig, Germany; waldmann@imn.htwk-leipzig.de

ABSTRACT. For a given (terminating) term rewriting system one can often estimate its *derivational complexity* indirectly by looking at the proof method that established termination. In this spirit we investigate two instances of the interpretation method: *matrix interpretations* and *context dependent interpretations*. We introduce a subclass of matrix interpretations, denoted as *triangular matrix interpretations*, which induce polynomial derivational complexity and establish tight correspondence results between a subclass of context dependent interpretations and restricted triangular matrix interpretations. The thus obtained new results are easy to implement and considerably extend the analytic power of existing results. We provide ample numerical data for assessing the viability of the method.

1 Introduction

Term rewriting is a conceptually simple but Turing-complete model of computation. The foundation of rewriting is equational logic and term rewrite systems are conceivable as sets of directed equations. This orientation of equations naturally gives rise to computations, where a term is rewritten by successively replacing subterms by equal terms until no further reduction is possible. Such a sequence of rewrite steps is also called a *derivation*. In order to assess the complexity of a (terminating) term rewrite system (TRS for short) it is natural to look at the maximal length of derivations, as suggested by Hofbauer and Lautemann in [10]. More precisely, the *derivational complexity function* with respect to a (terminating) TRS $\mathcal{R}$ relates the length of a longest derivation sequence to the size of the initial term. Observe that the derivational complexity function is conceivable as a measure of proof complexity. Suppose an equational theory is representable as a convergent (i.e. a confluent and terminating) TRS, then rewriting to normal form induces an effective procedure to decide whether two terms are equal over a given equational theory. Thus the derivational complexity with respect to a convergent TRS essentially amounts to the *proof complexity* of this proof of identity.

For a given terminating TRS one can often estimate its derivational complexity indirectly by looking at the proof method that established termination. For example *polynomial*

*This research is partly supported by FWF (Austrian Science Fund) project P20133.

FSTTCS 2008
IARCS Annual Conference on
Foundations of Software Technology and Theoretical Computer Science
http://drops.dagstuhl.de/opus/volltexte/2008/1762

interpretations induce double-exponential derivational complexity (see [10], but also compare [8, 19, 9, 14, 7, 5, 12] for the derivational complexity induced by other termination techniques). The following example illustrates the situation.

Example 1 ([9]). Consider the following TRS $\mathcal{R}$ over the signature $\mathcal{F} = \{\circ, \mathsf{c}\}$.

$$(x \circ y) \circ z \to x \circ (y \circ z) \ .$$

It is easy to see that the polynomial interpretation $\mathcal{A}$ on the carrier $\mathbb{N} - \{0\}$ given through the interpretation functions $\circ_{\mathcal{A}}(n, m) = 2n + m$ and $\mathsf{c}_{\mathcal{A}} = 1$, is compatible with $\mathcal{R}$. Now, consider the (ground) terms $(t_n)_{n\in\mathbb{N}}$, defined as $t_0 := \mathsf{c}$ and $t_{n+1} = t_n \circ \mathsf{c}$. Note that the evaluation $[t_n]_{\mathcal{A}}$ of t_n with respect to the algebra $\mathcal{A}$ is exponential in n. Hence the maximal length of a derivation starting from t_n is (at most) exponential in n.

However the upper bound given in Example 1 is not optimal: The derivation length can be easily seen to be bounded quadratically in n. This overestimation is typical for polynomial interpretations. Hofbauer introduced *context dependent interpretations* as a remedy, cf. [9]. These interpretations extend traditional interpretations by introducing an additional parameter. The parameter changes in the course of evaluating a term, which makes the interpretation dependent on the context. With respect to Example 1 an interpretation can be found that estimates the derivation length optimally, compare [9]. Recently the first and second author introduced a technique to *automatically* search for context dependent interpretations, cf. [15]. This was achieved by delineating two subclasses of context dependent interpretations that made automation possible. However, up to now, we couldn't handle the TRS in Example 1 automatically.

In this paper we introduce an (easily automatable) technique to overcome this obstacle. We restrict *matrix interpretations* for terms (see [5], but compare also [11]) in such a way that we only employ matrices of particular simple form in the interpretation functions. Such interpretations (called *triangular matrix interpretations*) induce at most polynomial derivational complexity, where the degree of the polynomial depends on the dimension of the matrix.

Moreover, we identify a subclass of context dependent interpretations and a subclass of (two-dimensional) matrix interpretations which correspond to each other with respect to orientability: For any context dependent interpretation $\mathcal{C}$ from this class that is compatible with a TRS $\mathcal{R}$ there exists a matrix interpretation $\mathcal{A}$ compatible with $\mathcal{R}$ and vice versa. This theoretical result is interesting in its own right as it links two different termination techniques, which were previously conceived as incomparable.

The obtained new techniques are easy to implement and considerably extend the analytic power of existing results. We provide ample numerical data for assessing the viability of the method. In particular, we want to emphasise that Example 1 can be handled fully automatically and the resulting estimation on the derivational complexity is optimal.

The remainder of this paper is organised as follows. In the next section we recall basic notions. Section 3 introduces triangular matrix interpretations, while in Section 4 we recall context dependent interpretations and state the correspondence result mentioned above. In Section 5 we provide the experimental data for our implementation. Finally in Section 6 we conclude and mention possible future work.

2 Preliminaries

We assume familiarity with term rewriting [2, 18] but briefly review basic concepts and notations. Let $\mathcal{V}$ denote a countably infinite set of variables and $\mathcal{F}$ a signature. The set of terms over $\mathcal{F}$ and $\mathcal{V}$ is denoted by $\mathcal{T}(\mathcal{F},\mathcal{V})$. $\mathcal{V}\mathsf{ar}(t)$ denotes the set of variables occurring in a term t and the *size* $|t|$ of a term is defined as the number of symbols in t, i.e., for example the size of the term $\mathsf{f}(\mathsf{a},x)$ is 3. The *depth* $\mathsf{dp}(t)$ of a term t is defined as follows: (i) $\mathsf{dp}(t) := 0$, if t is a variable or a constant and (ii) $\mathsf{dp}(f(t_1,\ldots,t_n)) := 1 + \max\{\mathsf{dp}(t_i) \mid 1 \leqslant i \leqslant n\}$.

A *term rewrite system* (*TRS* for short) $\mathcal{R}$ over $\mathcal{T}(\mathcal{F},\mathcal{V})$ is a *finite* set of rewrite rules $l \to r$, such that $l \notin \mathcal{V}$ and $\mathcal{V}\mathsf{ar}(l) \supseteq \mathcal{V}\mathsf{ar}(r)$. A relation on $\mathcal{T}(\mathcal{F},\mathcal{V})$ is a *rewrite relation* if it is compatible with $\mathcal{F}$-operations and closed under substitutions. The smallest rewrite relation that contains $\mathcal{R}$ is denoted by $\to_{\mathcal{R}}$. The transitive and reflexive closure of $\to_{\mathcal{R}}$ is denoted by $\to_{\mathcal{R}}^*$. We simply write $\to$ for $\to_{\mathcal{R}}$ if $\mathcal{R}$ is clear from context. A term $s \in \mathcal{T}(\mathcal{F},\mathcal{V})$ is called a *normal form* if there is no $t \in \mathcal{T}(\mathcal{F},\mathcal{V})$ such that $s \to t$.

A TRS is called *confluent* if for all $s,t_1,t_2 \in \mathcal{T}(\mathcal{F},\mathcal{V})$ with $s \to^* t_1$ and $s \to^* t_2$ there exists a term t_3 such that $t_1 \to^* t_3$ and $t_2 \to^* t_3$. We call a TRS *terminating* if no infinite rewrite sequence exists. Let s and t be terms. If exactly n steps are performed to rewrite s to t we write $s \to^n t$. The *derivation length* of a terminating term t with respect to a TRS $\mathcal{R}$ is defined as: $\mathsf{dl}(s,\to_{\mathcal{R}}) = \max\{n \mid \exists t\ s \to_{\mathcal{R}}^n t\}$. The *derivational complexity* (with respect to $\mathcal{R}$) is defined as follows:

$$\mathsf{dc}_{\mathcal{R}}(n) = \max\{\mathsf{dl}(t,\to_{\mathcal{R}}) \mid |t| \leqslant n\}\ .$$

We sometimes say the derivational complexity of $\mathcal{R}$ is *linear*, *quadratic*, or *polynomial* if $\mathsf{dc}_{\mathcal{R}}(n)$ is bounded linearly, quadratically, or polynomially in n, respectively.

A *proper order* is a transitive and irreflexive relation. A proper order $\succ$ is *well-founded* if there is no infinite decreasing sequence $t_1 \succ t_2 \succ t_3 \cdots$. A well-founded proper order that is also a rewrite relation is called a *reduction order*. We say a reduction order $\succ$ and a TRS $\mathcal{R}$ are *compatible* if $\mathcal{R} \subseteq \succ$. It is well-known that a TRS is terminating if and only if there exists a compatible reduction order. An *$\mathcal{F}$-algebra* $\mathcal{A}$ consists of a carrier set A and a collection of interpretations $f_{\mathcal{A}}$ for each function symbol in $\mathcal{F}$. A *well-founded* and *monotone* algebra (*WMA* for short) is a pair $(\mathcal{A},\succ)$, where $\mathcal{A}$ is an algebra and $\succ$ is a well-founded proper order on A such that every $f_{\mathcal{A}}$ is monotone in all arguments. An *assignment* $\alpha\colon \mathcal{V} \to A$ is a function mapping variables to elements in the carrier. Let $[\alpha]_{\mathcal{A}}(\cdot)$ denote the usual evaluation function associated with $\mathcal{A}$. A WMA naturally induces a proper order $\succ_{\mathcal{A}}$ on terms: $s \succ_{\mathcal{A}} t$ if $[\alpha]_{\mathcal{A}}(s) \succ [\alpha]_{\mathcal{A}}(t)$ for all assignments $\alpha\colon \mathcal{V} \to A$.

3 Matrix Interpretations That Induce Polynomial Derivational Complexity

In this section we introduce a specific form of matrix interpretations, called *triangular matrix interpretations*, which induce a polynomial upper bound on the derivational complexity. This contrasts with general matrix interpretations, which yield an exponential upper bound, cf. [5]. Hence the introduced restriction defines a strict subclass of those TRSs that admit a matrix interpretation.

We start by recalling the concept of *matrix interpretations* (see [5] but compare also [11]). Let $\mathcal{F}$ denote a signature. We fix a dimension $d \in \mathbb{N}$ and use the set $\mathbb{N}^d$ as the carrier of an algebra $\mathcal{A}$, together with the following extension of the natural order $>$ on $\mathbb{N}$:

$$(x_1, x_2, \ldots, x_d) > (y_1, y_2, \ldots, y_d) :\Longleftrightarrow x_1 > y_1 \wedge x_2 \geqslant y_2 \wedge \ldots \wedge x_d \geqslant y_d \,.$$

For each n-ary function symbol f, we choose as an interpretation a linear function of the following shape:

$$f_{\mathcal{A}} : (\mathbb{N}^d)^n \to \mathbb{N}^d : (\vec{v}_1, \ldots, \vec{v}_n) \mapsto F_1\vec{v}_1 + \ldots + F_n\vec{v}_n + \vec{f} \,,$$

where $\vec{v}_1, \ldots, \vec{v}_n$ are (column) vectors of variables, $F_1, \ldots, F_n$ are matrices (each of size $d \times d$), and $\vec{f}$ is a vector over $\mathbb{N}$. Moreover, for any i ($1 \leqslant i \leqslant n$) the top left entry $(F_i)_{1,1}$ is positive. It is easy to see that the algebra forms a well-founded monotone algebra.

The following lemma states how compatibility of a matrix interpretation $\mathcal{A}$ with a given rewrite system can be easily verified (compare [5, Lemma 4]).

LEMMA 2. *Let $\mathcal{A}$ be a matrix interpretation and let $\mathcal{R}$ be a TRS. Let $l \to r \in \mathcal{R}$, let k denote the number of variables in l (and r) and let α be an assignment. Then there exist matrices $L_1, \ldots, L_k, R_1, \ldots, R_k$ and vectors $\vec{l}, \vec{r}$ such that $[\alpha]_{\mathcal{A}}(l) = \sum_{i=1}^{k} L_i\vec{x}_i + \vec{l}$ and $[\alpha]_{\mathcal{A}}(r) = \sum_{i=1}^{k} R_i\vec{x}_i + \vec{r}$. Moreover $l >_{\mathcal{A}} r$ if and only if $\vec{l} > \vec{r}$ and $L_i \geqslant R_i$ for all $1 \leqslant i \leqslant k$. (Here $\geqslant$ refers to the point-wise extension of the standard order on natural numbers to matrices.)*

We are now going to restrict the shape of the matrices, in order to obtain better bounds on derivational complexities.

DEFINITION 3. *An* upper triangular matrix *is a matrix M in $\mathbb{N}^{d\times d}$ such that for all $d \geqslant i > j \geqslant 1$, we have $M_{i,j} = 0$, and for all $d \geqslant i \geqslant 1$, we have $M_{i,i} \leqslant 1$.*

We say that a TRS $\mathcal{R}$ admits a *triangular matrix interpretation* (*TMI* for short) if $\mathcal{R}$ is compatible with a matrix interpretation $\mathcal{A}$ and all matrices employed in $\mathcal{A}$ are of upper triangular form.

Example 4 (continued from Example 1). We define a triangular matrix interpretation $\mathcal{A}$, as follows:

$$\circ_{\mathcal{A}}(\vec{x}, \vec{y}) = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \cdot \vec{x} + \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \cdot \vec{y} + \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

It is easy to see that $\mathcal{A}$ is compatible with $\mathcal{R}$, i.e., $[\alpha]_{\mathcal{A}}((x \circ y) \circ z) > [\alpha]_{\mathcal{A}}(x \circ (y \circ z))$ holds for all assignments α.

LEMMA 5. *Let M be an upper triangular matrix in $\mathbb{N}^{d\times d}$ and $n \in \mathbb{N}$. Then all entries of M^n are polynomially bounded in n. More precisely, if $i > j$ then $(M^n)_{i,j} = 0$, otherwise $(M^n)_{i,j} \leqslant (j-i)!(an)^{j-i}$, where $a = \max\{M_{i,j} \mid 1 \leqslant i, j \leqslant d\}$.*

PROOF. The case $i > j$ is easy to see. In the other case, we have $j \geqslant i$. Then the lemma follows by a straightforward induction on $j - i$. ∎

Due to Lemma 5, for any finite subset $M \subseteq \mathbb{N}^{d\times d}$ of upper triangular matrices, there is a polynomial p of degree $d - 1$ such that for each sequence $M_1 \in M, \ldots, M_n \in M$, and

for each i, j, it holds that $(M_1 \cdot \ldots \cdot M_n)_{i,j} \leqslant p(n)$. Such products occur when computing values of matrix interpretations on (ground) terms: For example, let $\mathcal{A}$ denote a matrix interpretation, α an arbitrary assignment and $t = \mathsf{f}(\mathsf{g}(\mathsf{a}, \mathsf{b}), \mathsf{c})$. Then

$$[\alpha]_{\mathcal{A}}(t) = F_1 G_1 \vec{a} + F_1 G_2 \vec{b} + F_1 \vec{g} + F_2 \vec{c} + \vec{f} \,.$$

Clearly the length of each product is at most the depth of the term, which is smaller or equal to its size. Hence the entries in each product are polynomially bounded (with degree $d-1$) in the size of t. The number of products equals the number of subterms of t, which is exactly the size of t. Therefore, the entries in $[\alpha]_{\mathcal{A}}(t)$ are bounded by a polynomial of degree d in the size of t. This observation leads us directly to the main result of this section.

THEOREM 6. *If a TRS $\mathcal{R}$ admits a triangular matrix interpretation $\mathcal{A}$ of dimension d, then the derivational complexity of $\mathcal{R}$ is bounded by a polynomial of degree d.*

PROOF. Any k-step derivation $s \to_{\mathcal{R}}^{k} t$ implies $[\alpha]_{\mathcal{A}}(s) >^{k} [\alpha]_{\mathcal{A}}(t)$, referring to the k-th iterate of the relation $>$ on $\mathbb{N}^d$ defined earlier. This implies $[\alpha]_{\mathcal{A}}(s)_1 \geq k + [\alpha]_{\mathcal{A}}(t)_1 \geq k$. So the top entry in $[\alpha]_{\mathcal{A}}(s)$ bounds the length of any derivation starting at s. In conjunction with the above observation, this suffices to prove the theorem. ∎

Example 7. It is easy to see that the derivational complexity of the following TRS $\mathcal{R}_1 = \{\mathsf{a}(\mathsf{b}(x)) \to \mathsf{b}(\mathsf{a}(x)), \mathsf{c}(\mathsf{a}(x)) \to \mathsf{b}(\mathsf{c}(x)), \mathsf{c}(\mathsf{b}(x)) \to \mathsf{a}(\mathsf{c}(x)))\}$ is (at least) cubic. The following TMI $\mathcal{A}$ is compatible with $\mathcal{R}_1$.

$$\mathsf{a}_{\mathcal{A}}(\vec{x}) = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} \vec{x} + \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \qquad \mathsf{b}_{\mathcal{A}}(\vec{x}) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} \vec{x} + \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

$$\mathsf{c}_{\mathcal{A}}(\vec{x}) = \begin{pmatrix} 1 & 1 & 3 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \vec{x} + \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$$

Applying Theorem 6 we conclude that the derivational complexity function with respect to $\mathcal{R}_1$ is cubic.

Despite Example 7 the criterion is in general not complete. There exist TRSs of polynomial derivational complexity that do not admit a compatible TMI. One such example can be found in [5]: $\mathcal{R}_2 = \{\mathsf{f}(\mathsf{a}, \mathsf{b}) \to \mathsf{f}(\mathsf{b}, \mathsf{b}), \mathsf{f}(\mathsf{b}, \mathsf{a}) \to \mathsf{f}(\mathsf{a}, \mathsf{a})\}$. Then $\mathcal{R}_2$ has linear derivational complexity, but in fact no compatible matrix interpretation can exist, cf. [5]. Even if there is a compatible matrix interpretation and the complexity of the system is polynomial, it might be lacking a triangular interpretation. Consider the following TRS: $\mathcal{R}_3 = \{\mathsf{a}(\mathsf{a}(x)) \to \mathsf{b}(\mathsf{c}(x)), \mathsf{b}(\mathsf{b}(x)) \to \mathsf{a}(\mathsf{c}(x)), \mathsf{c}(\mathsf{c}(x)) \to \mathsf{a}(\mathsf{b}(x))\}$, a straightforward adaption of the string rewrite system `z086` introduced by Zantema as TRS. We conjecture that $\mathcal{R}_3$ admits at most polynomial derivational complexity and is not compatible with a triangular matrix interpretation. (This is related to the open problem number 105 in the RTA list of open problems, see `http://rtaloop.pps.jussieu.fr/`.)

We conclude this section by considering the following example, that can be handled automatically by TMIs, but not with any other known method. (See Section 5 for further details about the implementation.)

Example 8. Consider the TRS $\mathcal{R}_4$ with the following rewrite rules, which is example 4.30 from [17]:

$$\begin{aligned} \mathsf{f}(\mathsf{nil}) &\to \mathsf{nil} & \mathsf{g}(\mathsf{nil}) &\to \mathsf{nil} \\ \mathsf{f}(\mathsf{nil} \circ y) &\to \mathsf{nil} \circ \mathsf{f}(y) & \mathsf{g}(x \circ \mathsf{nil}) &\to \mathsf{g}(x) \circ \mathsf{nil} \\ \mathsf{f}((x \circ y) \circ z) &\to \mathsf{f}(x \circ (y \circ z)) & \mathsf{g}(x \circ (y \circ z)) &\to \mathsf{g}((x \circ y) \circ z) \end{aligned}$$

It is not difficult to check that the following TMI $\mathcal{A}$ of dimension 4 is compatible with $\mathcal{R}_4$.

$$\circ_{\mathcal{A}}(\vec{x},\vec{y}) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \vec{x} + \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \vec{y} + \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \qquad \mathsf{nil}_{\mathcal{A}} = \begin{pmatrix} 0 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

$$\mathsf{f}_{\mathcal{A}}(\vec{x}) = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \vec{x} + \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} \qquad \mathsf{g}_{\mathcal{A}}(\vec{x}) = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \vec{x} + \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \end{pmatrix}$$

Due to Theorem 6 we conclude that $\mathsf{dc}_{\mathcal{R}_4}$ can be asymptotically bounded by a polynomial of degree 4. Notice that this bound is not optimal, as it is easy to see that the derivational complexity is quadratic.

4 Context Dependent Interpretations and Matrices

In this section, we show a tight correspondence between (triangular) matrix interpretations as introduced in Section 3 and context dependent interpretations, see [9]. More precisely we define a subclass of context dependent interpretations such that any such interpretation $\mathcal{C}$ gives rise to a restricted TMI $\mathcal{A}$ and vice versa. Moreover $\mathcal{C}$ is compatible with a TRS $\mathcal{R}$ if and only if $\mathcal{A}$ is compatible with $\mathcal{R}$.

We recall context dependent interpretations. For that we follow the presentation in [15] in a simplified form. See [9, 15] for motivating examples and intuitions behind the definitions. A *context dependent $\mathcal{F}$-algebra* (*CDA* for short) $\mathcal{C}$ is a family of $\mathcal{F}$-algebras over the reals. A CDA $\mathcal{C}$ associates to each function symbol $f \in \mathcal{F}$ of arity n, a collection of $n+1$ mappings $f_{\mathcal{C}} \colon \mathbb{R}^+ \times (\mathbb{R}_0^+)^n \to \mathbb{R}_0^+$ and $f_{\mathcal{C}}^i \colon \mathbb{R}^+ \to \mathbb{R}^+$ for all $1 \leqslant i \leqslant n$. As usual $f_{\mathcal{C}}$ is called the *interpretation function*, while the mappings $f_{\mathcal{C}}^i$ are called the *parameter functions*. In addition $\mathcal{C}$ is equipped with a set $\{>_\Delta \mid \Delta \in \mathbb{R}^+\}$ of strict orders, where we define: $z >_\Delta z'$ if and only if $z - z' \geqslant \Delta$. Let $\mathcal{C}$ be a CDA and let a Δ-*assignment* denote a mapping $\alpha : \mathbb{R}^+ \times \mathcal{V} \to \mathbb{R}_0^+$. We define a mapping $[\alpha, \Delta]_{\mathcal{C}}$ from the set of terms into the set $\mathbb{R}_0^+$ of non-negative reals:

$$[\alpha,\Delta]_{\mathcal{C}}(t) := \begin{cases} \alpha(\Delta, t) & \text{if } t \in \mathcal{V} \\ f_{\mathcal{C}}(\Delta, [\alpha, f_{\mathcal{C}}^1(\Delta)]_{\mathcal{C}}(t_1), \ldots, [\alpha, f_{\mathcal{C}}^n(\Delta)]_{\mathcal{C}}(t_n)) & \text{if } t = f(t_1, \ldots, t_n)\,. \end{cases}$$

We fix some notational conventions: Due to the special role of the additional variable Δ, we often write $f_{\mathcal{C}}[\Delta](z_1, \ldots, z_n)$ instead of $f_{\mathcal{C}}(\Delta, z_1, \ldots, z_n)$. If t is a ground term, we sometimes write $[\Delta]_{\mathcal{C}}(t)$ instead of $[\alpha, \Delta]_{\mathcal{C}}(t)$. We say that a CDA $\mathcal{C}$ is Δ-*monotone* if for all

$\Delta \in \mathbb{R}^+$ and for all $a_1, \ldots, a_n, b \in \mathbb{R}_0^+$ with $a_i >_{f^i_{\mathcal{C}}(\Delta)} b$ for some $i \in \{1, \ldots, n\}$, we have $f_{\mathcal{C}}[\Delta](a_1, \ldots, a_i, \ldots, a_n) >_\Delta f_{\mathcal{C}}[\Delta](a_1, \ldots, b, \ldots, a_n)$. A CDA $\mathcal{C}$ and a TRS $\mathcal{R}$ are *compatible* if for every rewrite rule $l \to r \in \mathcal{R}$, every $\Delta \in \mathbb{R}^+$, and any Δ-assignment α: $[\alpha, \Delta](l) >_\Delta [\alpha, \Delta](r)$ holds.

DEFINITION 9. *A* Δ^2-interpretation *is a CDA* $\mathcal{C}$ *with interpretation functions and parameter functions of the following form:*

$$f_{\mathcal{C}}(\Delta, z_1, \ldots, z_n) = \sum_{i=1}^{n} a_{(f,i)} z_i + \sum_{i=1}^{n} b_{(f,i)} z_i \Delta + g_f + h_f \Delta \tag{†}$$

$$f^i_{\mathcal{C}}(\Delta) = \frac{c_{(f,i)} + d_{(f,i)}\Delta}{a_{(f,i)} + b_{(f,i)}\Delta}\,, \tag{‡}$$

where $a_{(f,i)} > 0$ *or* $b_{(f,i)} > 0$ *(for each* $f \in \mathcal{F}$, $1 \leqslant i \leqslant n$*) and the occurring coefficients are natural numbers.*

The following lemma is a direct consequence of the definitions.

LEMMA 10. *Let* $\mathcal{C}$ *denote a* Δ^2*-interpretation. If for all* $f \in \mathcal{F}$, $1 \leqslant i \leqslant n$ *in* (‡)*, we have* $d_{(f,i)} \geqslant 1$*, then* $\mathcal{C}$ *is* Δ*-monotone.*

In [15] two (strict) subclasses of Δ^2-interpretations were studied: Δ-linear interpretations and Δ-restricted interpretations. A Δ-*linear interpretation* is a Δ^2-interpretation, where for the parameter functions as presented in (‡) we have $c_{(f,i)} = 0$ and $d_{(f,i)} = 1$ for all $f \in \mathcal{F}$, $1 \leqslant i \leqslant n$, and a Δ-*restricted interpretation* is a Δ-linear interpretation with the additional requirement that $a_{(f,i)} \in \{0, 1\}$.

Example 11 (continued from Example 1). Consider the following Δ-linear interpretation $\mathcal{C}$:

$$\circ_{\mathcal{C}}[\Delta](x, y) = (1 + \Delta)x + y + 1 \qquad \circ^1_{\mathcal{C}}(\Delta) = \frac{\Delta}{1 + \Delta} \qquad \circ^2_{\mathcal{C}}(\Delta) = \Delta$$

For all ground terms r, s, t, we have $[\Delta]_{\mathcal{C}}((r \circ s) \circ t) - [\Delta]_{\mathcal{C}}(r \circ (s \circ t)) \geqslant \Delta$. This is shown in [9, Lemma 3] by an inductive argument. However, this argument is not well-suited for automation: The implementation described in [15] doesn't find this interpretation.

Example 12 (continued from Example 11). The Δ-linear interpretation $\mathcal{C}$ is also a Δ-restricted interpretation. Due to [15, Theorem 29] we conclude quadratic derivational complexity for $\mathcal{R}$. Note that this upper bound is optimal.

It seems worthy of note, that the matrix interpretation $\mathcal{A}$ employed in Example 4 is obtained fully automatically, while the context dependent interpretation $\mathcal{C}$ is obtained by hand. On the other hand $\mathcal{A}$ and $\mathcal{C}$ use exactly the same coefficients. We exploit this observation below.

DEFINITION 13. *Let* $\mathcal{C}$ *be a CDA and let* $\mathcal{A}$ *be a matrix interpretation over two dimensions. We say the* Δ*-assignment* $\alpha\colon \mathbb{R}^+ \times \mathcal{V} \to \mathbb{R}_0^+$ *and the assignment* $\alpha'\colon \mathcal{V} \to \mathbb{N}^2$ *are* corresponding *if for all variables* x *and all* $\Delta \in \mathbb{R}^+$*:* $\alpha(\Delta, x) = a + b\Delta$ *if and only if* $\alpha'(x) = \begin{pmatrix} b \\ a \end{pmatrix}$.

We arrive at the main lemma of this section, whose technical, but not difficult proof has been omitted due to space restrictions.

LEMMA 14. *Let $\mathcal{C}$ denote a Δ^2-interpretation:*

$$f_{\mathcal{C}}(\Delta, z_1, \ldots, z_n) = \sum_{i=1}^{n} a_{(f,i)} z_i + \sum_{i=1}^{n} b_{(f,i)} z_i \Delta + g_f + h_f \Delta$$
$$f^i_{\mathcal{C}}(\Delta) = \frac{c_{(f,i)} + d_{(f,i)}\Delta}{a_{(f,i)} + b_{(f,i)}\Delta} ,$$

and let $\mathcal{A}$ denote a matrix interpretation of the following form:

$$f_{\mathcal{A}}(x_1, \ldots, x_n) = (\sum_{i=1}^{n} \begin{pmatrix} d_{(f,i)} & b_{(f,i)} \\ c_{(f,i)} & a_{(f,i)} \end{pmatrix} \cdot x_i) + \begin{pmatrix} h_f \\ g_f \end{pmatrix} ,$$

where $d_{(f,i)} \geqslant 1$, and either $a_{(f,i)} > 0$ or $b_{(f,i)} > 0$ for all $f \in \mathcal{F}$ and $1 \leqslant i \leqslant n$. Then for any term t

$$[\alpha, \Delta]_{\mathcal{C}}(t) = s_1 + s_2\Delta \quad \Longleftrightarrow \quad [\alpha']_{\mathcal{A}}(t) = \begin{pmatrix} s_2 \\ s_1 \end{pmatrix} ,$$

whenever α and α' are corresponding.

We say a matrix interpretation $\mathcal{A}$ *corresponds* to a Δ^2-interpretation $\mathcal{C}$, if $\mathcal{A}$ and $\mathcal{C}$ are defined as in Lemma 14.

Example 15. Consider the triangular matrix interpretation $\mathcal{A}$ introduced in Example 4 and the Δ-restricted interpretation $\mathcal{C}$ from Example 11. Then it is easy to see that $\mathcal{A}$ and $\mathcal{C}$ are corresponding.

THEOREM 16. *Let $\mathcal{R}$ be a TRS and let $\mathcal{C}$ be a Δ^2-interpretation such that $\mathcal{R}$ is compatible with $\mathcal{C}$. Then there exists a corresponding matrix interpretation $\mathcal{A}$ (of dimension 2) compatible with $\mathcal{R}$.*

PROOF. Let $\alpha \colon \mathcal{V} \to \mathbb{N}^2$ be arbitrary, but fixed. To prove the theorem, it suffices to verify that for any rule $l \to r \in \mathcal{R}$: $[\alpha]_{\mathcal{A}}(l) > [\alpha]_{\mathcal{A}}(r)$ holds, where $\mathcal{A}$ is the matrix interpretation constructed in Lemma 14. To apply Lemma 14, we choose a Δ-assignment $\alpha' \colon \mathbb{R}^+ \times \mathcal{V} \to \mathbb{R}^+_0$ that corresponds to α. For every $l \to r \in \mathcal{R}$, for every $\Delta \in \mathbb{R}^+$, and every $\alpha' \colon \mathbb{R}^+ \times \mathcal{V} \to \mathbb{R}^+_0$, we have $\big([\alpha', \Delta]_{\mathcal{C}}(l) - [\alpha', \Delta]_{\mathcal{C}}(r)\big) = a + b\Delta$. Here we make use of the fact that for any term t: $[\alpha', \Delta]_{\mathcal{C}}(t) = c + d\Delta$. This follows from an inductive argument employing the assumed form of the assignment α'. Moreover, as $a + b\Delta \geqslant \Delta$, we conclude $a \geqslant 0$ and $b \geqslant 1$. Thus an application of Lemma 14 yields

$$\big([\alpha]_{\mathcal{A}}(l) - [\alpha]_{\mathcal{A}}(r)\big) = \begin{pmatrix} b \\ a \end{pmatrix} \geqslant \begin{pmatrix} 1 \\ 0 \end{pmatrix} ,$$

from which compatibility with $\mathcal{A}$ directly follows. ∎

An easy consequence of Theorem 16 in conjunction with Theorem 6 is that Δ^2-interpretations induce at most exponential derivational complexity. In particular, we obtain the following corollary. (A direct proof of this result, i.e., a proof that argues only about context dependent interpretations, can be found in [15].)

COROLLARY 17. *Let $\mathcal{R}$ be a TRS and let $\mathcal{C}$ denote a Δ-linear interpretation compatible with $\mathcal{R}$. Then $\mathcal{R}$ is terminating and $\mathsf{dc}_{\mathcal{R}}(n) = 2^{\mathsf{O}(n)}$. Moreover, if $\mathcal{C}$ is a Δ-restricted interpretation, then $\mathsf{dc}_{\mathcal{R}}(n) = \mathsf{O}(n^2)$. Observe that the bounds are tight, i.e., we can find TRSs $\mathcal{R}$ that fulfill these requirements, such that $\mathsf{dc}_{\mathcal{R}}$ is an exponential or quadratic function, respectively.*

Theorem 16 raises the question, whether the other direction may hold for Δ^2-interpretations: Given a matrix interpretation, compatible with $\mathcal{R}$, does there exist a Δ^2-interpretation that is compatible with $\mathcal{R}$? Recall that a CDA $\mathcal{C}$ is compatible with a TRS $\mathcal{R}$, if for every $l \to r \in \mathcal{R}$, for every $\Delta \in \mathbb{R}^+$, and every $\alpha\colon \mathbb{R}^+ \times \mathcal{V} \to \mathbb{R}_0^+$, we have $[\alpha, \Delta]_{\mathcal{C}}(l) - [\alpha, \Delta]_{\mathcal{C}}(r) \geqslant \Delta$. The next example shows that this definition of compatibility is too general in this context.

Example 18 (continued from Example 4). The Δ-restricted interpretation $\mathcal{C}$ is not compatible with $\mathcal{R}$ as defined above, i.e. we do not have $[\alpha, \Delta]_{\mathcal{C}}((x \circ y) \circ z) - [\alpha, \Delta]_{\mathcal{C}}(x \circ (y \circ z)) \geqslant \Delta$ for arbitrary assignments α. To construct a counter-example we set $\alpha(\Delta, x) := \Delta^2$, and $\alpha(\Delta, u)$ is arbitrary for $u \neq x$. Following the proof of Lemma 3 in [9], we obtain

$$[\alpha, \Delta]_{\mathcal{C}}((x \circ y) \circ z) - [\alpha, \Delta]_{\mathcal{C}}(x \circ (y \circ z)) = (1 + 2\Delta)[\alpha, \frac{\Delta}{1 + 2\Delta}]_{\mathcal{C}}(x) +$$
$$+\Delta - (1 + \Delta)[\alpha, \frac{\Delta}{1 + \Delta}]_{\mathcal{C}}(x) = \Delta + \frac{\Delta^2}{1 + 2\Delta} - \frac{\Delta^2}{1 + \Delta} \not\geqslant \Delta\,.$$

This violates the compatibility condition.

Example 18 motivates the next definition.

DEFINITION 19. *We say a Δ-assignment $\alpha\colon \Delta \times \mathcal{V} \to \mathbb{R}_0^+$ is* linear*, if there exist natural numbers a and b, such that $\alpha(\Delta, x) = a + b\Delta$.*

Example 20 (continued from Example 18). For any linear Δ-assignment α, we have $[\alpha, \Delta]_{\mathcal{C}}((x \circ y) \circ z) - [\alpha, \Delta]_{\mathcal{C}}(x \circ (y \circ z)) \geqslant \Delta$. This can be seen by just applying a linear Δ-assignment of the following parametric form: $\alpha(\Delta, x) = x_1 + x_2\Delta$, $\alpha(\Delta, y) = y_1 + y_2\Delta$, and $\alpha(\Delta, z) = z_1 + z_2\Delta$.

LEMMA 21. *Let σ be a ground substitution and let $\mathcal{C}$ be a Δ^2-interpretation. Then there exists a linear Δ-assignment α such that $[\Delta]_{\mathcal{C}}(t\sigma) = [\alpha, \Delta]_{\mathcal{C}}(t)$ for all $\Delta \in \mathbb{R}^+$ and terms t.*

PROOF. We set $\alpha(\Delta, x) = [\Delta]_{\mathcal{C}}(x\sigma)$ for any $x \in \mathsf{dom}(\sigma)$ and $\alpha(\Delta, x) = 0$ otherwise. The fact that $[\Delta]_{\mathcal{C}}(x\sigma)$ has a linear shape can be shown by an easy induction on $x\sigma$. ∎

By now, the following main result of this section, is an easy consequence of Lemma 14, Theorem 16 and Lemma 21.

THEOREM 22. *Let $\mathcal{A}$ be a monotone matrix interpretation of dimension 2 such that no zero column occurs in any matrix, let $\mathcal{C}$ be the corresponding Δ^2-interpretation and let $\mathcal{R}$ be a TRS. Then $\mathcal{A}$ is compatible with $\mathcal{R}$ if and only if for all linear Δ-assignments α, all $\Delta \in \mathbb{R}^+$ and all rules $l \to r \in \mathcal{R}$, we have $[\alpha, \Delta]_{\mathcal{C}}(l) >_{\Delta} [\alpha, \Delta]_{\mathcal{C}}(r)$.*

Note that the restriction on zero columns expressed in the assumptions of the theorem appears to be negligible, if we consider the automation of the introduced techniques. This is the subject of the next section.

Table 1: Termination Methods as Complexity Analysers

	BOUNDS	CDI	TMI				TMI+BOUNDS
dimension			2	3	4	5	3
# successes	125	85	143	158	154	156	216
avg. success time	0.68	3.84	0.19	1.33	0.56	2.39	8.65
# timeouts	328	272	66	224	237	244	237

5 Experiments

We have implemented the methods described in this paper, and tested their viability to analyse polynomial derivational complexity on version 4.0 of the Termination Problem Data Base (TPDB for short), which is used in the annual RTA termination competition. (Available at `http://www.lri.fr/~marche/tpdb/`, but we also included the secret systems from the competition 2007.) This database contains a total of 1381 TRSs, 957 of which are known to be terminating. Arguably, the TPDB is an imperfect choice as it has been designed to test the strength of termination provers in rewriting, not as a testbed to analyse feasible bounds on the derivational complexity of TRSs. Examples such as TRS encodings of the Ackermann function and the Hydra Battle reinforce this point. On the other hand, the TPDB is the only relatively large collection of TRSs that is publicly available.

We briefly sketch our implementation: Similar to [3], we build a set of Diophantine constraints which express all necessary restrictions on the matrix interpretation. Then, we put a finite upper bound on the variables in the constraints and encode these constraints as a problem of propositional logic (see [6] but also [5]). We give the final SAT problem to MiniSAT [4] and use a satisfying assignment to construct a suitable matrix interpretation, where we use the fact that all matrix products are upper triangular, so values for entries below the main diagonal can be ignored.

In order to compare Δ-restricted interpretations (referred to by CDI) and triangular matrix interpretations (TMI) to other results, we compared them to the implementation of the match-bound technique (BOUNDS for short) as in [13]: Linear TRSs are tested for match-boundedness, non-linear, but non-duplicating TRSs are tested for match-raise-boundedness. It is not difficult to see that this technique implies *linear* derivational complexity. Last, we tested the union of the two strongest methods (BOUNDS and TMI for dimension 3) by using half of the time on TMI and the rest on BOUNDS. For both CDI and TMI, we restricted all coefficients to at most 15 (allowing us to use at most 4 bits to encode each coefficient).

All tests were executed single-threaded on a server equipped with 8 AMD Opteron™ 2.8 GHz dual core processors with 64GB of RAM. We used a timeout of 60 seconds for each TRS. The results of the tests are shown in Table 1 (see `http://cl-informatik.uibk.ac.at/users/aschnabl/experiments/08msw/` for the full experimental evidence). The times given in the table are seconds. (In examples for which the according method was neither successful nor had a timeout, the proof attempt was given up before the timeout.)

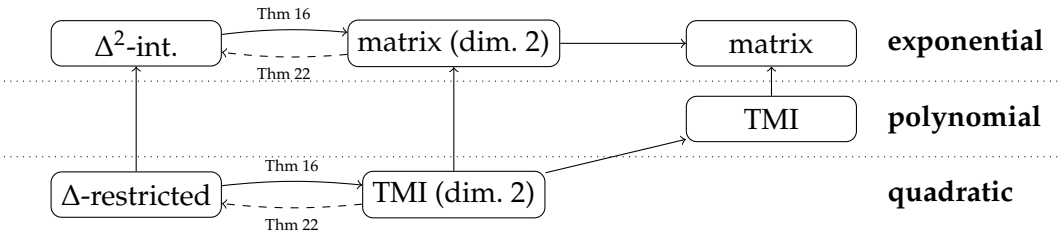
As we can see, triangular matrix interpretations are clearly the most powerful method for proving polynomial derivational complexity of rewriting on our testbed. As suggested

by our results in Section 4, the systems that can be handled by Δ-restricted interpretations are a strict subset of the problems solved by TMI with matrices of dimension 2. We want to note that out of the latter 143 systems, 140 can still be handled with the restriction on zero columns in Theorem 22. In total, TMIs of dimensions 2 to 5 are successful in 162 instances.

It is worthy of note that "full" matrix interpretations (for dimensions 2 to 5 and coefficients at most 15) can handle (only) 222 TRSs. Hence a clear majority of those systems that can in principle shown to be terminating with matrix interpretations have polynomial derivational complexity.

6 Conclusion

In this paper we studied the complexity of rewrite systems $\mathcal{R}$ as expressed by the derivational complexity function $\mathsf{dc}_{\mathcal{R}}$. The following diagram provides a condensed view of the studied classes of matrix and context dependent interpretations, where the right column gives the induced derivational complexity. (The arrows depict set inclusions and the dashed arrows refer to the additional restriction on the zero columns.)



We emphasise the pictured correspondence result: Consider Δ^2-interpretations and triangular matrix interpretations of dimension 2, where no zero columns occur, then these interpretations are *equivalent* for orientability. This correspondence sheds light on the expressivity of matrix interpretations and (significantly) extends the class of rewrite systems whose compatibility with context dependent interpretations can be shown automatically. As witnessed by Example 1, it is sometimes possible to automatically obtain a context-dependent interpretation via the correspondence result, where the direct approach fails. The mentioned techniques have been implemented and the experimental data clearly shows that triangular matrix interpretations extend the power of previously known methods to automatically analyse polynomial derivational complexity. In particular, the TMI method is the only known automatic method to prove polynomial complexity for Example 1, 7, and 8.

In concluding, we note that matrix interpretations for termination of string rewriting systems are also known as $\mathbb{N}$-rational series in the theory of weighted (tree) automata, and they have been investigated in connection with the growth of DT0L systems (see [16]). It remains to connect this knowledge to the present application. Other directions for research are concerned with extending the presented *direct* termination techniques with transformation techniques like the *dependency pair method* [1] or with *multi-step termination proofs* as invoked in [5]. Although these extensions are necessary to increase the power of the studied techniques further, already simple examples show the challenging nature of this endeavour, compare [5].

Bibliography

[1] T. Arts and J. Giesl. Termination of term rewriting using dependency pairs. *TCS*, 236:133–178, 2000.

[2] F. Baader and T. Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.

[3] E. Contejean, C. Marché, A.-P. Tomás, and X. Urbain. Mechanically proving termination using polynomial interpretations. *JAR*, 34(4):325–363, 2005.

[4] N. Eèn and N. Sõrensson. An Extensible SAT-solver. In *Proc. 6th SAT*, volume 2919 of *LNCS*, pages 272–286, 2003.

[5] J. Endrullis, J. Waldmann, and H. Zantema. Matrix interpretations for proving termination of term rewriting. *JAR*, 40(3):195–220, 2008.

[6] C. Fuhs, J. Giesl, A. Middeldorp, P. Schneider-Kamp, R. Thiemann, and H. Zankl. SAT solving for termination analysis with polynomial interpretations. In *Proc. 10th SAT*, volume 4501 of *LNCS*, pages 340–354, 2007.

[7] A. Geser, D. Hofbauer, J. Waldmann, and H. Zantema. On tree automata that certify termination of left-linear term rewriting systems. *IC*, 205(4):512–534, 2007.

[8] D. Hofbauer. Termination proofs by multiset path orderings imply primitive recursive derivation lengths. *TCS*, 105(1):129–140, 1992.

[9] D. Hofbauer. Termination proofs by context-dependent interpretations. In *Proc. 12th RTA*, volume 2051 of *LNCS*, pages 108–121, 2001.

[10] D. Hofbauer and C. Lautemann. Termination proofs and the length of derivations. In *Proc. 3rd RTA*, volume 355 of *LNCS*, pages 167–177, 1989.

[11] D. Hofbauer and J. Waldmann. Termination of string rewriting with matrix interpretations. In *Proc. 17th RTA*, volume 4098 of *LNCS*, pages 328–342, 2006.

[12] A. Koprowski and J. Waldmann. Artic termination ... below zero. In *Proc. 19th RTA*, volume 5117 of *LNCS*, pages 202–216, 2008.

[13] M. Korp and A. Middeldorp. Proving termination of rewrite systems using bounds. In *Proc. 18th RTA*, volume 4533 of *LNCS*, pages 273–287, 2007.

[14] G. Moser. Derivational complexity of Knuth Bendix orders revisited. In *Proc. 13th LPAR*, volume 4246 of *LNCS*, pages 75–89, 2006.

[15] G. Moser and A. Schnabl. Proving quadratic derivational complexities using context dependent interpretations. In *Proc. 19th RTA*, volume 5117 of *LNCS*, pages 276–290, 2008.

[16] G. Rozenberg and A. Salomaa. *The Mathematical Theory of L Systems*. Academic Press, New York, 1980.

[17] J. Steinbach and U. Kühler. Check your ordering – termination proofs and open problems. Technical Report SR-90-25, Universität Kaiserslautern, 1990.

[18] Terese. *Term Rewriting Systems*, volume 55 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2003.

[19] A. Weiermann. Termination proofs for term rewriting systems with lexicographic path orderings imply multiply recursive derivation lengths. *TCS*, 139:355–362, 1995.

Foundations of Software Technology and Theoretical Computer Science (Bangalore) 2008.
Editors: R. Hariharan, M. Mukund, V. Vinay; pp 316-327

Analyzing the Implicit Computational Complexity of object-oriented programs

.

Jean-Yves Marion[1], **Romain Péchoux**[2]

[1] LORIA, Carte team, and ENSMN, INPL, Nancy-Université,
Nancy, France.
jean-yves.marion@loria.fr

[2] Department of Computer Science, Trinity College,
Dublin, Ireland
romain.pechoux@cs.tcd.ie

ABSTRACT. A sup-interpretation is a tool which provides upper bounds on the size of the values computed by the function symbols of a program. Sup-interpretations have shown their interest to deal with the complexity of first order functional programs. This paper is an attempt to adapt the framework of sup-interpretations to a fragment of object-oriented programs, including loop and while constructs and methods with side effects. We give a criterion, called brotherly criterion, which uses the notion of sup-interpretation to ensure that each brotherly program computes objects whose size is polynomially bounded by the inputs sizes. Moreover we give some heuristics in order to compute the sup-interpretation of a given method.

1 Introduction

Computer security is defined as ensuring confidentiality, integrity and availability requirements in whatever context [6]. For example, a secured system should resist to a buffer-overflow. In this paper, we focus on analyzing the complexity of object-oriented programs, that is the number of objects created by a program during its execution, by static analysis. For that purpose, we use semantics interpretation tools called sup-interpretations. Sup-interpretations were introduced in [14, 15] in order to study the complexity of first order functional programs. A sup-interpretation consists in a function which provides an upper bound on the size of the values computed by some symbol of a given program. The notion of sup-interpretation strictly generalizes the notion of quasi-interpretation [8] (i.e. analyzes the complexity of strictly more algorithms) which has already been used to perform Bytecode verification [4] and which has been extended to reactive programs [5, 10]. Sup-interpretations allow to characterize complexity classes and, in particular, the class of NC^k functions [7, 16].

A major challenge consists in the adaptation of such an analysis to object-oriented programs with respect to the following points. Firstly, we have to carefully translate the notion of sup-interpretation from the functional paradigm to the object-oriented paradigm, taking into account the new object features such as method calls or side effects. Secondly, we also want to ensure the viability of our study by obtaining heuristics to compute sup-interpretations.

FSTTCS 2008
IARCS Annual Conference on
Foundations of Software Technology and Theoretical Computer Science
http://drops.dagstuhl.de/opus/volltexte/2008/1763

The considered language is inspired by the Featherweight Java of [12] and is a fragment of the Java language of [11] which includes side effects and to which we add loop and while constructs. This language is a purely object-oriented language like SmallTalk. For simplicity, inheritance, typing and subtyping are not considered in this paper. However, the analysis presented in this paper can be extended without restriction to a Java-like language including primitives types such as characters, integers or booleans.

Our work is a continuation of recent studies on the Implicit Computational Complexity of imperative programs [18, 13]. Contrarily to these seminal works, we work on polynomial algebra instead of matrix algebra. There are at least two reasons for such an approach. Firstly, the use of polynomials gives a clearest intuition and pushes aside a lot of technicalities. Secondly, polynomials give more flexibility in order to deal with method calls, which is essential in order to study the object oriented paradigm. Some studies on the cost analysis of Java Bytecode have already been developed in [1, 2]. In this paper, we make a distinct choice by considering a more formal and restricted language. We perform the analysis at the language level and not at the Bytecode level. The pros are that our study has more formal basis and more portability (i.e. it can easily be adapted to distinct object-oriented languages). The cons are the restrictions on the considered language. However, these restrictions are put in order to make the study more comprehensible and we claim that they could be withdraw without any difficulty.

The paper is organized as follows. After introducing our language and the notion of sup-interpretation of an object-oriented program, we give a criterion, called brotherly criterion, which ensures that each brotherly program computes objects whose size is polynomially bounded by the input size. Then, we extend this criterion to methods, thus obtaining heuristics for synthesizing sup-interpretations of non-recursive methods.

2 Object-oriented Programs

2.1 Syntax of programs

A program is composed by a sequence of classes, including a main class, which are named by class identifiers in `Class`. A class $\mathtt{C} \in \mathtt{Class}$ is composed by a sequence of attribute declarations, a constructor and a sequence of methods. The main class `main` is only composed by attribute declarations and commands, i.e. there is no method and no constructor in `main`. `var` X; corresponds to the declaration of the attribute X, where X represents a field of a given class and belongs to a fixed set $\mathcal{X}$. A method is composed by a method identifier `f` belonging to a set $\mathcal{F}$, a sequence of arguments $x_1, \ldots, x_n \in \mathcal{P}$, also called parameters, and a command `Cm` and is of the shape $\mathtt{f}(x_1, \ldots, x_n)\,\{\mathtt{Cm}\,;\,\mathtt{return}\ X;\}$, where the attribute X corresponds to the field returned as output. A constructor $\mathtt{C}(x_1, ..., x_n)\,\{X_1 := x_1; \ldots; X_n := x_n\}$ assigns a parameter to each attribute of the corresponding class. Throughout the paper, we use capital letters X, Y, Z and lower-case letters x, y, z in order to make the distinction between attributes and, respectively, parameters. A command is either the skip command, a variable assignment, a sequence of commands $\mathtt{Cm}_1;\mathtt{Cm}_2$, a loop command, a while command or a conditional command. An expression is either a parameter x, an attribute X, the null reference or the creation of a new object using a constructor. A method call is of the shape

$X.\mathtt{f}(e_1,\ldots,e_n)$, with $\mathtt{f} \in \mathcal{F}$, $X \in \mathcal{X}$ and with $e_1,\ldots,e_n$ expressions. The precise syntax of the language is summed up by the following grammar:

$$
\begin{array}{lll}
\texttt{Attributes} & \ni A & ::= \mathtt{var}\ X; \mid \mathtt{var}\ X; A \\
\texttt{Expressions} & \ni e & ::= x \mid X \mid \mathtt{null} \mid \mathtt{new}\ \mathtt{C}(e_1,\ldots,e_n) \\
\texttt{Method call} & \ni a & ::= X.\mathtt{f}(e_1,\ldots,e_n) \\
\texttt{Commands} & \ni \mathtt{Cm} & ::= \mathtt{skip} \mid X := a \mid X := e \mid \mathtt{Cm}_1;\mathtt{Cm}_2 \mid \mathtt{loop}\ X\{\mathtt{Cm}\} \\
 & & \mid \mathtt{if}(e)\mathtt{then}\{\mathtt{Cm}_1\}\mathtt{else}\{\mathtt{Cm}_2\} \mid \mathtt{while}\ e\{\mathtt{Cm}\} \\
\texttt{Methods} & \ni \mathtt{M} & ::= \mathtt{f}(x_1,\ldots,x_n)\{\mathtt{Cm}\ ;\ \mathtt{return}\ X;\} \\
\texttt{Constructors} & \ni \mathtt{Cons} & ::= \mathtt{C}(x_1,\ldots,x_n)\{X_1 := x_1;\ldots;X_n := x_n\} \\
\texttt{Class} & \ni \mathtt{C} & ::= \mathtt{Class}\ \mathtt{C}\{A\ \mathtt{Cons}\ M_1 \ldots M_n\} \\
 & \mathtt{main} & ::= \mathtt{Class}\ \mathtt{main}\{A\ \mathtt{Cm}\}
\end{array}
$$

Notation 1 *We will use the notation $\bar{e}$ to represent the sequence $e_1,\ldots,e_n$ when n is clear from the context.*

The sets $\mathcal{X}$, $\mathcal{P}$, $\mathcal{F}$ and `Class` are pairwise disjoint. All attributes occurring in the methods of a given class `C` must belong to the attributes of this class. All parameters occurring in the command `Cm` of a given method must belong to the parameters $x_1,\ldots,x_n$. Let $\mathcal{F}_{\mathtt{C}}$ and $\mathcal{X}_{\mathtt{C}}$ be respectively the sets of methods and attributes declared in the class `C`.

We add the following syntactic restrictions to our language: We suppose that $\mathtt{C} \neq \mathtt{C}'$ implies $\mathcal{F}_{\mathtt{C}} \cap \mathcal{F}_{\mathtt{C}'} = \mathcal{X}_{\mathtt{C}} \cap \mathcal{X}_{\mathtt{C}'} = \emptyset$. There is no method overloading. A program is not allowed to write the attribute X during the execution of a $\mathtt{loop}\ X\{\mathtt{Cm}\}$. There are neither local variables, nor static variables. All these restrictions are put in order to simplify the discussion. However we claim that they also could be analyzed by our framework.

Example 1 (Linked list) *Consider the linked list class described in figure 1. X and Y represent the head and tail attributes whereas W and Z store intermediate computations. Notice that W and Z are required since the considered language has no local variables.*

$$
\begin{array}{l|l}
\mathtt{Class}\ \mathtt{List}\{\ \mathtt{var}\ X;\ \mathtt{var}\ Y;\ \mathtt{var}\ W;\ \mathtt{var}\ Z; & W := Y; \\
\mathtt{List}(x,y,w,z)\{X := x;\ Y := y;\ W := w;\ Z := z;\} & \mathtt{loop}\ \ Y\{ \\
\mathtt{getHead}()\{\mathtt{skip}\ ;\ \mathtt{return}\ X;\} & Z := \mathtt{new}\ \mathtt{List}(W.\mathtt{getHead}(), Z, \overline{\mathtt{null}}); \\
\mathtt{getTail}()\{\mathtt{skip}\ ;\ \mathtt{return}\ Y;\} & W := W.\mathtt{getTail}(); \\
\mathtt{setTail}(y)\{Y := y\ ;\ \mathtt{return}\ X;\} & \}\ ; \\
\mathtt{reverse}()\{\quad Z := \mathtt{new}\ \mathtt{List}(X, \overline{\mathtt{null}}); & \mathtt{return}\ Z;\ \} \\
 & \}
\end{array}
$$

Figure 1: Linked list

2.2 Semantics

In this section, we define a semantics without references. This semantic weakening is not a hard restriction since we are more concerned with providing a semantics which takes into account the number of object creations than by giving a precise semantics of object-oriented programs, as it will be illustrated by remark 2.2. The domain of computation is the set of

objects (values) described in [12] and is defined inductively by:

$$\texttt{Objects} \ni o ::= \texttt{null} \mid \texttt{new C}(o_1, \ldots, o_n)$$

where $\texttt{C} \in \texttt{Class}$ is a class having n attributes and $o_1, \ldots, o_n$ are objects. Notice that objects are particular expressions, only using class constructors.

DEFINITION 1. *[Size] The size $|o|$ of an object o is defined inductively by $|o| = 0$, if $o =$ $\texttt{new C}()$, and $|o| = \sum_{i=1}^{n} |o_i| + 1$, if $o = \texttt{new C}(o_1, \ldots, o_n)$.*

Objects are created through explicit requests, using a constructor and the `new` construct. Consequently, an attribute X may be successively attached to distinct objects during the program execution. The operational semantics of our language is inspired by the operational semantics of the Java fragment given in [11]. It is closer to [11] than to [12] since we use variable assignments (i.e. there are side effects). Contrarily to [11], we do not make explicit use of references since the object description suggested above is sufficient to control program complexity (i.e. the number of object creations). In general, an object of the shape $\texttt{new C}(o_1, \ldots, o_n)$ can be viewed as an object of the class `C` with n implicit references to the objects $o_1, \ldots, o_n$.

A store σ is a partial mapping from attributes $\mathcal{X}$ and parameters $\mathcal{P}$ to objects in `Objects`. A store can be extended to expressions and method calls by $\texttt{null}\sigma = \texttt{null}$, $\texttt{new C}(e_1, \ldots, e_n)\sigma = \texttt{new C}(e_1\sigma, \ldots, e_n\sigma)$ and $X.\texttt{f}(e_1, \ldots, e_n)\sigma = X\sigma.\texttt{f}(e_1\sigma, \ldots, e_n\sigma)$. Given a store σ, the notation $\sigma\{\diamond_1 \leftarrow o_1, ..., \diamond_n \leftarrow o_n\}$ means that the object stored in $\diamond_i \in \mathcal{X} \cup \mathcal{P}$ is updated to the object o_i in σ, for each $i \in \{1, n\}$. Given an expression (or a method call) d and a store σ, the notation $\langle d, \sigma\rangle \downarrow \langle o, \sigma'\rangle$ means that d is evaluated to o and that the store σ is updated to the store σ' during this evaluation. Given a command `Cm`, we use the notation $\langle \texttt{Cm}, \sigma\rangle \downarrow \langle \sigma'\rangle$, if σ is updated to σ' during the execution of `Cm`. Given a program `p` of main class $\texttt{Class main } \{A; \texttt{Cm}\}$ and a store σ, `p` computes a store σ' defined by $\langle \texttt{Cm}, \sigma\rangle \downarrow \langle \sigma'\rangle$.

The expression `null` is evaluated to `null`. Given a store σ, a variable or a parameter $\diamond$ is evaluated to $\diamond\sigma$. The expression $\texttt{new C}(e_1, \ldots, e_n)$ is evaluated to $\texttt{new C}(o_1, \ldots, o_n)$, if the expressions $e_1, \ldots, e_n$ are evaluated to the objects $o_1, \ldots, o_n$. The operational semantics of expressions is described in figure 2.

$$\frac{}{\langle \diamond, \sigma\rangle \downarrow \langle \diamond\sigma, \sigma\rangle}\ \diamond \in \mathcal{X} \cup \mathcal{P} \qquad \frac{}{\langle \texttt{null}, \sigma\rangle \downarrow \langle \texttt{null}, \sigma\rangle}$$

$$\frac{\forall i \in \{1, n\}\ \langle e_i, \sigma\rangle \downarrow \langle o_i, \sigma\rangle}{\langle \texttt{new C}(e_1, \ldots, e_n), \sigma\rangle \downarrow \langle \texttt{new C}(o_1, \ldots, o_n), \sigma\rangle}\ \texttt{C} \in \texttt{Class}$$

Figure 2: Operational semantics of an expression

The command `skip` does nothing. The command $X := d$ assigns the object computed by d to the attribute X in the store. The command $\texttt{Cm}_1; \texttt{Cm}_2$ corresponds to the sequential execution of $\texttt{Cm}_1$ and $\texttt{Cm}_2$. $\texttt{if}(e)\texttt{then}\{\texttt{Cm}_1\}\texttt{else}\{\texttt{Cm}_2\}$ executes either the command $\texttt{Cm}_1$ or the command $\texttt{Cm}_2$ depending on whether the expression e is evaluated to the object `null` or to any other object. The command $\texttt{loop } X\{\texttt{Cm}\}$ executes $|o|$ times the command `Cm`,

if o is the object stored in X. Finally, the command $\texttt{while}\ e\,\{\texttt{Cm}\}$ is evaluated to $\texttt{skip}$, if e is evaluated to the object $\texttt{null}$, and to $\texttt{Cm;while}\ e\,\{\texttt{Cm}\}$ otherwise. The operational semantics of commands is described in figure 3.

$$\frac{}{\langle \texttt{skip}, \sigma\rangle \downarrow \langle \sigma\rangle} \qquad \frac{\langle d, \sigma\rangle \downarrow \langle o, \sigma'\rangle}{\langle X := d\rangle \downarrow \langle \sigma'\{X \leftarrow o\}\rangle}$$

$$\frac{\langle e, \sigma\rangle \downarrow \langle \texttt{null}, \sigma\rangle}{\langle \texttt{if}(e)\texttt{then}\{\texttt{Cm}_1\}\texttt{else}\{\texttt{Cm}_2\}, \sigma\rangle \downarrow \langle \texttt{Cm}_1, \sigma\rangle} \qquad \frac{\langle e, \sigma\rangle \downarrow \langle o, \sigma\rangle \quad o \neq \texttt{null}}{\langle \texttt{if}(e)\texttt{then}\{\texttt{Cm}_1\}\texttt{else}\{\texttt{Cm}_2\}, \sigma\rangle \downarrow \langle \texttt{Cm}_2, \sigma\rangle}$$

$$\frac{\langle \texttt{Cm}_1, \sigma\rangle \downarrow \langle \sigma'\rangle \quad \langle \texttt{Cm}_2, \sigma'\rangle \downarrow \langle \sigma''\rangle}{\langle \texttt{Cm}_1\texttt{;}\,\texttt{Cm}_2, \sigma\rangle \downarrow \langle \sigma''\rangle} \qquad \frac{\langle X, \sigma\rangle \downarrow \langle o, \sigma\rangle}{\langle \texttt{loop}\ \ X\,\{\texttt{Cm}\}, \sigma\rangle \downarrow \langle \underbrace{\texttt{Cm;}\ldots\texttt{;Cm}}_{|o|\text{ times}}, \sigma\rangle}$$

$$\frac{\langle e, \sigma\rangle \downarrow \langle o, \sigma\rangle \quad o \neq \texttt{null}}{\langle \texttt{while}\ \ e\,\{\texttt{Cm}\}, \sigma\rangle \downarrow \langle \texttt{Cm;while}\ \ e\,\{\texttt{Cm}\}, \sigma\rangle} \qquad \frac{\langle e, \sigma\rangle \downarrow \langle \texttt{null}, \sigma\rangle}{\langle \texttt{while}\ \ e\,\{\texttt{Cm}\}, \sigma\rangle \downarrow \langle \texttt{skip}, \sigma\rangle}$$

Figure 3: Operational semantics of a command

If $\texttt{f}$ is a method defined by $\texttt{f}(x_1, \ldots, x_m)\,\{\texttt{Cm}\,;\ \texttt{return}\ X_k;\}$ in a class $\texttt{C}$ having n attributes $X_1, \ldots, X_n$, then, given a store σ s.t. $X\sigma = \texttt{new C}(o_1, \ldots, o_n)$, the evaluation of $X.\texttt{f}(e_1, \ldots, e_m)$ is performed first by evaluating the expressions e_j to the objects p_j, then, by evaluating the command $\texttt{Cm}$ with a store $\sigma\{x_1 \leftarrow p_1, \ldots, x_m \leftarrow p_m, X_1 \leftarrow o_1, \ldots, X_n \leftarrow o_n\}$ and, finally, by returning the object stored in X_k. The operational semantics of method call is described in figure 4.

$$\frac{\begin{array}{ll}\forall i\ \langle e_i, \sigma\rangle \downarrow \langle p_i, \sigma\rangle & \text{Class C}\,\{\ldots\texttt{var}\ \ X_j;\ldots\texttt{f}(x_1, \ldots, x_m)\,\{\texttt{Cm}\,;\ \texttt{return}\ X_k;\}\} \\ X\sigma = \texttt{new C}(o_1, \ldots, o_n) & \langle \texttt{Cm}, \sigma\{X_1 \leftarrow o_1, \ldots, X_n \leftarrow o_n, x_1 \leftarrow p_1, \ldots, x_m \leftarrow p_m\}\rangle \downarrow \langle \sigma'\rangle\end{array}}{\langle X.\texttt{f}(e_1, \ldots, e_m), \sigma\rangle \downarrow \langle X_k\sigma', \sigma'\{X \leftarrow \texttt{new C}(X_1\sigma', \ldots, X_n\sigma')\}\rangle}$$

Figure 4: Operational semantics of a method call

Example 2 *Consider the following program together with the class of example 1:*

$\texttt{Class}\ \text{main}\,\{\ \texttt{var}\ U;\ \texttt{var}\ V;\ \texttt{var}\ T;\ V := \texttt{new List}(U, \overline{\texttt{null}});\ \texttt{loop}\ \ T\,\{U := V.\texttt{setTail}(V);\}\ \}$

Given a store σ such that $U\sigma = o^U$ and $T\sigma = o^T$ we have:

$$\langle V := \texttt{new List}(U, \overline{\texttt{null}}), \sigma\rangle \downarrow \left\langle \sigma \left\{ V \leftarrow \texttt{new List}(o^U, \overline{\texttt{null}}) \right\} \right\rangle$$

$$\langle U := V.\texttt{setTail}(V), \sigma\rangle \downarrow \left\langle \sigma \left\{ V \leftarrow \texttt{new List}(o^U, \texttt{new List}(o^U, \overline{\texttt{null}}), \overline{\texttt{null}}) \right\} \right\rangle$$

$$\langle \texttt{loop } T\{U := V.\texttt{setTail}(V)\}, \sigma\rangle \downarrow \left\langle \sigma \left\{ V \leftarrow \mathtt{f}^{|o^T|}(\texttt{null}) \right\} \right\rangle$$

where $\mathtt{f}(x) = \texttt{new List}(o^U, x, \overline{\texttt{null}})$ *and* $\forall n \in \mathbb{N} - \{0\}$, $\mathtt{f}^{n+1} = \mathtt{f} \circ \mathtt{f}^n$.

Remarks: The considered domain of computation is a set of terms without references and, consequently, it roughly approximates complex data structures such as cyclic data structure.

For example, given a store σ, a main program of the shape:

$$\begin{array}{l|l} X_1 := \texttt{new List}(X, \overline{\texttt{null}}); & X_0 := X_1.\texttt{setTail}(X_2); \\ X_2 := \texttt{new List}(Y, \overline{\texttt{null}}); & X_0 := X_2.\texttt{setTail}(X_1); \end{array}$$

computes a store σ' such that:

$$X_1\sigma' = \texttt{new List}(X\sigma, \texttt{new List}(Y\sigma, \overline{\texttt{null}}), \overline{\texttt{null}})$$

$$X_2\sigma' = \texttt{new List}(Y\sigma, \texttt{new List}(X\sigma, \texttt{new List}(Y\sigma, \overline{\texttt{null}}), \overline{\texttt{null}}), \overline{\texttt{null}})$$

However, this is not a serious drawback since the concern of this paper is to provide upper bounds to the number of object creations and such data are preserved by the representation of objects by terms.

3 Sup-interpretations and weights

3.1 Assignments

Let $\mathbb{R}^+$ be the set of positive real numbers.

DEFINITION 2. *[Class assignment] Given a class* C *with n attributes, the assignment $I_{\mathtt{C}}$ of the class* C *is a mapping of domain $dom(I_{\mathtt{C}}) \subseteq \mathcal{F}_{\mathtt{C}} \cup \{\mathtt{C}\}$, where $\mathcal{F}_{\mathtt{C}}$ is the set of the methods of the class* C*. It assigns a function $I_{\mathtt{C}}(\mathtt{f}) : (\mathbb{R}^+)^{m+1} \longmapsto \mathbb{R}^+$ to each method symbol $\mathtt{f} \in dom(I_{\mathtt{C}})$ of arity m and a function $I_{\mathtt{C}}(\mathtt{C}) : (\mathbb{R}^+)^n \longmapsto \mathbb{R}^+$ to the constructor* C*.*

DEFINITION 3. *[Program assignment] Given a program* p*, the assignment I of* p *consists in the union of the assignments of each class* C *of* Class*, i.e. $I(b) =^{def} I_{\mathtt{C}}(b)$ whenever $b \in dom(I_{\mathtt{C}})$.*

DEFINITION 4. *[Canonical extension] A program assignment I is defined over an expression or method call d if each symbol of $\mathcal{F} \cup$ Class in d belongs to $dom(I)$. Suppose that the assignment I is defined over d, the partial assignment of d w.r.t. I, that we note $I^*(d)$ is the canonical extension of the assignment I defined as follows:*

1. *If $\diamond$ is an attribute or a parameter (in $\mathcal{X} \cup \mathcal{P}$), then $I^*(\diamond) = \square$, with $\square$ a new variable ranging over $\mathbb{R}^+$, s.t. the restriction of I^* to $\mathcal{X} \cup \mathcal{P}$ is an injective function.*
2. *If* C *is a constructor of a class* C $\in$ Class *having n attributes and $e_1, \ldots, e_n$ are expressions then we have $I^*(\texttt{new C}(e_1, \ldots, e_n)) = I(\mathtt{C})(I^*(e_1), \ldots, I^*(e_n))$.*

3. *If* $\mathtt{f} \in \mathcal{F}$ *is a method of arity* m *and* $e, e_1, \ldots, e_m$ *are expressions, then:*
$$I^*(e.\mathtt{f}(e_1, \ldots, e_m)) = I(\mathtt{f})(I^*(e_1), \ldots, I^*(e_m), I^*(e))$$

Notice that the assignment $I^*(d)$ of an expression or method call d with m parameters $x_1, \ldots, x_m$ occurring in a class $\mathtt{C}$ having n attributes $X_1, \ldots, X_n$ denotes a function ϕ from $(\mathbb{R}^+)^{n+m} \to \mathbb{R}^+$ satisfying $\phi(I^*(X_1), \ldots, I^*(X_n), I^*(x_1), \ldots, I^*(x_m)) = I^*(d)$. Throughout the paper, we use the notation $I^*(e)(a_1, \ldots, a_n, b_1, \ldots, b_m)$ to denote $\phi(a_1, \ldots, a_n, b_1, \ldots, b_m)$.

DEFINITION 5. *Let* ***Max-Poly*** $\{\mathbb{R}^+\}$ *be the set of functions defined to be constant functions in* $\mathbb{R}^+$*, projections,* max, $+$, $\times$ *and closed by composition. Given a class with* n *attributes, an assignment* I *is said to be polynomial if for every symbol* b *of* $dom(I)$*,* $I(b)$ *is a function of* ***Max-Poly*** $\{\mathbb{R}^+\}$*.*

DEFINITION 6. *The assignment of a constructor* $\mathtt{C}$ *of arity* n *is* additive *if:*

$$I(\mathtt{C})(\diamond_1, \ldots, \diamond_n) = \sum_{i=1}^{n} \diamond_i + \alpha_{\mathtt{C}}, \text{ where } \alpha_{\mathtt{C}} \geq 1, \qquad \text{if } n > 0$$
$$I(\mathtt{C}) = 0 \qquad \text{if } n = 0$$

If the assignment of each constructor $\mathtt{C} \in \mathtt{Class}$ *is additive then the program assignment is additive.*

LEMMA 7. *Given a program* $\mathtt{p}$ *having an additive assignment* I*, there is a constant* α *such that for each attribute* X *and each store* σ*, the following inequalities are satisfied:*

$$|X\sigma| \leq I^*(X\sigma) \leq \alpha \times |X\sigma|$$

PROOF. Define $\alpha = \max_{\mathtt{C} \in \mathtt{Class}}(\alpha_{\mathtt{C}})$, where $\alpha_{\mathtt{C}}$ is taken to be the constant of definition 6, if $\mathtt{C}$ is of strictly positive arity, and $\alpha_{\mathtt{C}}$ is equal to the constant 0 otherwise. The inequalities follow directly by induction on the size of an object.

3.2 Sup-interpretations

DEFINITION 8. *Given a program* $\mathtt{p}$*, a sup-interpretation of* $\mathtt{p}$ *is an assignment* θ *of* $\mathtt{p}$ *which satisfies:*

1. *The assignment* θ *is weakly monotonic. i.e. for each symbol* $b \in dom(\theta)$*, the function* $\theta(b)$ *satisfies* $\forall \diamond_1, \ldots, \diamond_n, \diamond'_1, \ldots, \diamond'_n \in \mathbb{R}^+, \diamond_i \geq \diamond'_i \Rightarrow \theta(b)(\ldots, \diamond_i, \ldots) \geq \theta(b)(\ldots, \diamond'_i, \ldots)$.
2. *For each object* $o \in \mathtt{Object}$*,* $\theta^*(o) \geq |o|$
3. *For each method* $\mathtt{f} \in dom(\theta)$ *of arity* m*, for each* $o_1, \ldots, o_m \in \mathtt{Objects}$ *and for each store* σ*, if* $\langle X.\mathtt{f}(o_1, \ldots, o_m), \sigma\rangle \downarrow \langle o, \sigma'\rangle$ *then:*
   - $\theta(\mathtt{f})(\theta^*(o_1), \ldots, \theta^*(o_m), \theta^*(X\sigma)) \geq \theta^*(o)$
   - $\theta(\mathtt{f})(\theta^*(o_1), \ldots, \theta^*(o_m), \theta^*(X\sigma)) \geq \theta^*(X\sigma')$

A sup-interpretation is polynomial if it is a polynomial assignment.

Notice that the last condition on methods allows to bound both the sup-interpretation of the output $\theta^*(o)$ and the sup-interpretation of the side effect $\theta^*(X\sigma')$.

Example 3 *Consider the program of example 1. We claim that the partial assignment θ defined by $\theta(\texttt{null}) = 0$, $\theta(\texttt{setTail})(\Box_y, \Box) = \Box_y + \Box$ and $\theta(\texttt{List})(\Box_X, \Box_Y, \Box_W, \Box_Z) = \Box_X + \Box_Y + \Box_W + \Box_Z + 1$ is a sup-interpretation of this program. Indeed, the considered functions are monotonic. Since this assignment is additive, by lemma 7, we obtain that for each list l, $\theta(l) \geq |l|$. Finally, given a store σ such that $\langle X.\texttt{setTail}(o), \sigma\rangle \downarrow \langle v, \sigma'\rangle$ and $X\sigma = \texttt{new List}(h, t, o_W, o_Z)$, for some objects h, t o_W and o_Z, we have that $v = h$ and $\sigma' = \sigma\{X \leftarrow \texttt{new List}(h, o, o_W, o_Z)\}$. Consequently, we check that θ is a sup-interpretation:*

$$\begin{aligned}\theta(\texttt{setTail})(\theta^*(o), \theta^*(\texttt{new List}(h, t, o_W, o_Z))) &\geq \theta^*(o) + \theta^*(\texttt{new List}(h, t, o_W, o_Z))\\ &\geq \theta^*(o) + \theta^*(h) + \theta^*(t) + \theta^*(o_W) + \theta^*(o_Z) + 1\\ &\geq \max(\theta^*(v), \theta^*(X\sigma'))\end{aligned}$$

LEMMA 9. *Given a program p having a sup-interpretation θ defined over $X.\mathtt{f}(e_1, \ldots, e_n)$, for each store σ, if $\langle X.\mathtt{f}(e_1, \ldots, e_n), \sigma\rangle \downarrow \langle o, \sigma'\rangle$, then $\theta^*(X.\mathtt{f}(e_1, \ldots, e_n)\sigma) \geq \max(\theta^*(o), \theta^*(X\sigma'))$.*

PROOF. We show this lemma in two steps. First, we can show easily by structural induction on an expression e that, for each store σ, if $\langle e, \sigma\rangle \downarrow \langle o, \sigma\rangle$ then $\theta^*(e\sigma) = \theta^*(o)$. Second, suppose that $a = X.\mathtt{f}(e_1, \ldots, e_n)$, $\langle a, \sigma\rangle \downarrow \langle o, \sigma'\rangle$ and that, for each $i \in \{1, n\}$ $\langle e_i, \sigma\rangle \downarrow \langle o_i, \sigma\rangle$.

$$\begin{aligned}\theta^*(a\sigma) &\geq \theta^*(X\sigma.\mathtt{f}(e_1\sigma, \ldots, e_n\sigma)) && \text{By definition of } \sigma\\ &\geq \theta(\mathtt{f})(\theta^*(e_1\sigma), \ldots, \theta^*(e_n\sigma), \theta^*(X\sigma)) && \text{By definition of } \theta\\ &\geq \theta(\mathtt{f})(\theta^*(o_1), \ldots, \theta^*(o_n), \theta^*(X\sigma)) && \text{By step 1}\\ &\geq \max(\theta^*(o), \theta^*(o')) && \text{By definition 8}\end{aligned}$$

Example 4 *Consider the linked list class of example 1, a method call $V.\texttt{setTail}(U)$ and a store σ such that $U\sigma = \texttt{new List}(\texttt{null}, \texttt{new List}(\overline{\texttt{null}}), \overline{\texttt{null}})$ and $V\sigma = \texttt{new List}(h, t, \overline{\texttt{null}})$, for some objects h and t. The method call $V.\texttt{setTail}(U)$ updates the tail of the object contained in V to the object contained in U and then returns the head of the object contained in V. Consequently, we obtain that:*

$$\langle V.\texttt{setTail}(U), \sigma\rangle \downarrow \langle h, \sigma\{V \leftarrow \texttt{new List}(h, \texttt{new List}(\texttt{null}, \texttt{new List}(\overline{\texttt{null}}), \overline{\texttt{null}}), \overline{\texttt{null}})\}\rangle$$

Taking the sup-interpretation θ defined in example 3, we check that:

$$\begin{aligned}\theta^*(V.\texttt{setTail}(U)\sigma) &\geq \theta(\texttt{setTail})(\theta^*(U\sigma), \theta^*(V\sigma))\\ &\geq \theta^*(\texttt{new List}(h, t, \overline{\texttt{null}})) + \theta^*(\texttt{new List}(\texttt{null}, \texttt{new List}(\overline{\texttt{null}}), \overline{\texttt{null}}))\\ &\geq \theta^*(h) + \theta^*(t) + 3\\ &\geq \max(\theta^*(h), \theta^*(h) + 3)\\ &\geq \max(\theta^*(h), \theta^*(\texttt{new List}(h, \texttt{new List}(\texttt{null}, \texttt{new List}(\overline{\texttt{null}}), \overline{\texttt{null}}), \overline{\texttt{null}})))\end{aligned}$$

3.3 Weights

The notion of weight allows to control the size of the objects (and a fortiori the number of instantiated objects) during loop iterations. A weight is a partial mapping over commands.

DEFINITION 10. *[Context] A context* $\mathsf{C}[\bullet_1,\ldots,\bullet_n]$ *is a special command defined by the following grammar:*

$$\mathsf{C}[\overline{\bullet}] ::= \texttt{skip} \mid \bullet_1 \mid \ldots \mid \bullet_n \mid X := a \mid X := e \mid \mathsf{C}_1[\overline{\bullet}];\mathsf{C}_2[\overline{\bullet}] \mid \texttt{loop}\ X\{\mathsf{C}_1[\overline{\bullet}]\} \mid \texttt{if}(e)\texttt{then}\{\mathsf{C}_1[\overline{\bullet}]\}\texttt{else}\{\mathsf{C}_2[\overline{\bullet}]\} \mid \texttt{while}\ e\{\mathsf{C}_1[\overline{\bullet}]\}$$

Let $\mathsf{C}[\mathtt{Cm}_1,\ldots,\mathtt{Cm}_n]$ *denote the substitution of each* $\bullet_i$ *by the command* $\mathtt{Cm}_i$ *in* $\mathsf{C}[\bullet_1,\ldots,\bullet_n]$. *A one-hole context is a context having exactly one occurrence of each* $\bullet_i$. *One-hole contexts induce a partial ordering* $\sqsubseteq$ *(resp. strict partial ordering* $\sqsubset$*) over commands defined by* $\mathtt{Cm}_1 \sqsubseteq \mathtt{Cm}_2$ *(resp.* $\mathtt{Cm}_1 \sqsubset \mathtt{Cm}_2$*) if and only if there is a one-hole context* $\mathsf{C}[\bullet]$ *(resp. distinct from* $\bullet$*) such that* $\mathtt{Cm}_2 = \mathsf{C}[\mathtt{Cm}_1]$.

DEFINITION 11. *[Minimal, while and loop commands] A command* Cm *is:*
- a minimal command if there is no context of the shape $\mathsf{C}[\bullet_1,\bullet_2] = \texttt{if}(e)\texttt{then}\{\bullet_1\}\texttt{else}\{\bullet_2\}$ *or* $\mathsf{C}[\bullet_1,\bullet_2] = \bullet_1;\bullet_2$ *such that* $\mathtt{Cm} = \mathsf{C}[\mathtt{Cm}_1,\mathtt{Cm}_2]$, *for some commands* $\mathtt{Cm}_1$ *and* $\mathtt{Cm}_2$.

- a while command if there are a one-hole context $\mathsf{C}[\bullet]$ *and a command* $\mathtt{Cm}_1 = \texttt{while}\ e\{\mathtt{Cm}_2\}$ *such that* $\mathtt{Cm} = \mathsf{C}[\mathtt{Cm}_1]$.

- a loop command if Cm *is not a while command and there are a one-hole context* $\mathsf{C}[\bullet]$ *and a command* $\mathtt{Cm}_1 = \texttt{loop}\ X\{\mathtt{Cm}_2\}$ *such that* $\mathtt{Cm} = \mathsf{C}[\mathtt{Cm}_1]$.

Example 5 *We illustrate the distinct notions introduced above by the following example:*

```
Class main { var X; var Y; var Z;
             Cm1 : X := Y; loop X {while Y { Y = Y.getTail(); }};
             Cm2 : loop X {Cm3 : Z := Z.getTail(); }; }
```

The command $\mathtt{Cm}_1$ *is a while command but neither a minimal command nor a loop command. The command* $\mathtt{Cm}_2$ *is a minimal and loop command. The command* $\mathtt{Cm}_3$ *is only a minimal command.*

DEFINITION 12. *[Weight] Given a program* p *having a main class with* n *attributes, the weight* ω *is a partial mapping which assigns to every minimal and loop command* Cm, *a total function* $\omega_{\mathtt{Cm}}$ *from* $(\mathbb{R}^+)^{n+1}$ *to* $\mathbb{R}^+$ *which satisfies:*

1. $\omega_{\mathtt{Cm}}$ *is weakly monotonic* $\forall i,\ \Box_i \geq \Box_i' \Rightarrow \omega_{\mathtt{Cm}}(\ldots,\Box_i,\ldots) \geq \omega_{\mathtt{Cm}}(\ldots,\Box_i',\ldots)$
2. $\omega_{\mathtt{Cm}}$ *has the subterm property* $\forall i,\ \forall \Box_i \in \mathbb{R}^+\ \omega_{\mathtt{Cm}}(\ldots,\Box_i,\ldots) \geq \Box_i$

A weight ω *is polynomial if each* $\omega_{\mathtt{Cm}}$ *is a function of* ***Max-Poly*** $\{\mathbb{R}^+\}$.

Example 6 *The program of example 5 has three attributes and exactly one minimal and loop command* $\mathtt{Cm}_2$. *Consequently, the mapping* ω *defined by* $\omega_{\mathtt{Cm}_2}(\Box,\Box_X,\Box_Y,\Box_Z) = \Box + \max(\Box_X,\Box_Y,\Box_Z)$ *is a polynomial weight.*

4 Criteria to control resources

4.1 Brotherly criterion

The brotherly criterion gives constraints on weights and sup-interpretations in order to bound the size of the objects computed by the program by some polynomial in the size of the inputs.

DEFINITION 13. *A program having a main class with n attributes $X_1, \ldots, X_n$ is **brotherly** if there are a total, polynomial and additive sup-interpretation θ and a polynomial weight ω such that:*

- *For every minimal and loop command* $\mathtt{Cm}$ *of the main class:*
  - *For every method call a of the shape $X_j.\mathtt{f}(e_1, \ldots, e_m)$ occurring in* $\mathtt{Cm}$*:*

    $$\omega_{\mathtt{Cm}}(\square + 1, \theta(X_1), \ldots, \theta(X_n)) \geq \omega_{\mathtt{Cm}}(\square, \theta(X_1), \ldots, \theta(X_{j-1}), \theta^*(a), \theta(X_{j+1}), \ldots, \theta(X_n))$$

    where $\square$ is a fresh variable.
  - *For every variable assignment $X_i := d \sqsubseteq \mathtt{Cm}$:*

    $$\omega_{\mathtt{Cm}}(\square + 1, \theta(X_1), \ldots, \theta(X_n)) \geq \omega_{\mathtt{Cm}}(\square, \theta(X_1), \ldots, \theta(X_{i-1}), \theta^*(d), \theta(X_{i+1}), \ldots, \theta(X_n))$$

    where $\square$ is a fresh variable.
- *For every minimal and while command* $\mathtt{Cm}$ *of the main class:*
  - *For every variable assignment $X_i := d \sqsubseteq \mathtt{Cm}$, $\max(\theta(X_1), \ldots, \theta(X_n)) \geq \theta^*(d)$*

Intuitively, the first condition on loop commands ensures that the size of the objects held by the attributes remains polynomially bounded. The fresh variable $\square$ can be seen as a temporal factor which takes into account the number of iterations allowed in a loop. Such a number is polynomially bounded by the size of the objects held by the attributes in the store. The second condition on while commands ensures that a computation is non-size-increasing since we have no piece of information about the termination of while commands.

THEOREM 14. *Given a brotherly program* $\mathtt{p}$ *of main class* $\mathtt{Class\ main}\{A\ \mathtt{Cm}\}$*, having n attributes $X_1, \ldots, X_n$, there exists a polynomial P such that for any store σ and any command $\mathtt{Cm}_1 \sqsubseteq \mathtt{Cm}$ if $\langle \mathtt{Cm}_1, \sigma \rangle \downarrow \langle \sigma' \rangle$ then $P(|X_1\sigma|, \ldots, |X_n\sigma|) \geq \max_{i=1..n}(|X_i\sigma'|)$.*

PROOF. We can build the polynomial P by structural induction on commands.

Example 7 *Consider the following program*

$$\mathtt{Class\ main}\{\mathtt{Var}\ U; \mathtt{Var}\ V; \mathtt{Var}\ T; \mathtt{loop}\ T\{U := V.\mathtt{reverse}()\}; U.\mathtt{setTail}(T)\}$$

$\mathtt{Cm} = \mathtt{loop}\ T\{U := V.\mathtt{reverse}()\}$ *is the only minimal and loop command. Consequently, we have to find a polynomial weight ω and a polynomial and additive sup-interpretation θ such that:*

$$\omega_{\mathtt{Cm}}(\square + 1, \theta(U), \theta(V), \theta(T)) \geq \omega_{\mathtt{Cm}}(\square, \theta(U), \theta(V.\mathtt{reverse}()), \theta(T))$$
$$\omega_{\mathtt{Cm}}(\square + 1, \theta(U), \theta(V), \theta(T)) \geq \omega_{\mathtt{Cm}}(\square, V.\mathtt{reverse}(), \theta(V), \theta(T))$$

in order to check the brotherly criterion. We let the reader check that the assignment θ defined by $\theta(\mathtt{reverse})(\square) = \square$ together with the assignment of example 3 defines a total (i.e. defined for every method symbol), polynomial and additive sup-interpretation.

Moreover, taking $\omega_{\mathtt{Cm}}(\square, \square_U, \square_V, \square_T) = \square + \square_U + \square_V + \square_T$, we obtain that this program is brotherly by checking that the above inequalities are satisfied.

4.2 Heuristics for method sup-interpretation synthesis

The previous criterion is very powerful. However, before being applied, it requires to know the sup-interpretation of the methods. Consequently, an interesting issue is to give some criterion on a method of some class in order to build its sup-interpretation.

DEFINITION 15. *[Method weight] The weight of a method* D *having* m *parameters and belonging to a class* C *having* n *attributes is a monotonic and subterm function* $\omega_{\mathtt{D}}$ *from* $(\mathbb{R}^+)^{m+2}$ *to* $\mathbb{R}^+$. *A weight* $\omega_{\mathtt{D}}$ *is polynomial if it belongs to* ***Max-Poly*** $\{\mathbb{R}^+\}$.

DEFINITION 16. *Given a class* C *with* n *attributes* $X_1, \ldots, X_n$, *a method* D *of* C *of the shape* $\mathtt{f}(x_1, \ldots, x_m)\,\{\mathtt{Cm}\,;\,\mathtt{return}\ X;\}$ *is* ***brotherly*** *if there is a polynomial and additive sup-interpretation* θ *s.t.:*

1. *If* Cm *is a while command then for every variable assignment* $X_i := d \sqsubseteq \mathtt{Cm}$, *we have:*
$$\max(\theta(x_1), \ldots, \theta(x_m), \theta(X_1), \ldots, \theta(X_n)) \geq \theta^*(d)$$
2. *Else there is a polynomial method weight* $\omega_{\mathtt{D}}$ *such that:*
   - *For every method call* $a = X_j.\mathtt{f}(e_1, \ldots, e_m)$ *occurring in* Cm:
   $\omega_{\mathtt{D}}(\Box + 1, \theta(x_1), \ldots, \theta(x_m), \sum_{k=1}^{n} \theta(X_k)) \geq \omega_{\mathtt{D}}(\Box, \theta(x_1), \ldots, \theta(x_m), \sum_{k\neq j, k=1}^{n} \theta(X_k) + \theta^*(a))$
   - *For every variable assignment* $X_i := d \sqsubseteq \mathtt{Cm}$, *we have:*
   $\omega_{\mathtt{D}}(\Box + 1, \theta(x_1), \ldots, \theta(x_m), \sum_{k=1}^{n} \theta(X_k)) \geq \omega_{\mathtt{D}}(\Box, \theta(x_1), \ldots, \theta(x_m), \sum_{k\neq i, k=1}^{n} \theta(X_k) + \theta^*(d))$

   where $\Box$ *is a fresh variable.*

THEOREM 17. *Given a program* p, *a class* C *having* n *attributes* $X_1, \ldots, X_n$ *and a sup-interpretation* θ *such that the method* $\mathtt{D} = \mathtt{f}(x_1, \ldots, x_m)\,\{\mathtt{Cm}\,;\,\mathtt{return}\ X_i;\}$ *of* C *is brotherly, we have:*

- *If* Cm *is a while command then* $\theta(\mathtt{f})(\Box_1, \ldots, \Box_m, \Box) =_{def} \max(\Box_1, \ldots, \Box_m, \Box)$ *is a sup-interpretation of* f.
- *Else, if* R *is a polynomial upper bound on the number of variable assignments occurring during the execution of* Cm *then* $\theta(\mathtt{f})(\Box_1, \ldots, \Box_m, \Box) =_{def} \omega_{\mathtt{D}}(R(\Box), \Box_1, \ldots, \Box_m, \Box)$ *is a sup-interpretation of* f.

PROOF. The proof is similar to the proof of theorem 14. The only distinction is that parameters can appear in the commands.

Remarks: Since a command $\mathtt{loop}\ X\,\{\mathtt{Cm}\}$ cannot write in the attribute X, the polynomial R can be computed by static analysis. Consequently, if we manage to check the brotherly criterion for a given method then we obtain a sup-interpretation of the method.

Example 8 *Consider the method* setTail *of example 1. The command* $Y := y$ *is not a while command. Consequently, we have to find a polynomial weight* $\omega_{\mathtt{D}} : (\mathbb{R}^+)^3 \to \mathbb{R}^+$ *satisfying:*

$$\omega_{\mathtt{D}}(\Box + 1, \theta(y), \sum_{K\in\{X,Y,W,Z\}} \theta(K)) \geq \omega_{\mathtt{D}}(\Box, \theta(y), \sum_{K\in\{X,W,Z\}} \theta(K) + \theta(y))$$

This inequality is satisfied by taking $\omega_{\mathtt{D}}(\Box, \Box_y, \Box') = \Box \times \Box_y + \Box'$. *We know that there is exactly one variable assignment in the execution of such a method (i.e.* $R = 1$*) and, by theorem 17,* $\theta(\mathtt{setTail})(\Box_y, \Box) = 1 \times \Box_y + \Box = \Box_y + \Box$ *is a sup-interpretation of* setTail.

5 Conclusion and perspectives

We have suggested a high level approach for analyzing the complexity of object oriented programs. This static analysis is performed using semantics interpretations and provides upper bounds on the number of object creations during the execution of a given program.

Consequently, this study is complementary to the works of [9, 17, 3] using abstract interpretations which guarantee that there is no buffer overflow in the memory locations of a given program. Our study allows to perform a resource analysis of a huge number of programs. Some improvements can obviously be performed in several directions: Currently, a while iteration cannot compute more than a maximum function. A more precise analysis of while iterations should be performed using the work of [19] on the termination of imperative while programs. The criterion for sup-interpretation synthesis has no sense when considering recursive (and a fortiori mutual recursive) methods (Since we have to previously know the sup-interpretation of the considered symbol). As a consequence, we have to develop a criterion in the general recursive case, even if side effects make such a study difficult.

References

[1] E. Albert, P. Arenas, S. Genaim, G. Puebla, and D. Zanardini. Cost Analysis of Java Bytecode. In *Programming Languages and Systems*, volume 4421 of *LNCS*, pages 157–172. Springer, 2007.

[2] E. Albert, P. Arenas, S. Genaim, G. Puebla, and D. Zanardini. Removing useless variables in cost analysis of Java bytecode. In *Proceedings of the 2008 ACM symposium on Applied computing*, pages 368–375. ACM New York, NY, USA, 2008.

[3] X. Allamigeon and C. Hymans. Static Analysis by Abstract Interpretation: Application to the Detection of Heap Overflows. *Journal in Computer Virology*, 4, 2007.

[4] R. Amadio, S. Coupet-Grimal, S. Dal-Zilio, and L. Jakubiec. A functional scenario for bytecode verification of resource bounds. In *CSL*, volume 3210 of *LNCS*, pages 265–279. Springer, 2004.

[5] R. Amadio and S. Dal-Zilio. Resource control for synchronous cooperative threads. In *CONCUR*, volume 3170 of *LNCS*, pages 68–82. Springer, 2004.

[6] R.J. Anderson. *Security engineering: a guide to building dependable distributed systems*. John Wiley & Sons, New York, 2001.

[7] G. Bonfante, J.-Y. Marion, and R. Péchoux. A characterization of alternating log time by first order functional programs. In *LPAR 2006*, volume 4246 of *LNAI*, pages 90–104. Springer, 2006.

[8] G. Bonfante, J.Y. Marion, and J.Y. Moyen. Quasi-interpretations, a way to control resources. *Theoretical Computer Science*. Accepted.

[9] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. *POPL'77*, pages 238–252, 1977.

[10] S. Dal-Zilio and R. Gascon. Resource bound certification for a tail-recursive virtual machine. In *APLAS 2005*, volume 3780 of *LNCS*, pages 247–263. Springer, 2005.

[11] S. Drossopoulou and S. Eisenbach. Describing the semantics of Java and proving type soundness. LNCS, pages 41–82. Springer, 1999.

[12] A. Igarashi, B.C. Pierce, and P. Wadler. Featherweight Java: A Minimal Core Calculus for Java and GJ. *ACM Transactions on Programming Languages and Systems*, 23(3):396–450, 2001.

[13] L. Kristiansen and N.D. Jones. The flow of data and the complexity of algorithms. *New Computational Paradigms*, 3526:263–274, 2006.

[14] J.-Y. Marion and R. Péchoux. Resource analysis by sup-interpretation. In *FLOPS 2006*, volume 3945 of *LNCS*, pages 163–176, 2006.

[15] J.-Y. Marion and R. Péchoux. Sup-interpretations, a semantic method for static analysis of program resources. *ACM Transactions on Computational Logic (TOCL)*, 2008. Accepted.

[16] J.Y. Marion and R. Péchoux. A Characterization of NCk by First Order Functional Programs. In *TAMC*, volume 4978 of *LNCS*, pages 136–147. Springer, 2008.

[17] A. Miné. Field-sensitive value analysis of embedded C programs with union types and pointer arithmetics. In *ACM SIGPLAN/SIGBED Conf. on Languages, Compilers, and Tools for Embedded Systems (LCTES'06)*, pages 54–63, Ottawa, Ontario, Canada, June 2006. ACM Press. `http://www.di.ens.fr/~mine/publi/article-mine-lctes06.pdf`.

[18] K.H. Niggl and H. Wunderlich. Certifying Polynomial Time and Linear/Polynomial Space for Imperative Programs. *SIAM Journal on Computing*, 35:1122, 2006.

[19] A. Podelski and A. Rybalchenko. Transition predicate abstraction and fair termination. In *POPL*, pages 132–144. ACM, 2005.

Foundations of Software Technology and Theoretical Computer Science (Bangalore) 2008.
Editors: R. Hariharan, M. Mukund, V. Vinay; pp 328-339

A Hierarchy of Semantics for Non-deterministic Term Rewriting Systems*

Juan Rodríguez-Hortalá
Departamento de Sistemas Informáticos y Computación
Universidad Complutense de Madrid, Spain
juanrh@fdi.ucm.es

ABSTRACT. Formalisms involving some degree of nondeterminism are frequent in computer science. In particular, various programming or specification languages are based on term rewriting systems where confluence is not required. In this paper we examine three concrete possible semantics for non-determinism that can be assigned to those programs. Two of them –call-time choice and run-time choice– are quite well-known, while the third one –plural semantics– is investigated for the first time in the context of term rewriting based programming languages. We investigate some basic intrinsic properties of the semantics and establish some relationships between them: we show that the three semantics form a hierarchy in the sense of set inclusion, and we prove that call-time choice and plural semantics enjoy a remarkable compositionality property that fails for run-time choice; finally, we show how to express plural semantics within run-time choice by means of a program transformation, for which we prove its adequacy.

1 Introduction

Term rewriting systems (*TRS's*) [4] have a long tradition as a suitable basic formalism to address a wide range of tasks in computer science, in particular, many specification languages [5, 7], theorem provers [21, 6] and programming languages are based on TRS's. For instance, the syntax and theory of TRS's was the basis of the first formulations of *functional logic programming* (*FLP*) through the idea of narrowing [9]. On the other hand, non-determinism is an expressive feature that has been used for a long time in system specification (e.g., non-deterministic Turing machines or automata) or for programming (the constructions of McCarthy and Dijkstra are classical examples). One of the appeals of term rewriting is its elegant way to express non-determinism through the use of a non-confluent TRS, obtaining a clean and high level representation of complex systems. In the field of FLP, non-confluent TRS's are used as programs to support non-strict non-deterministic functions, which are one of the most distinctive features of the paradigm [8, 3]. Those TRS's follow the constructor discipline also, corresponding to a value-based semantic view, in which the purpose of computations is to produce values made of constructors.

Therefore non-confluent constructor-based TRS's can be used as a common syntactic framework for FLP and rewriting. The set of rewrite rules constitutes a program and so we also call them *program rules*. Nevertheless the behaviour of current implementations of FLP

*This work has been partially supported by the Spanish projects Merit-Forms-UCM (TIN2005-09207-C03-03), Promesas-CAM (S-0505/TIC/0407) and FAST-STAMP (TIN2008-06622-C03-01/TIN).

FSTTCS 2008
IARCS Annual Conference on
Foundations of Software Technology and Theoretical Computer Science
http://drops.dagstuhl.de/opus/volltexte/2008/1764

and rewriting differ substantially, because the introduction of non-determinism in a functional setting gives rise to a variety of semantic decisions, that were explored in [20]. There the different language variants that result after adding non-determinism to a basic functional language were expounded, structuring the comparison as a choice among different options over several dimensions: strict/non-strict functions, angelic/demonic/erratic non-deterministic choices and *singular/plural semantics* for parameter passing. In the present paper we assume non-strict angelic non-determinism, and we are concerned about the last dimension only. The key difference is that under a singular semantics, in the substitutions used to instantiate the program rules for function application, the variables of the program rules should range over single objects of the set of values considered; in a plural semantics those range over sets of objects. This has been traditionally identified with the distinction between *call-time choice* and *run-time choice* [11] parameter passing mechanisms. Under call-time choice a value for each argument is computed before performing parameter passing, this corresponds to call-by-value in a strict setting and to call-by-need in a non-strict setting, in which a partial value instead of a total value is computed. On the other hand, run-time-choice corresponds to call-by-name, each argument is copied without any evaluation and so the different copies of any argument may evolve in different ways afterwards. Thus, traditionally it has been considered that call-time choice parameter passing inducts a singular semantics while run-time choice inducts a plural semantics.

EXAMPLE 1. *Consider the TRS* $\mathcal{P} = \{f(c(X)) \to d(X,X), X\ ?\ Y \to X, X\ ?\ Y \to Y\}$. *With call-time choice/singular semantics to compute a value for the term* $f(c(0?1))$ *we must first compute a (partial) value for* $c(0?1)$*, and then we may continue the computation with* $f(c(0))$ *or* $f(c(1))$ *which yield* $d(0,0)$ *or* $d(1,1)$*. Note that* $d(0,1)$ *and* $d(1,0)$ *are not correct values for* $f(c(0?1))$ *in that setting.*
On the other hand with run-time choice/plural semantics to evaluate the term $f(c(0?1))$*:*

- *Under the run-time choice point of view, the step* $f(c(0?1)) \to d(0?1,0?1)$ *is sound, hence not only* $d(0,0)$ *and* $d(1,1)$ *but also* $d(0,1)$ *and* $d(1,0)$ *are valid values for* $f(c(0?1))$*.*
- *Under the plural semantics point of view, we consider the set* $\{c(0), c(1)\}$ *which is a subset of the set of values for* $c(0?1)$ *in which every element matches the argument pattern* $c(X)$*. Therefore the set* $\{0,1\}$ *can be used for parameter passing obtaining a kind of "set expression"* $d(\{0,1\},\{0,1\})$*, which evaluation yields the values* $d(0,0)$*,* $d(1,1)$*,* $d(0,1)$ *and* $d(1,0)$*.*

In general, call-time choice/singular semantics produces less results than run-time choice/plural semantics.

A standard formulation for call-time choice[†] in FLP is the *CRWL*[‡] logic [8], which is implemented by current FLP languages like Toy [15] or Curry [10]; traditional term rewriting may be considered the standard semantics for run-time choice[§], and is the basis for the semantics of languages like Maude [5], but has been rarely [1] thought as a valuable global alternative to call-time choice for the value-based view of FLP. However, there might be

[†] In fact angelic non-strict call-time choice.
[‡] **C**onstructor-based **Re**-**W**riting **L**ogic.
[§] In fact angelic non-strict run-time choice.

parts in a program or individual functions for which run-time choice could be a better option, and therefore it would be convenient to have both possibilities (run-time/call-time) at programmer's disposal [13]. Nevertheless the use of an operational notion like term rewriting as the semantic basis of a FLP language can lead us to confusing situations, not very compatible with the value-based semantic view that we wanted for the constructor-based TRS's used in FLP.

EXAMPLE 2. *Starting with the TRS of Example 1 we want to evaluate the expression* $f(c(0) ? c(1))$ *with run-time choice/plural semantics:*

- *Under the run-time choice point of view, that is, using term rewriting, the evaluation of the subexpression* $c(0)?c(1)$ *is needed in order to get an expression that matches the left hand side* $f(c(X))$*. Hence the derivations* $f(c(0)?c(1)) \to f(c(0)) \to d(0,0)$ *and* $f(c(0)?c(1)) \to f(c(1)) \to d(1,1)$ *are sound and compute the values* $d(0,0)$ *and* $d(1,1)$*, but neither* $d(0,1)$ *nor* $d(1,0)$ *are correct values for* $f(c(0)?c(1))$*.*
- *Under the plural semantics point of view, we consider the set* $\{c(0), c(1)\}$ *which is a subset of the set of values for* $c(0)?c(1)$ *in which every element matches the argument pattern* $c(X)$*. Therefore the set* $\{0,1\}$ *can be used for parameter passing obtaining a kind of "set expression"* $d(\{0,1\},\{0,1\})$ *that yields the values* $d(0,0)$*,* $d(1,1)$*,* $d(0,1)$ *and* $d(1,0)$*.*

Which of these is the more suitable perspective for FLP?

This problem did not appear in [20] because no pattern matching was present, nor in [11] because only call-time choice was adopted (and this conflict does not appear between the call-time choice and the singular semantics views). Choosing the run-time choice perspective of term rewriting has some unpleasant consequences. First of all the expression $f(c(0?1))$ has more values than the expression $f(c(0)?c(1))$, even when the only difference between them is the subexpressions $c(0?1)$ and $c(0)?c(1)$, which have the same values both in call-time choice, run-time choice and plural semantics. This is pretty incompatible with the value-based semantic view we are looking for in FLP. And this has to do with the loss of some desirable properties, present in *CRWL*, when switching to run-time choice. We will see how plural semantics recovers those properties, which are very useful for reasoning about computations. Furthermore it allows natural encodings of some programs that need to do some collecting work, as we will see later (Example 8). Hence we claim that the plural semantics perspective is more suitable for a value-based programming language.

The rest of the paper is organized as follows. Section 2 contains some technical preliminaries and notations about *CRWL* and term rewriting systems. In Section 3 we introduce π*CRWL*, a variation of *CRWL* to express plural semantics, and present some of its properties. In Section 4 we discuss about the different properties of these semantics and prove the inclusion chain *CRWL* $\subseteq$ rewriting $\subseteq$ π*CRWL*, that constitutes a hierarchy of semantics for non-determinism. Section 5 recalls that no straight simulation of *CRWL* in term rewriting can be done by a program transformation, and vice versa, and shows a novel transformation to simulate π*CRWL* using term rewriting. Finally Section 6 summarizes some conclusions and future work. Fully detailed proofs, including some auxiliary results, can be found in [19].

2 Preliminaries

2.1 Constructor based term rewriting systems

We consider a first order signature $\Sigma = CS \cup FS$, where CS and FS are two disjoint set of *constructor* and defined *function* symbols respectively, all them with associated arity. We write CS^n (FS^n resp.) for the set of constructor (function) symbols of arity n. We write $c, d, \ldots$ for constructors, $f, g, \ldots$ for functions and $X, Y, \ldots$ for variables of a numerable set $\mathcal{V}$. The notation $\bar{o}$ stands for tuples of any kind of syntactic objects. Given a set $\mathcal{A}$ we denote by $\mathcal{A}^*$ the set of finite sequences of elements of that set. For any sequence $a_1 \ldots a_n \in \mathcal{A}^*$ and function $f : \mathcal{A} \to \{true, false\}$, by $a_1 \ldots a_n \mid f$ we denote the sequence constructed taking in order every element from $a_1 \ldots a_n$ for which f holds.

The set Exp of *expressions* is defined as $Exp \ni e ::= X \mid h(e_1, \ldots, e_n)$, where $X \in \mathcal{V}$, $h \in CS^n \cup FS^n$ and $e_1, \ldots, e_n \in Exp$. The set $CTerm$ of *constructed terms* (or *c-terms*) is defined like Exp, but with h restricted to CS^n (so $CTerm \subseteq Exp$). The intended meaning is that Exp stands for evaluable expressions, i.e., expressions that can contain function symbols, while $CTerm$ stands for data terms representing **values**. We will write $e, e', \ldots$ for expressions and $t, s, \ldots$ for c-terms. The set of variables occurring in an expression e will be denoted as $var(e)$. We will frequently use *one-hole contexts*, defined as $Cntxt \ni \mathcal{C} ::= [\,] \mid h(e_1, \ldots, \mathcal{C}, \ldots, e_n)$, with $h \in CS^n \cup FS^n$. The application of a context $\mathcal{C}$ to an expression e, written by $\mathcal{C}[e]$, is defined inductively as $[\,][e] = e$ and $h(e_1, \ldots, \mathcal{C}, \ldots, e_n)[e] = h(e_1, \ldots, \mathcal{C}[e], \ldots, e_n)$.

Substitutions $\theta \in Subst$ are finite mappings $\theta : \mathcal{V} \longrightarrow Exp$, extending naturally to $\theta : Exp \longrightarrow Exp$. We write ϵ for the identity (or empty) substitution. We write $e\theta$ for the application of θ to e, and $\theta\theta'$ for the composition, defined by $X(\theta\theta') = (X\theta)\theta'$. The domain and range of θ are defined as $dom(\theta) = \{X \in \mathcal{V} \mid X\theta \neq X\}$ and $vran(\theta) = \bigcup_{X \in dom(\theta)} var(X\theta)$. If $dom(\theta_0) \cap dom(\theta_1) = \emptyset$, their disjoint union $\theta_0 \uplus \theta_1$ is defined by $(\theta_0 \uplus \theta_1)(X) = \theta_i(X)$, if $X \in dom(\theta_i)$ for some θ_i; $(\theta_0 \uplus \theta_1)(X) = X$ otherwise. Given $W \subseteq \mathcal{V}$ we write $\theta|_W$ for the restriction of θ to W, and $\theta|_{\setminus D}$ is a shortcut for $\theta|_{(\mathcal{V} \setminus D)}$. We will sometimes write $\theta = \sigma[W]$ instead of $\theta|_W = \sigma|_W$. *C-substitutions* $\theta \in CSubst$ verify that $X\theta \in CTerm$ for all $X \in dom(\theta)$.

A *constructor-based term rewriting system* $\mathcal{P}$ (CS) is a set of c-rewrite rules of the form $f(\bar{t}) \to r$ where $f \in FS^n$, $e \in Exp$ and $\bar{t}$ is a linear n-tuple of c-terms, where linearity means that variables occur only once in $\bar{t}$. In the present work we restrict ourselves to CS's not containing *extra variables*, i.e., CS's for which $var(r) \subseteq var(f(\bar{t}))$ holds for any rewrite rule; the extension of this work to rules with extra variables is a subject of future work. We assume that every CS $\mathcal{P}$ contains the rules $\{X \,?\, Y \to X, X \,?\, Y \to Y, if\ true\ then\ X \to X\}$, defining the behaviour of $\_?\_ \in FS^2$, $if\_then\_ \in FS^2$, both used in mixfix mode, and that those are the only rules for that function symbols. For the sake of conciseness we will often omit these rules when presenting a CS.

Given a TRS $\mathcal{P}$, its associated *rewrite relation* $\to_{\mathcal{P}}$ is defined as: $\mathcal{C}[l\sigma] \to_{\mathcal{P}} \mathcal{C}[r\sigma]$ for any context $\mathcal{C}$, rule $l \to r \in \mathcal{P}$ and $\sigma \in Subst$. We write $\overset{*}{\to}_{\mathcal{P}}$ for the reflexive and transitive closure of the relation $\to_{\mathcal{P}}$. In the following, we will usually omit the reference to $\mathcal{P}$ or denote it by $\mathcal{P} \vdash e \to e'$ and $\mathcal{P} \vdash e \to^* e'$.

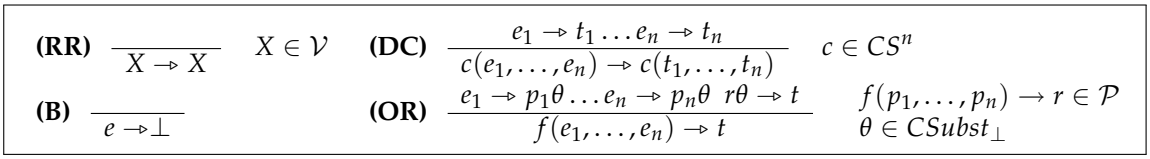


Figure 1: Rules of *CRWL*

2.2 The CRWL framework

In the *CRWL* framework [8], programs are CS's, also called *CRWL-programs* (or simply 'programs') from now on. To deal with non-strictness at the semantic level, we enlarge Σ with a new constant constructor symbol $\bot$. The sets $Exp_\bot$, $CTerm_\bot$, $Subst_\bot$, $CSubst_\bot$ of partial expressions, etc., are defined naturally. Notice that $\bot$ does not appear in programs. Partial expressions are ordered by the *approximation* ordering $\sqsubseteq$ defined as the least partial ordering satisfying $\bot \sqsubseteq e$ and $e \sqsubseteq e' \Rightarrow \mathcal{C}[e] \sqsubseteq \mathcal{C}[e']$ for all $e, e' \in Exp_\bot, \mathcal{C} \in Cntxt$. This partial ordering can be extended to substitutions: given $\theta, \sigma \in Subst_\bot$ we say $\theta \sqsubseteq \sigma$ if $X\theta \sqsubseteq X\sigma$ for all $X \in \mathcal{V}$.

The semantics of a program $\mathcal{P}$ is determined in *CRWL* by means of a proof calculus able to derive reduction statements of the form $e \twoheadrightarrow t$, with $e \in Exp_\bot$ and $t \in CTerm_\bot$, meaning informally that t is (or approximates to) a *possible value* of e, obtained by iterated reduction of e using $\mathcal{P}$ under call-time choice. The *CRWL*-proof calculus is presented in Figure 1. Rule **B** (bottom) allows any expression to be undefined or not evaluated (non-strict semantics). Rule **OR** (outer reduction) expresses that to evaluate a function call we must choose a compatible program rule, perform parameter passing (by means of a $CSubst_\bot$ θ) and then reduce the right-hand side. The use of partial c-substitutions in OR is essential to express call-time choice, as only single partial values are used for parameter passing.

We write $\mathcal{P} \vdash_{CRWL} e \twoheadrightarrow t$ to express that $e \twoheadrightarrow t$ is derivable in the *CRWL*-calculus using the program $\mathcal{P}$. Given a program $\mathcal{P}$, the *CRWL-denotation* of an expression $e \in Exp_\bot$ is defined as $[\![e]\!]^{sg}_{\mathcal{P}} = \{t \in CTerm_\bot \mid \mathcal{P} \vdash_{CRWL} e \twoheadrightarrow t\}$. In the following, we will usually omit the reference to $\mathcal{P}$.

3 π*CRWL*: a plural semantics for FLP

The new calculus $\pi CRWL$ is defined by modifying the rules of *CRWL* to consider sets of partial values for parameter passing instead of single partial values: hence, only the rule OR should be modified. To avoid the need to extend the syntax with new constructions to represent those "set expressions" that we talked about in the introduction, we will exploit the fact that $[\![e_1 \; ? \; e_2]\!] = [\![e_1]\!] \cup [\![e_2]\!]$. Therefore the substitutions used for parameter passing will map variables to "disjunctions of values". We define the set $CSubst^?_\bot = \{\theta \in Subst_\bot \mid \forall X \in dom(\theta), \theta(X) = t_1 \; ? \; \ldots \; ? \; t_n$ such that $t_1, \ldots, t_n \in CTerm_\bot, n > 0\}$, for which $CSubst_\bot \subseteq CSubst^?_\bot \subseteq Subst_\bot$ obviously holds. The operator $? : CSubst^*_\bot \to CSubst^?_\bot$ constructs the $CSubst^?_\bot$ corresponding to a non empty sequence of $CSubst_\bot$, and is defined as $?(\theta_1 \ldots \theta_n)(X) = X$ if $X \notin \bigcup_{i \in \{1,\ldots,n\}} dom(\theta_i)$; $?(\theta_1 \ldots \theta_n)(X) = \rho_1(X) \; ? \; \ldots \; ? \; \rho_m(X)$, where $\rho_1 \ldots \rho_m = \theta_1 \ldots \theta_n \mid \lambda\theta.(X \in dom(\theta))$, otherwise. Then $dom(?(\theta_1 \ldots \theta_n)) = \bigcup_i dom(\theta_i)$. This

$$\textbf{(RR)}\ \frac{}{X \twoheadrightarrow X}\quad X \in \mathcal{V} \qquad \textbf{(DC)}\ \frac{e_1 \twoheadrightarrow t_1 \ldots e_n \twoheadrightarrow t_n}{c(e_1,\ldots,e_n) \twoheadrightarrow c(t_1,\ldots,t_n)}\quad c \in CS^n$$

$$\textbf{(B)}\ \frac{}{e \twoheadrightarrow \bot} \qquad \textbf{(POR)}\ \frac{\begin{array}{ccc} e_1 \twoheadrightarrow p_1\theta_{11} & & e_n \twoheadrightarrow p_n\theta_{n1} \\ \ldots & \ldots & \ldots \\ e_1 \twoheadrightarrow p_1\theta_{1m_1} & & e_n \twoheadrightarrow p_n\theta_{nm_n} \quad r\theta \twoheadrightarrow t \end{array}}{f(e_1,\ldots,e_n) \twoheadrightarrow t}$$

$$(f(\overline{p}) \to r) \in \mathcal{P}, \theta = ?\{\theta_{11},\ldots,\theta_{1m_1}\} \uplus \ldots \uplus ?\{\theta_{n1},\ldots,\theta_{nm_n}\}$$
$$\forall i,j\ \theta_{ij} \in CSubst_\bot \wedge dom(\theta_{ij}) = var(p_i), \forall i\ m_i > 0$$

Figure 2: Rules of $\pi CRWL$

operator is overloaded to handle non empty sets $\Theta \subseteq CSubst_\bot$ as $?\Theta = ?(\theta_1 \ldots \theta_n)$ where the sequence $\theta_1 \ldots \theta_n$ corresponds to an arbitrary reordering of the elements of Θ.

The $\pi CRWL$-proof calculus is presented in Figure 2. The only difference with the calculus in Figure 1 is that the rule OR has been replaced by **POR** (plural outer reduction), in which we may compute more that one partial value for each argument, and then use a substitution from $CSubst^?_\bot$ instead of $CSubst_\bot$ for parameter passing, achieving a plural semantics¶. This calculus derives reduction statements of the form $\mathcal{P} \vdash_{\pi CRWL} e \twoheadrightarrow t$ that express that t is (or approximates to) a possible value for e in this semantics, under the program $\mathcal{P}$. The *$\pi CRWL$-denotation* of an expression $e \in Exp_\bot$ under a program $\mathcal{P}$ in $\pi CRWL$ is defined as $[\![e]\!]^{pl}_{\mathcal{P}} = \{t \in CTerm_\bot \mid \mathcal{P} \vdash_{\pi CRWL} e \twoheadrightarrow t\}$.

EXAMPLE 3. *Consider the program of Example 1, that is $\mathcal{P} = \{f(c(X)) \to d(X,X), X\,?\,Y \to X, X\,?\,Y \to Y\}$. The following is a $\pi CRWL$-proof for the statement $f(c(0)?c(1)) \twoheadrightarrow d(0,1)$ (some steps have been omitted for the sake of conciseness):*

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{0 \twoheadrightarrow 0}\,DC}{c(0) \twoheadrightarrow c(0)}\,DC \quad \dfrac{}{c(1) \twoheadrightarrow \bot}\,B}{c(0)?c(1) \twoheadrightarrow c(0)}\,POR \quad c(0)?c(1) \twoheadrightarrow c(1) \quad \dfrac{0?1 \twoheadrightarrow 0 \quad 0?1 \twoheadrightarrow 1}{d(0?1,0?1) \twoheadrightarrow d(0,1)}\,DC}{f(c(0)?c(1)) \twoheadrightarrow d(0,1)}\,POR$$

$\pi CRWL$ enjoys some nice properties, like the following monotonicity property, where for any proof we define its *size* as the number of applications of rules of the calculus.

LEMMA 4. *For any CRWL-program, $e, e' \in Exp_\bot$, $t, t' \in CTerm_\bot$ if $e \sqsubseteq e'$ and $t' \sqsubseteq t$ then $\mathcal{P} \vdash_{\pi CRWL} e \twoheadrightarrow t$ implies $\mathcal{P} \vdash_{\pi CRWL} e' \twoheadrightarrow t'$ with a proof of the same size or smaller.*

One of the most important properties is its compositionality, a property very close to the DET-additivity property for algebraic specifications of [11]:

THEOREM 5. *For any CRWL-program, $\mathcal{C} \in Contx$ and $e \in Exp_\bot$, $[\![\mathcal{C}[e]]\!]^{pl} = \bigcup_{\{t_1,\ldots,t_n\} \subseteq [\![e]\!]^{pl}} [\![\mathcal{C}[t_1\ ?\ \ldots\ ?\ t_n]]\!]^{pl}$, for any arrangement of the elements of $\{t_1,\ldots,t_n\}$ in $t_1\ ?\ \ldots\ ?\ t_n$.*

The proof for that theorem is based upon the commutativity, associativity of ?, and the idempotence of its partial application (see [19]). With Theorem 5 at hand is very easy to prove the following distributivity property for $\pi CRWL$, also known as the *bubbling* operational rule [2]:

¶In fact angelic non-strict plural non-determinism.

THEOREM 6. *[Correctness of bubbling] For any CRWL-program, $\mathcal{C} \in Contx$ and $e_1, e_2 \in Exp_\perp$, $[\![\mathcal{C}[e_1 \,?\, e_2]]\!]^{pl} = [\![\mathcal{C}[e_1] \,?\, \mathcal{C}[e_2]]\!]^{pl}$.*

$\pi CRWL$ also has some monotonicity properties related to substitutions. We define the preorder $\sqsubseteq_\pi$ over $CSubst^?_\perp$ by $\theta \sqsubseteq_\pi \theta'$ iff $\forall X \in \mathcal{V}$, given $\theta(X) = t_1 \,?\, \ldots \,?\, t_n$ and $\theta(X) = t'_1 \,?\, \ldots \,?\, t'_m$ then $\forall t \in \{t_1, \ldots, t_n\} \exists t' \in \{t'_1, \ldots, t'_m\}$ such that $t \sqsubseteq t'$; and the preorder $\trianglelefteq$ over $Subst_\perp$ by $\sigma \trianglelefteq \sigma'$ iff $\forall X \in \mathcal{V}$, $[\![\sigma(X)]\!]^{pl} \subseteq [\![\sigma'(X)]\!]^{pl}$.

LEMMA 7. *For any CRWL-program, $e \in Exp_\perp$, $t \in CTerm_\perp$, $\sigma, \sigma' \in Subst_\perp$, $\theta, \theta' \in CSubst^?_\perp$:*

1. ***Strong monotonicity of $Subst_\perp$:*** *If $\forall X \in \mathcal{V}, s \in CTerm_\perp$ given $\mathcal{P} \vdash_{\pi CRWL} \sigma(X) \twoheadrightarrow s$ with size K we also have $\mathcal{P} \vdash_{\pi CRWL} \sigma'(X) \twoheadrightarrow s$ with size $K' \leq K$, then $\vdash_{\pi CRWL} e\sigma \twoheadrightarrow t$ with size L implies $\vdash_{\pi CRWL} e\sigma' \twoheadrightarrow t$ with size $L' \leq L$.*
2. ***Monotonicity of $CSubst_\perp$:*** *If $\theta, \theta' \in CSubst_\perp$ and $\theta \sqsubseteq \theta'$ then $\mathcal{P} \vdash_{\pi CRWL} e\theta \twoheadrightarrow t$ with size K implies $\mathcal{P} \vdash_{\pi CRWL} e\theta' \twoheadrightarrow t$ with size $K' \leq K$.*
3. ***Monotonicity of $Subst_\perp$:*** *If $\sigma \trianglelefteq \sigma'$ then $[\![e\sigma]\!]^{pl} \subseteq [\![e\sigma']\!]^{pl}$.*
4. ***Monotonicity of $CSubst^?_\perp$:*** *If $\theta \sqsubseteq_\pi \theta'$ then $[\![e\theta]\!]^{pl} \subseteq [\![e\theta']\!]^{pl}$.*

We end this section with an example of the use of $\pi CRWL$ to model problems in which some collecting work has to be done.

EXAMPLE 8. *We want to represent the database of a bank in which we hold some data about its employees, this bank has several branches and we want to organize the information according to them. So we define a non-deterministic function branches to represent the set of branches: a set is identified then with a non-deterministic expression. In this line we define a non-deterministic function employees which conceptually returns the set of records containing the information regarding the employees that work in a branch. Now, to search for the names of two clerks we define the function twoclerks which is based upon find, which forces the desired pattern $e(N, S, clerk)$ over the set defined by $employees(branches)$.*

$\mathcal{P} = \{branches \to madrid, branches \to vigo, employees(madrid) \to e(pepe, men, clerk), employees(madrid) \to e(paco, men, boss), employees(vigo) \to e(maria, women, clerk), employees(vigo) \to e(jaime, women, boss), twoclerks \to find(employees(branches)), find(e(N, S, clerk)) \to (N, N)\}$

With term rewriting $twoclerks \to find(employees(branches)) \not\twoheadrightarrow^* (pepe, maria)$, because in that expression the evaluation of branches is needed and so one of the branches must be chosen. On the other hand with $\pi CRWL$ (some steps have been omitted for the sake of conciseness):

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dots}{employees(branches) \twoheadrightarrow e(pepe, \perp, clerk)}\,POR \quad \dfrac{\dots}{(pepe \,?\, maria, pepe \,?\, maria) \twoheadrightarrow (pepe, maria)}\,DC}{employees(branches) \twoheadrightarrow e(pepe, men, clerk) \twoheadrightarrow e(pepe, \perp, clerk)}\,POR \quad \dfrac{\dots}{employees(branches) \twoheadrightarrow e(maria, \perp, clerk)}\,POR}{find(employees(branches)) \twoheadrightarrow (pepe, maria)}\,POR}{twoclerks \twoheadrightarrow (pepe, maria)}\,POR$$

where

$$\dfrac{\dfrac{}{branches \twoheadrightarrow madrid}\,POR \quad \dfrac{\dots}{e(pepe, men, clerk) \twoheadrightarrow e(pepe, \perp, clerk)}\,DC}{employees(branches) \twoheadrightarrow e(pepe, \perp, clerk)}\,POR$$

4 Comparison: a hierarchy of semantics

When comparing these semantics is not surprising finding that *CRWL* and $\pi CRWL$ have similar properties. For example the monotonicity Lemma 4 also holds for *CRWL*; this lemma

does not even make sense for term rewriting, as it only works with total terms. On the other hand term rewriting is closed under *Subst* ($e \to^* e'$ implies $e\sigma \to^* e'\sigma$, for any $\sigma \in Subst$); *CRWL* is not closed under *Subst* but under $CSubst_\bot$, as corresponds to call-time choice; π*CRWL* is closed under $CSubst_\bot$ too (see [19]), and we conjecture that for $\theta \in CSubst^?_\bot$ if $\vdash_{\pi CRWL} e \twoheadrightarrow t$ then $[\![t\theta]\!]^{pl} \subseteq [\![e\theta]\!]^{pl}$. For *CRWL* a compositionality result similar to Theorem 5 also holds, and bubbling is correct too [14]. This is not the case for term rewriting, as we saw when switching from $f(c(0?1))$ to $f(c(0)?c(1))$ in examples 1 and 2.

4.1 The hierarchy

As π*CRWL* is a modification of *CRWL*, the relation between them is very direct.

THEOREM 9. *For any CRWL-program $\mathcal{P}$, $e \in Exp_\bot, t \in CTerm_\bot$ given a CRWL-proof for $\mathcal{P} \vdash e \twoheadrightarrow t$ we can build a πCRWL-proof for $\mathcal{P} \vdash_{\pi CRWL} e \twoheadrightarrow t$ just replacing every **OR** step by the corresponding **POR** step. As a consequence $[\![e]\!]^{sg}_{\mathcal{P}} \subseteq [\![e]\!]^{pl}_{\mathcal{P}}$.*

Concerning the relation of *CRWL* and π*CRWL* with term rewriting, we will use the notion of *shell* $|e|$ *of an expression* e that represents the outer constructor (and thus computed) part of e, defined as $|\bot| = \bot$, $|X| = X$, $c(e_1,\ldots,e_n) = c(|e_1|,\ldots,|e_n|)$, $|f(e_1,\ldots,e_n)| = \bot$ (for $X \in \mathcal{V}, c \in CS, f \in FS$). We also define the denotation of $e \in Exp$ under term rewriting as $[\![e]\!]^{rw} = \{t \in CTerm_\bot \mid \exists e' \in Exp\ .\ e \to^* e' \wedge t \sqsubseteq |e'|\}$. In a previous joint work the author explored the relation between *CRWL* and term rewriting ([12], Theorem 9), recast in the following theorem:

THEOREM 10. *For any CRWL-program $\mathcal{P}$, $e \in Exp$, $[\![e]\!]^{sg} \subseteq [\![e]\!]^{rw}$. The converse inclusion does not hold in general.*

As we saw in Example 1, in general call-time choice semantics like *CRWL* produce less results than run-time choice semantics like the one induced by term rewriting. We will see that this kind of relation also holds for term rewriting and π*CRWL*.

THEOREM 11. *For any CRWL-program $\mathcal{P}$, $e \in Exp$, $[\![e]\!]^{rw} \subseteq [\![e]\!]^{pl}$. The converse inclusion does not hold in general.*

The key for proving Theorem 11 is a lemma stating that $\forall e, e' \in Exp$ if $e \to e'$ then $[\![e']\!]^{pl} \subseteq [\![e]\!]^{pl}$, that is, that every rewriting step is sound wrt. π*CRWL*. The evident corollary for these theorems is the announced inclusion chain.

COROLLARY 12. *For any CRWL-program $\mathcal{P}$, $e \in Exp$, $[\![e]\!]^{sg} \subseteq [\![e]\!]^{rw} \subseteq [\![e]\!]^{pl}$. Hence $\forall t \in CTerm$, $\vdash_{CRWL} e \twoheadrightarrow t$ implies $e \to^* t$ which implies $\vdash_{\pi CRWL} e \twoheadrightarrow t$.*

5 Simulating plural semantics

In [12, 13] it was shown that neither *CRWL* can be simulated by term rewriting with a simple program transformation, nor vice versa. Nevertheless, plural semantics can be simulated by rewriting using the transformation presented in the current section, which could be used as the basis for a first implementation of π*CRWL* that we might use for experimentation. First we will present a naive version of this transformation, and show its adequacy; later we will propose some simple optimizations for it.

5.1 A simple transformation

DEFINITION 13. *Given a CRWL-program P, for every rule* $(f(p_1,\ldots,p_n) \to r) \in \mathcal{P}$ *such that* $f \notin \{\_?\_, if\_then\_\}$ *we define its transformation as:*

$$pST(f(p_1,\ldots,p_n) \to r) = f(Y_1,\ldots,Y_n) \to if\ match(Y_1,\ldots,Y_n)\ then\ r[\overline{X_{ij}/project_{ij}(Y_i)}]$$

- $\forall i \in \{1,\ldots,n\}, \{X_{i1},\ldots,X_{ik_i}\} = var(p_i) \cap var(r)$ *and* $Y_i \in \mathcal{V}$ *is fresh.*
- $match \in FS^n$ *fresh is defined by the rule* $match(p_1,\ldots,p_n) \to true$.
- *Each* $project_{ij} \in FS^1$ *is a fresh symbol defined by the single rule* $project_{ij}(p_i) \to X_{ij}$.

For $f \in \{\_?\_, if\_then\_\}$ *the transformation leaves its rules untouched.*

The function *match* is used to impose a "guard" that enforces the matching of each argument with its corresponding pattern. If we dropped this condition the translation of, for example, to rule $(null(nil) \to true)$, would be $(null(Y) \to true)$, which is clearly unsound as then $null(0 : nil) \to true$. Besides each pattern p_i has been replaced by a fresh variable Y_i, to which any expression can match, hence the arguments may be replicated freely by the rewriting process without demanding any evaluation and thus keeping its denotation untouched: this is the key to achieve completeness wrt. $\pi CRWL$. Later on, the functions $project_{ij}$ will just make the projection of each variable when needed.

EXAMPLE 14. *Applying this to Example 1 we get* $\{f(Y) \to if\ match(Y)\ then\ d(project(Y), project(Y)), match(c(X)) \to true, project(c(X)) \to X\}$ *, under which we can do:*

$\underline{f(c(0)?c(1))} \to if\ \underline{match(c(0)?c(1))}\ then\ d(project(c(0)?c(1)), project(c(0)?c(1)))$
$\to^* if\ true\ then\ d(\underline{project(c(0)?c(1))}, project(c(0)?c(1)))$
$\to d(project(\underline{c(0)?c(1)}), project(\underline{c(0)?c(1)})) \to^* d(\underline{project(c(0))}, \underline{project(c(1))}) \to^* d(0,1)$

Concerning the adequacy of the transformation:

THEOREM 15. *For any CRWL-program* $\mathcal{P}$, $e \in Exp_\perp$ *built up on the signature of* $\mathcal{P}$, $[\![e]\!]^{pl}_{pST(\mathcal{P})} \subseteq [\![e]\!]^{pl}_{\mathcal{P}}$.

THEOREM 16. *For any CRWL-program* $\mathcal{P}$, $e \in Exp, t \in CTerm_\perp$ *built up on the signature of* $\mathcal{P}$, *if* $\mathcal{P} \vdash_{\pi CRWL} e \twoheadrightarrow t$ *then exists some* $e' \in Exp$ *built using symbols of the signature of* $pST(\mathcal{P})$ *such that* $pST(\mathcal{P}) \vdash e \to^* e'$ *and* $t \sqsubseteq |e'|$.

COROLLARY 17. *For any CRWL-program* $\mathcal{P}$, $e \in Exp$ *built using symbols of the signature of* $\mathcal{P}$, $[\![e]\!]^{pl}_{\mathcal{P}} = [\![e]\!]^{rw}_{pST(\mathcal{P})}$. *Hence* $\forall t \in CTerm\ \mathcal{P} \vdash_{\pi CRWL} e \twoheadrightarrow t$ *iff* $pST(\mathcal{P}) \vdash e \to^* t$.

5.2 An optimized transformation

Concerning the transformation, if a pattern is ground then no parameter passing will be done for it and so no transformation is needed: for $null(nil) \to true$ we get $\{null(Y) \to if\ match(Y)\ then\ true, match(nil) \to true\}$, which is equivalent. Besides, if the pattern is a variable then any expression matches it and the projection functions are trivial, so no transformation is needed neither, as happens with $pair(X) \to (X, X)$ for which $\{pair(Y) \to if\ match(Y)\ then\ (project(Y), project(Y)), match(X) \to true, project(X) \to X\}$ are returned.

DEFINITION 18. *Given a CRWL-program P, for every rule* $(f(p_1,\ldots,p_n) \to r) \in \mathcal{P}$ *we define its transformation as:*

$$pST(f(p_1,\ldots,p_n) \to r) = \begin{cases} f(p_1,\ldots,p_n) \to r & \text{if } \rho_1 \ldots \rho_m \text{ is empty} \\ f(\tau(p_1),\ldots,\tau(p_n)) \to \begin{array}{l} if\ match(Y_1,\ldots,Y_m) \\ \quad then\ r[\overline{X_{ij}/project_{ij}(Y_i)}] \end{array} & \text{otherwise} \end{cases}$$

where $\rho_1 \ldots \rho_m = p_1 \ldots p_n \mid \lambda p.(p \notin \mathcal{V} \wedge var(p) \neq \emptyset)$.
- $\forall \rho_i, \{X_{i1},\ldots,X_{ik_i}\} = var(\rho_i) \cap var(r)$ *and* $Y_i \in \mathcal{V}$ *is fresh.*
- $\tau : CTerm \to CTerm$ *is defined by* $\tau(p) = p$ *if* $p \in \mathcal{V} \vee var(p) = \emptyset$ *and* $\tau(p) = Y_i$ *otherwise, for* $p \equiv \rho_i$.
- $match \in FS^m$ *fresh is defined by the rule* $match(\rho_1,\ldots,\rho_m) \to true$.
- *Each* $project_{ij} \in FS^1$ *is a fresh symbol defined by the single rule* $project_{ij}(\rho_i) \to X_{ij}$.

We will not give a formal proof for the adequacy of the optimization. Nevertheless note how this transformation leaves untouched the rules for $\_?\_$ and $if\_then\_$ without defining an special case for them. As the simple transformation worked well for that rules that suggests that we are doing the right thing. We end this section with an example application of the optimized transformation, over the program of Example 8:

EXAMPLE 19. *The only rule modified is the one for* $find$: $\{find(Y) \to if\ match(Y)\ then\ (project(Y), project(Y)), match(e(N,s,clerk)) \to true, project(e(N,s,clerk)) \to N\}$ *so:*

$$\begin{array}{l} \underline{twoclerks} \to \underline{find(employees(branches))} \\ \to if\ match(\underline{employees(branches)})\ then\ (project(employees(branches)), project(employees(branches))) \\ \to^* if\ match(\underline{e(pepe,men,clerk)})\ then\ (project(employees(branches)), project(employees(branches))) \\ \to^* (project(\underline{employees(branches)}), project(\underline{employees(branches)})) \\ \to^* (\underline{project(e(pepe,men,clerk))}, \underline{project(e(maria,women,clerk))}) \to^* (pepe, maria) \end{array}$$

6 Conclusions

In this work we have pointed the different interpretations of run-time choice and plural semantics caused by pattern matching. To the best of our knowledge this distinction is stablished in the present paper for the first time, because in [20] no pattern matching was present and in [11] only call-time choice was adopted. We argue that the run-time choice semantics induced by term rewriting is not the best option for a value-based programming language like current implementations of FLP. For that context a plural semantics has been proposed for which the compositionality properties lost when turning from call-time choice to rewriting are recovered. Nevertheless, for other kind of rewriting based languages like Maude, which are not limited to constructor-based TRS's, term rewriting has been proven to be an effective formalism.

Our concrete contributions can be summarized as follows:
- We have presented the proof calculus $\pi CRWL$, a novel formulation of plural semantics for left-linear constructor-based TRS's, which are the kind of TRS's used in FLP. Some basic properties of the new semantics have been stated and proved, and by some examples we have shown how it allows natural encodings of some programs that need to do some collecting work (Sect. 3).

- We have compared the new calculus with *CRWL* and term rewriting, which are standard formulations for call-time choice and run-time choice respectively. The different properties of these calculi have been discussed and the inclusion chain $CRWL \subseteq$ rewriting $\subseteq \pi CRWL$ has been proved (Sect. 4).
- We have recalled some previous results about the impossibility of a straight simulation of *CRWL* in term rewriting or viceversa by a simple program transformation. Besides we have proposed a novel program transformation to simulate plural semantics with term rewriting, and proved its adequacy (Sect. 5).

From a practical point of view, it might be unrealistic to think that a monolithic semantic view is adequate for addressing all non-determinism present in a large program. In [13] we have started to investigate the combination of call-time choice and run-time choice in a unified framework. But as $\pi CRWL$ seems to be more suitable than run-time choice for a value-based language, we are planning to extend that work to plural semantics.

We contemplate other relevant subjects of future work:
- Extending the current results to programs with extra variables, that is, with rules $l \to r$ in which $var(r) \subseteq var(l)$ does not hold in general. We should also deal with conditional rules and equality constraints to cover the basic features of FLP languages.
- Studying the relation between the determinism of programs under *CRWL* [12] and $\pi CRWL$, which we conjecture is equivalent. We also conjecture that for deterministic programs $\forall e \in Exp, [\![e]\!]^{sg} = [\![e]\!]^{rw} = [\![e]\!]^{pl}$. Getting results about the relation of confluence and determinism of programs could be useful for analyzing the confluence of a TRS through its determinism. In the same line, the inclusion chain $CRWL \subseteq$ rewriting $\subseteq \pi CRWL$ could be used to study the termination of a TRS through its termination in *CRWL* and $\pi CRWL$.
- Developing a more operational rewrite notion for $\pi CRWL$ in the line of [12], which could be extended to narrowing like in [14]. A complexity study would be needed to ensure that the extra nondeterminism does not preclude the design of an efficient implementation. On the other hand the natural value for $\pi CRWL$ seems to be $\mathcal{P}(CTerm_\bot)$ instead of $CTerm_\bot$, a formulation in the line of [16] could be useful to forget about the tricky use of $\_?\_$.
- Finally, for the immediate future, it could be interesting implementing the transformation to simulate $\pi CRWL$ in some term rewriting based language like Maude [5]. Maybe the context-sensitive rewriting [18] features of Maude could be used to improve the laziness of the transformed program like in [17]. Besides, the matching-module capacities of Maude could be used to enhance the expressivity of plural semantics.

Acknowledgements: The author would like to thank Paco López Fraguas and Jaime Sánchez Hernández for their support and their useful suggestions. I would also like to thank the referees for their very valuable comments.

References

[1] S. Antoy. Optimal non-deterministic functional logic computations. In *Proc. ALP'97*, pages 16–30. Springer LNCS 1298, 1997.

[2] S. Antoy, D. Brown, and S. Chiang. Lazy context cloning for non-deterministic graph rewriting. In Proc. Termgraph'06, pages 61–70. ENTCS, 176(1), 2007.

[3] S. Antoy and M. Hanus. Functional logic design patterns. In Proc. FLOPS'02, pages 67–87. Springer LNCS 2441, 2002.

[4] F. Baader and T. Nipkow. Term Rewriting and All That. Cambridge University Press, United Kingdom, 1998.
[5] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. L. Talcott, editors. All About Maude - A High-Performance Logical Framework, How to Specify, Program and Verify Systems in Rewriting Logic. Springer LNCS 4350, 2007
[6] M. Clavel, M. Palomino, and A. Riesco. Introducing the itp tool: a tutorial. J. UCS 12(11), pages 1618–1650, 2006.
[7] R. Diaconescu and K. Futatsugi. An overview of CafeOBJ. ENTCS 15,1998.
[8] J. C. González-Moreno, T. Hortalá-González, F. López-Fraguas, and M. Rodríguez-Artalejo. An approach to declarative programming based on a rewriting logic. J. Log. Program. 40(1), pages 47–87, 1999.
[9] M. Hanus. The integration of functions into logic programming: From theory to practice. J. Log. Program. 19/20, pages 583–628, 1994.
[10] M. Hanus (ed.). Curry: An integrated functional logic language (version 0.8.2). Available at *http://www.informatik.uni-kiel.de/~curry/report.html*, March 2006.
[11] H. Hussmann. Non-Determinism in Algebraic Specifications and Algebraic Programs. Birkhäuser Verlag, 1993.
[12] F. López-Fraguas, J. Rodríguez-Hortalá, and J. Sánchez-Hernández. A simple rewrite notion for call-time choice semantics. In Proc. PPDP'07, pages 197–208. ACM Press, 2007.
[13] F. López-Fraguas, J. Rodríguez-Hortalá, and J. Sánchez-Hernández. A flexible framework for programming with non-deterministic functions (Extended version). Tech. Rep. SIC-9-08, Universidad Complutense de Madrid, 2008. `http://gpd.sip.ucm.es/juanrh/pubs/tchrRTCT08.pdf`.
[14] F. López-Fraguas, J. Rodríguez-Hortalá, and J. Sánchez-Hernández. Rewriting and call-time choice: the HO case. In Proc. FLOPS'08, pages 147–162. Springer LNCS 4989, 2008.
[15] F. López-Fraguas and J. Sánchez-Hernández. $\mathcal{TOY}$: A multiparadigm declarative system. In Proc. RTA'99, pages 244–247. Springer LNCS 1631, 1999.
[16] F. López-Fraguas and J. Sánchez-Hernández. A proof theoretic approach to failure in functional logic programming. TPLP, 4(1&2), pages 41–74, 2004.
[17] S. Lucas. Needed reductions with context-sensitive rewriting. In Proc. ALP/HOA'97, pages 129–143. Springer LNCS 1298, 1997.
[18] S. Lucas. Context-sensitive computations in functional and functional logic programs. J. Fun. Log. Program 1998(1), 1998.
[19] J. Rodríguez-Hortalá. A Hierarchy of Semantics for Non-deterministic Term Rewriting Systems (Extended version). Tech. Rep. SIC-10-08, Universidad Complutense de Madrid, 2008. `http://gpd.sip.ucm.es/juanrh/pubs/tchrFSTTCS08.pdf`.
[20] H. Søndergaard and P. Sestoft. Non-determinism in functional languages. The Computer Journal 35(5), pages 514–523, 1992.
[21] M. Wenzel. The isabelle/isar reference manual. `http://isabelle.in.tum.de/dist/Isabelle99-2/doc/isar-ref.pdf`.

Foundations of Software Technology and Theoretical Computer Science (Bangalore) 2008.
Editors: R. Hariharan, M. Mukund, V. Vinay; pp 340-351

Average-Time Games*

Marcin Jurdziński[1] and Ashutosh Trivedi[2]

[1] Department of Computer Science, University of Warwick, UK
mju@dcs.warwick.ac.uk

[2] Computing Laboratory, University of Oxford, UK
trivedi@comlab.ox.ac.uk

ABSTRACT. An average-time game is played on the infinite graph of configurations of a finite timed automaton. The two players, Min and Max, construct an infinite run of the automaton by taking turns to perform a timed transition. Player Min wants to minimize the average time per transition and player Max wants to maximize it. A solution of average-time games is presented using a reduction to average-price game on a finite graph. A direct consequence is an elementary proof of determinacy for average-time games. This complements our results for reachability-time games and partially solves a problem posed by Bouyer et al., to design an algorithm for solving average-price games on priced timed automata. The paper also establishes the exact computational complexity of solving average-time games: the problem is EXPTIME-complete for timed automata with at least two clocks.

1 Introduction

Real-time open systems are computational systems that interact with environment and whose correctness depends critically on the time at which they perform some of their actions. The problem of design and verification of such systems can be formulated as *two-player zero-sum games*. A heart pacemaker is an example of a real-time open system as it interacts with the environment (heart, body movements, and breathing) and its correctness depends critically on the time at which it performs some of its actions (sending pace signals to the heart in real time). Other examples of safety-critical real-time open systems include nuclear reactor protective systems, industrial process controllers, aircraft-landing scheduling systems, satellite-launching systems, etc. Designing correct real-time systems is of paramount importance. Timed automata [2] are a popular and well-established formalism for modeling real-time systems, and games on timed automata can be used to model real-time open systems. In this paper, we introduce *average-time games* which model the interaction between the real-time open system and the environment; and we are interested in finding a strategy of the system which results in minimum average-time per transition, assuming adversarial environment.

Related Work. Games with quantitative payoffs can be studied as a model for optimal-controller synthesis [3, 1, 6]. Among various quantitative payoffs the average-price payoff [9, 8] is the most well-studied in game theory, Markov decision processes, and planning literature [8, 14], and it has numerous appealing interpretations in applications. Most algorithms for solving Markov decision processes [14] or games with average-price payoff

*This work was partially supported by the EPSRC grants EP/E022030/1 and EP/F001096/1.

FSTTCS 2008
IARCS Annual Conference on
Foundations of Software Technology and Theoretical Computer Science
http://drops.dagstuhl.de/opus/volltexte/2008/1765

work for finite graphs only [15, 8]. Asarin and Maler [3] presented the first algorithm for games on timed automata (timed games) with a quantitative payoff: reachability-time payoff. Their work was later generalized by Alur et al. [1] and Bouyer et al. [6] to give partial decidability results for reachability-price games on linearly-priced timed automata. The exact computational complexity of deciding the value in timed games with reachability-time payoff was shown to be EXPTIME in [11, 7]. Bouyer et al. [5] also studied the more difficult average-price payoffs, but only in the context of scheduling, which in game-theoretic terminology corresponds to 1-player games. They left open the problem of proving decidability of 2-player average-reward games on linearly-priced timed automata. We have recently extended the results of Bouyer et al. to solve 1-player games on more general concavely-priced timed automata [12]. In this paper we address the important and non-trivial special case of average-time games (i.e., all locations have unit costs), which was also left open by Bouyer et al.

Our Contributions. Average-time games on timed automata are introduced. This paper gives an elementary proof of determinacy for these games. A new type of region [2] based abstraction—boundary region graph—is defined, which generalizes the corner-point abstraction of Bouyer et al. [5]. Our solution allows computing the value of average-time games for an arbitrary starting state (i.e., including non-corner states). Finally, we establish the exact complexity of solving average-time games: the problem is EXPTIME-complete for timed automata with at least two clocks.

Organization of the Paper. In Section 2 we discuss average-price games (also known as mean-payoff games) on finite graphs and cite some important results for these games. In Section 3 we introduce average-time games on timed automata. In Section 4 we introduce some region-based abstractions of timed automata, including the closed region graph, and its subgraphs: the boundary region graph, and the region graph. While the region graph is semantically equivalent to the corresponding timed automaton, the boundary region graph has the property that for every starting state, the reachable state space is finite. We introduce average-time games on these graphs and show that if we have the solution of the average-time game for any of these graphs, then we get the solution of the average-time game for the corresponding timed automaton. In Section 5 we discuss the computational complexity of solving average-time games.

Notations. We assume that, wherever appropriate, sets $\mathbb{Z}$ of integers, $\mathbb{N}$ of non-negative integers and $\mathbb{R}$ of reals contain a maximum element ∞, and we write $\mathbb{N}_+$ for the set of positive integers and $\mathbb{R}_\oplus$ for the set of non-negative reals. For $n \in \mathbb{N}$, we write $(\!|n|\!)_{\mathbb{N}}$ for the set $\{0, 1, \ldots, n\}$, and $(\!|n|\!)_{\mathbb{R}}$ for the set $\{r \in \mathbb{R} : 0 \leq r \leq n\}$ of non-negative reals bounded by n. For a real number $r \in \mathbb{R}$, we write $|r|$ for its absolute value, we write $\lfloor r \rfloor$ for its integer part, i.e., the largest integer $n \in \mathbb{N}$, such that $n \leq r$, and we write $\langle\!| r |\!\rangle$ for its fractional part, i.e., we have $\langle\!| r |\!\rangle = r - \lfloor r \rfloor$.

2 Average-Price Games

A (perfect-information) two-player *average-price game* [15, 8] $\Gamma = (V, E, V_{\text{Max}}, V_{\text{Min}}, p)$ consists of a finite directed graph (V, E), a partition $V = V_{\text{Max}} \cup V_{\text{Min}}$ of vertices, and a *price function* $\pi : E \to \mathbb{Z}$. A play starts at a vertex $v_0 \in V$. If $v_0 \in V_p$, for $p \in \{\text{Max}, \text{Min}\}$, then

player p chooses a successor of the current vertex v_0, i.e., a vertex v_1, such that $(v_0, v_1) \in E$, and v_1 becomes the new current vertex. When this happens then we say that player p has made a move from the current vertex. Players keep making moves in this way indefinitely, thus forming an infinite path $r = (v_0, v_1, v_2, \dots)$ in the game graph. The goal of player Min is to minimize $\mathcal{A}_{\text{Min}}(r) = \limsup_{n\to\infty}(1/n) \cdot \sum_{i=1}^{n} \pi(v_{i-1}, v_i)$ and the goal of player Max is to maximize $\mathcal{A}_{\text{Max}}(r) = \liminf_{n\to\infty}(1/n) \cdot \sum_{i=1}^{n} \pi(v_{i-1}, v_i)$.

Strategies for players are defined as usual [15, 8]. We write Σ_{Min} (Σ_{Max}) for the set of strategies of player Min (Max) and Π_{Min} (Π_{Max}) for the set of positional strategies of player Min (Max). For strategies $\mu \in \Sigma_{\text{Min}}$ and $\chi \in \Sigma_{\text{Max}}$, and for an initial vertex $v \in V$, we write $\mathsf{run}(v, \mu, \chi)$ for the unique path formed if players start in the vertex v and then they follow strategies μ and χ, respectively. For brevity, we write $\mathcal{A}_{\text{Min}}(v, \mu, \chi)$ for $\mathcal{A}_{\text{Min}}(\mathsf{run}(v, \mu, \chi))$ and we write $\mathcal{A}_{\text{Max}}(v, \mu, \chi)$ for $\mathcal{A}_{\text{Max}}(\mathsf{run}(v, \mu, \chi))$.

For $v \in V$, we define the *upper value* $\overline{\mathsf{val}}(v) = \inf_{\mu\in\Sigma_{\text{Min}}} \sup_{\chi\in\Sigma_{\text{Max}}} \mathcal{A}_{\text{Min}}(v, \mu, \chi)$, and the *lower value* $\underline{\mathsf{val}}(v) = \sup_{\chi\in\Sigma_{\text{Max}}} \inf_{\mu\in\Sigma_{\text{Min}}} \mathcal{A}_{\text{Max}}(v, \mu, \chi)$. Note that the inequality $\underline{\mathsf{val}}(v) \leq \overline{\mathsf{val}}(v)$ always holds. A game is determined if for every $v \in V$, we have $\underline{\mathsf{val}}(v) = \overline{\mathsf{val}}(v)$. We then write $\mathsf{val}(v)$ for this number and we call it the *value* of the average-price game at the vertex v.

We say that the strategies $\mu^* \in \Sigma_{\text{Min}}$ and $\chi^* \in \Sigma_{\text{Max}}$ are *optimal* for the respective players, if for every vertex $v \in V$, we have that $\sup_{\chi\in\Sigma_{\text{Max}}} \mathcal{A}_{\text{Min}}(v, \mu^*, \chi) = \overline{\mathsf{val}}(v)$ and $\inf_{\mu\in\Sigma_{\text{Min}}} \mathcal{A}_{\text{Min}}(v, \mu^*, \chi) = \underline{\mathsf{val}}(v)$. Liggett and Lippman [13] show that all perfect-information (stochastic) average-price games are positionally determined.

THEOREM 1. *[13] Every average-price game is determined, and optimal positional strategies exist for both players, i.e., for all $v \in V$, we have:*

$$\inf_{\mu\in\Pi_{Min}} \sup_{\chi\in\Sigma_{Max}} \mathcal{A}_{Min}(v, \mu, \chi) = \sup_{\chi\in\Pi_{Max}} \inf_{\mu\in\Sigma_{Min}} \mathcal{A}_{Max}(v, \mu, \chi).$$

The decision problem for average-price games is in NP ∩ co-NP; no polynomial-time algorithm is currently known for the problem.

3 Average-Time Games

3.1 Timed Automata

Before we present the syntax of the timed automata, we need to introduce some concepts. Fix a constant $k \in \mathbb{N}$ for the rest of this paper. Let C be a finite set of *clocks*. Clocks in timed automata are usually allowed to take arbitrary non-negative real values. For the sake of simplicity and w.l.o.g [4], we restrict them to be bounded by some constant k, i.e., we consider only *bounded* timed automata models. A (k-bounded) *clock valuation* is a function $\nu : C \to (\!|k|\!)_{\mathbb{R}}$; we write $\mathcal{V}$ for the set $[C \to (\!|k|\!)_{\mathbb{R}}]$ of clock valuations. If $\nu \in \mathcal{V}$ and $t \in \mathbb{R}_{\oplus}$ then we write $\nu + t$ for the clock valuation defined by $(\nu + t)(c) = \nu(c) + t$, for all $c \in C$. For a set $C' \subseteq C$ of clocks and a clock valuation $\nu : C \to \mathbb{R}_{\oplus}$, we define $\mathsf{reset}(\nu, C')(c) = 0$ if $c \in C'$, and $\mathsf{reset}(\nu, C')(c) = \nu(c)$ if $c \notin C'$. A *corner* is an integer clock valuation, i.e., α is a corner if $\alpha(c) \in (\!|k|\!)_{\mathbb{N}}$, for every clock $c \in C$.

The set of *clock constraints* over the set of clocks C is the set of conjunctions of *simple clock constraints*, which are constraints of the form $c \bowtie i$ or $c - c' \bowtie i$, where $c, c' \in C$, $i \in (\!|k|\!)_{\mathbb{N}}$, and $\bowtie \in \{ <, >, =, \leq, \geq \}$. There are finitely many simple clock constraints. For every clock valuation $\nu \in \mathcal{V}$, let $\mathsf{SCC}(\nu)$ be the set of simple clock constraints which hold in $\nu \in \mathcal{V}$. A *clock region* is a maximal set $P \subseteq \mathcal{V}$, such that for all $\nu, \nu' \in P$, $\mathsf{SCC}(\nu) = \mathsf{SCC}(\nu')$. In other words, every clock region is an equivalence class of the indistinguishability-by-clock-constraints relation, and vice versa. Note that ν and ν' are in the same clock region iff all clocks have the same integer parts in ν and ν', and if the partial orders of the clocks, determined by their fractional parts in ν and ν', are the same. For all $\nu \in \mathcal{V}$, we write $[\nu]$ for the clock region of ν. A *clock zone* is a convex set of clock valuations, which is a union of a set of clock regions. Note that a set of clock valuations is a zone iff it is definable by a clock constraint. For $W \subseteq \mathcal{V}$, we write $\mathsf{clos}(W)$ for the smallest closed set in $\mathcal{V}$ which contains W. Observe that for every clock zone W, the set $\mathsf{clos}(W)$ is also a clock zone.

Let L be a finite set of *locations*. A *configuration* is a pair (ℓ, ν), where $\ell \in L$ is a location and $\nu \in \mathcal{V}$ is a clock valuation; we write Q for the set of configurations. If $s = (\ell, \nu) \in Q$ and $c \in C$, then we write $s(c)$ for $\nu(c)$. A *region* is a pair (ℓ, P), where ℓ is a location and P is a clock region. If $s = (\ell, \nu)$ is a configuration then we write $[s]$ for the region $(\ell, [\nu])$. We write $\mathcal{R}$ for the set of regions. A set $Z \subseteq Q$ is a *zone* if for every $\ell \in L$, there is a clock zone W_ℓ (possibly empty), such that $Z = \{(\ell, \nu) : \ell \in L \text{ and } \nu \in W_\ell\}$. For a region $R = (\ell, P) \in \mathcal{R}$, we write $\mathsf{clos}(R)$ for the zone $\{(\ell, \nu) : \nu \in \mathsf{clos}(P)\}$.

A *timed automaton* $\mathcal{T} = (L, C, S, A, E, \delta, \varrho)$ consists of a finite set of locations L, a finite set of clocks C, a set of *states* $S \subseteq Q$, a finite set of *actions* A, an *action enabledness function* $E : A \to 2^S$, a *transition function* $\delta : L \times A \to L$, and a *clock reset function* $\varrho : A \to 2^C$. We require that S, and $E(a)$ for all $a \in A$, are zones.

Clock zones, from which zones S, and $E(a)$, for all $a \in A$, are built, are typically specified by clock constraints. Therefore, when we consider a timed automaton as an input of an algorithm, its size should be understood as the sum of sizes of encodings of L, C, A, δ, and ϱ, and the sizes of encodings of clock constraints defining zones S, and $E(a)$, for all $a \in A$. Our definition of a timed automaton may appear to differ from the usual ones [2, 4], but the differences are superficial.

For a configuration $s = (\ell, \nu) \in Q$ and $t \in \mathbb{R}_\oplus$, we define $s + t$ to be the configuration $s' = (\ell, \nu + t)$ if $\nu + t \in \mathcal{V}$, and we then write $s \rightharpoonup_t s'$. We write $s \rightharpoonup_t s'$ if $s \rightharpoonup_t s'$ and for all $t' \in [0, t]$, we have $(\ell, \nu + t') \in S$. For an action $a \in A$, we define $\mathsf{succ}(s, a)$ to be the configuration $s' = (\ell', \nu')$, where $\ell' = \delta(\ell, a)$ and $\nu' = \mathsf{reset}(\nu, \varrho(a))$, and we then write $s \stackrel{a}{\rightharpoonup} s'$. We write $s \stackrel{a}{\to} s'$ if $s \stackrel{a}{\rightharpoonup} s'$; $s, s' \in S$; and $s \in E(a)$. For technical convenience, and without loss of generality, we will assume throughout that for every $s \in S$, there exists $a \in A$, such that $s \stackrel{a}{\to} s'$. For $s, s' \in S$, we say that s' is in the future of s, or equivalently, that s is in the past of s', if there is $t \in \mathbb{R}_\oplus$, such that $s \rightharpoonup_t s'$; we then write $s \rightharpoonup_* s'$.

For $R, R' \in \mathcal{R}$, we say that R' is in the future of R, or that R is in the past of R', if for all $s \in R$, there is $s' \in R'$, such that s' is in the future of s; we then write $R \rightharpoonup_* R'$. Similarly, for $R, R' \in \mathcal{R}$, we write $R \stackrel{a}{\to} R'$ if there is $s \in R$, and there is $s' \in R'$, such that $s \stackrel{a}{\to} s'$.

A *timed action* is a pair $\tau = (t, a) \in \mathbb{R}_\oplus \times A$. For $s \in Q$, we define $\mathsf{succ}(s, \tau) = \mathsf{succ}(s, (t, a))$ to be the configuration $s' = \mathsf{succ}(s + t, a)$, i.e., such that $s \rightharpoonup_t s'' \stackrel{a}{\rightharpoonup} s'$, and we

then write $s \xrightarrow{a}_t s'$. We write $s \xrightarrow{a}_t s'$ if $s \rightarrow_t s'' \xrightarrow{a} s'$, and we then say that $(s,(t,a),s')$ is a *transition* of the timed automaton. If $\tau = (t,a)$ then we write $s \xrightarrow{\tau} s'$ instead of $s \xrightarrow{a}_t s'$, and $s \xrightarrow{\tau} s'$ instead of $s \xrightarrow{a}_t s'$.

An infinite run of a timed automaton is a sequence $r = \langle s_0, \tau_1, s_1, \tau_2, \ldots \rangle$, such that for all $i \geq 1$, we have $s_{i-1} \xrightarrow{\tau_i} s_i$. A finite run of a timed automaton is a finite sequence $\langle s_0, \tau_1, s_1, \tau_2, \ldots, \tau_n, s_n \rangle \in S \times ((A \times \mathbb{R}_\oplus) \times S)^*$, such that for all $i, 1 \leq i \leq n$, we have $s_{i-1} \xrightarrow{\tau_i} s_i$. For a finite run $r = \langle s_0, \tau_1, s_1, \tau_2, \ldots, \tau_n, s_n \rangle$, we define $\mathsf{length}(r) = n$, and we define $\mathsf{last}(r) = s_n$ to be the state in which the run ends. For a finite run $r = \langle s_0, \tau_1, s_1, \tau_2, \ldots, s_n \rangle$, we define time of the run as $\mathsf{time}(r) = \sum_{i=1}^{n} t_i$. We write $\mathrm{Runs}_{\mathrm{fin}}$ for the set of finite runs.

3.2 Strategies

An average-time game Γ is a triple $(\mathcal{T}, L_{\mathrm{Min}}, L_{\mathrm{Max}})$, where $\mathcal{T} = (L, C, S, A, E, \delta, \varrho)$ is a timed automaton and $(L_{\mathrm{Min}}, L_{\mathrm{Max}})$ is a partition of L. We define $Q_{\mathrm{Min}} = \{(\ell, \nu) \in Q : \ell \in L_{\mathrm{Min}}\}$, $Q_{\mathrm{Max}} = Q \setminus Q_{\mathrm{Min}}$, $S_{\mathrm{Min}} = S \cap Q_{\mathrm{Min}}$, $S_{\mathrm{Max}} = S \setminus S_{\mathrm{Min}}$, $\mathcal{R}_{\mathrm{Min}} = \{[s] : s \in Q_{\mathrm{Min}}\}$, and $\mathcal{R}_{\mathrm{Max}} = \mathcal{R} \setminus \mathcal{R}_{\mathrm{Min}}$.

A *strategy* for Min is a function $\mu : \mathrm{Runs}_{\mathrm{fin}} \to A \times \mathbb{R}_\oplus$, such that if $\mathsf{last}(r) = s \in S_{\mathrm{Min}}$ and $\mu(r) = \tau$ then $s \xrightarrow{\tau} s'$, where $s' = \mathsf{succ}(s, \tau)$. Similarly, a strategy for player Max is a function $\chi : \mathrm{Runs}_{\mathrm{fin}} \to A \times \mathbb{R}_\oplus$, such that if $\mathsf{last}(r) = s \in S_{\mathrm{Max}}$ and $\chi(r) = \tau$ then $s \xrightarrow{\tau} s'$, where $s' = \mathsf{succ}(s, \tau)$. We write Σ_{Min} for the set of strategies for player Min, and we write Σ_{Max} for the set of strategies for player Max. If players Min and Max use strategies μ and χ, resp., then the (μ, χ)-run from a state s is the unique run $\mathsf{run}(s, \mu, \chi) = \langle s_0, \tau_1, s_1, \tau_2, \ldots \rangle$, such that $s_0 = s$, and for every $i \geq 1$, if $s_i \in S_{\mathrm{Min}}$, or $s_i \in S_{\mathrm{Max}}$, then $\mu(\mathsf{run}_i(s, \mu, \chi)) = \tau_{i+1}$, or $\chi(\mathsf{run}_i(s, \mu, \chi)) = \tau_{i+1}$, resp., where $\mathsf{run}_i(s, \mu, \chi) = \langle s_0, \tau_1, s_1, \ldots, s_{i-1}, \tau_i, s_i \rangle$.

We say that a strategy μ for Min is *positional* if for all finite runs $r, r' \in \mathrm{Runs}_{\mathrm{fin}}$, we have that $\mathsf{last}(r) = \mathsf{last}(r')$ implies $\mu(r) = \mu(r')$. A positional strategy for player Min can be then represented as a function $\mu : S_{\mathrm{Min}} \to A \times \mathbb{R}_\oplus$, which uniquely determines the strategy $\mu^\infty \in \Sigma_{\mathrm{Min}}$ as follows: $\mu^\infty(r) = \mu(\mathsf{last}(r))$, for all finite runs $r \in \mathrm{Runs}_{\mathrm{fin}}$. Positional strategies for player Max are defined and represented in the analogous way. We write Π_{Min} and Π_{Max} for the sets of positional strategies for player Min and for player Max, respectively.

3.3 Value of Average-Time Game

If player Min uses the strategy $\mu \in \Sigma_{\mathrm{Min}}$ and player Max uses the strategy $\chi \in \Sigma_{\mathrm{Max}}$ then player Min loses the value $\mathcal{A}_{\mathrm{Min}}(s, \mu, \chi) = \limsup_{n \to \infty} (1/n) \cdot \mathsf{time}(\mathsf{run}_n(s, \mu, \chi))$, and player Max wins the value $\mathcal{A}_{\mathrm{Max}}(s, \mu, \chi) = \liminf_{n \to \infty} (1/n) \cdot \mathsf{time}(\mathsf{run}_n(s, \mu, \chi))$. In an average-time game player Min is interested in minimizing the value she loses and player Max is interested in maximizing the value he wins. For every state $s \in S$ of a timed automaton, we define its *upper value* by $\overline{\mathsf{val}}^{\mathcal{T}}(s) = \inf_{\mu \in \Sigma_{\mathrm{Min}}} \sup_{\chi \in \Sigma_{\mathrm{Max}}} \mathcal{A}_{\mathrm{Min}}(s, \mu, \chi)$, and its lower value $\underline{\mathsf{val}}^{\mathcal{T}}(s) = \sup_{\chi \in \Sigma_{\mathrm{Max}}} \inf_{\mu \in \Sigma_{\mathrm{Min}}} \mathcal{A}_{\mathrm{Max}}(s, \mu, \chi)$.

The inequality $\underline{\mathsf{val}}^{\mathcal{T}}(s) \leq \overline{\mathsf{val}}^{\mathcal{T}}(s)$ always holds. An average-time game is *determined* if for every state $s \in S$, its lower and upper values are equal to each other; then we say that the *value* $\mathsf{val}^{\mathcal{T}}(s)$ exists and $\mathsf{val}^{\mathcal{T}}(s) = \underline{\mathsf{val}}^{\mathcal{T}}(s) = \overline{\mathsf{val}}^{\mathcal{T}}(s)$. For strategies $\mu \in \Sigma_{\mathrm{Min}}$ and

$\chi \in \Sigma_{\text{Max}}$, we define $\mathsf{val}^{\mu}(s) = \sup_{\chi \in \Sigma_{\text{Min}}} \mathcal{A}_{\text{Min}}(s, \mu, \chi)$, and $\mathsf{val}^{\chi}(s) = \inf_{\mu \in \Sigma_{\text{Min}}} \mathcal{A}_{\text{Max}}(s, \mu, \chi)$. For an $\varepsilon > 0$, we say that a strategy $\mu \in \Sigma_{\text{Min}}$ or $\chi \in \Sigma_{\text{Max}}$ is *ε-optimal* if for every $s \in S$ we have that $\mathsf{val}^{\mu}(s) \leq \mathsf{val}^{\mathcal{T}}(s) + \varepsilon$ or $\mathsf{val}^{\chi}(s) \geq \mathsf{val}^{\mathcal{T}}(s) - \varepsilon$, respectively. Note that if a game is determined then for every $\varepsilon > 0$, both players have ε-optimal strategies.

We say that a strategy $\chi \in \Sigma_{\text{Max}}$ of player Max is a best response to a strategy $\mu \in \Sigma_{\text{Min}}$ of player Min if for all $s \in S$ we have that $\mathcal{A}_{\text{Min}}(s, \mu, \chi) = \sup_{\chi' \in \Sigma_{\text{Max}}} \mathcal{A}_{\text{Min}}(s, \mu, \chi')$. Similarly we say that a strategy $\mu \in \Sigma_{\text{Min}}$ of player Min is a best response to a strategy $\chi \in \Sigma_{\text{Max}}$ of player Max if for all $s \in S$ we have that $\mathcal{A}_{\text{Max}}(s, \mu, \chi) = \inf_{\mu' \in \Sigma_{\text{Min}}} \mathcal{A}_{\text{Max}}(s, \mu', \chi)$.

4 Region Abstractions

4.1 Region Graphs

The region automaton, originally proposed by Alur and Dill [2], is a useful abstraction of a timed automaton as it preserves the validity of qualitative reachability, safety, and ω-regular properties. The *region automaton* [2] $\mathsf{RA}(\mathcal{T}) = (\mathcal{R}, \mathcal{M})$ of a timed automaton $\mathcal{T}$ consists of:

- the set $\mathcal{R}$ of regions of $\mathcal{T}$, and
- $\mathcal{M} \subseteq \mathcal{R} \times (\mathcal{R} \times A) \times \mathcal{R}$, such that for all $a \in A$, and for all $R, R', R'' \in \mathcal{R}$, we have that $(R, R'', a, R') \in \mathcal{M}$ iff $R \rightarrow_* R'' \xrightarrow{a} R'$.

The region automaton, however, is not sufficient for solving average-time games as it abstract away the timing information. Corner-point abstraction, introduced by Bouyer et al. [5], is a refinement of region automaton which preserves some timing information. Formally, the corner-point abstraction $\mathsf{CP}(\mathcal{T})$ of a timed automaton $\mathcal{T}$ is a finite graph (V, E) such that:

- $V \subseteq Q \times \mathcal{R}$ such that $(s, R) \in V$ iff $s = (\ell, \nu) \in \mathsf{clos}(R)$ and ν is a corner. Since timed automata we consider are bounded, there are finitely many regions, and every region has a finite number of corners. Hence the set of vertices finite.
- $E \subseteq V \times (\mathbb{R}_{\oplus} \times \mathcal{R} \times A) \times V$ such that for $(s, R), (s', R') \in V$ and $(t, R'', a) \in \mathbb{R}_{\oplus} \times \mathcal{R} \times A$, we have $((s, R), (t, R'', a), (s', R')) \in E$ iff $R \rightarrow_* R'' \xrightarrow{a} R'$ and $(s + t) \xrightarrow{a} s'$. Notice that such a t is always a natural number.

Bouyer et al. [5] showed that the corner-point abstraction is sufficient for deciding one-player average-price problem if the initial state is a corner-state, i.e., a state whose clock valuation is a corner. It follows from our results that the corner-point abstraction can be used to solve average-time games on timed automata if the initial state is a corner state.

We introduce the *boundary region graph*, which is a generalization of the corner-point abstraction. We prove that the value of the average-time game on a timed automaton is equal to the value of the average-time game on the corresponding boundary region graph, for all starting states, not just for corner states. In the process, we introduce two other refinements of the region automaton, which we call the *closed region graph* and the *region graph*. The analysis of average-time games on those objects allows us to establish equivalence of average-time games on the original timed automaton and the boundary region graph.

Closed Region Graph. A *closed region graph* $\overline{\mathcal{T}} = (\overline{Q}, \overline{E})$ of a timed automaton $\mathcal{T}$ is a refinement of its region automaton, where $\overline{Q} = \{(s, R) : s \in \mathsf{clos}(R) \text{ and } R \in \mathcal{R}\}$ and $\overline{E} \subseteq \overline{Q} \times (\mathbb{R}_{\oplus} \times \mathcal{R} \times A) \times \overline{Q}$, such that for all $(s, R), (s', R') \in \overline{Q}$ and $(t, R'', a) \in \mathbb{R}_{\oplus} \times \mathcal{R} \times A$,

we have $((s,R),(t,R'',a),(s',R')) \in \overline{E}$ iff $s' = \mathsf{succ}(s,t,a)$, $(R,R'',a,R') \in \mathcal{M}$, and $s+t \in \mathsf{clos}(R'')$. For a region $R \in \mathcal{R}$ we define the set $\overline{Q}(R) \subseteq \overline{Q}$ to be $\{(s,R) : (s,R) \in \overline{Q}\}$.

Boundary Region Graph. For a timed automaton $\mathcal{T}$, its *boundary region graph* $\widehat{\mathcal{T}} = (\widehat{Q},\widehat{E})$ is a sub-graph of its closed region graph $\overline{\mathcal{T}} = (\overline{Q},\overline{E})$ with $\widehat{Q} = \overline{Q}$ and $\widehat{E} \subseteq \overline{E}$, such that for all $(s,R),(s',R') \in \widehat{Q}$ and $(t,R'',a) \in \mathbb{R}_{\oplus} \times \mathcal{R} \times A$, we have $((s,R),(t,R'',a),(s',R')) \in \widehat{E}$ if: either $R \in \mathcal{R}_{\mathrm{Min}}$ and $t = \inf\{t : s+t \in \mathsf{clos}(R'')\}$, or $R \in \mathcal{R}_{\mathrm{Max}}$ and $t = \sup\{t : s+t \in \mathsf{clos}(R'')\}$. Boundary region graphs have the following property.

PROPOSITION 2. *For every configuration in a boundary region graph the set of reachable configurations is finite.*

We say that a configuration $q = (s = (\ell,\nu),R)$ is *corner configuration* if ν is a corner.

PROPOSITION 3. *The reachable sub-graph of the a boundary region graph $\widehat{\mathcal{T}}$ from a corner configuration is same as the corner-point abstraction $\mathsf{CP}(\mathcal{T})$.*

Region Graph. The *region graph* $\widetilde{\mathcal{T}} = (\widetilde{Q},\widetilde{E})$ of a timed automaton $\mathcal{T}$ is a sub-graph of its closed region graph $\overline{\mathcal{T}} = (\overline{Q},\overline{E})$ with $\widetilde{Q} = \overline{Q}$ and $\widetilde{E} \subseteq \overline{E}$, such that $((s,R),(t,R'',a),(s',R')) \in \widetilde{E}$ if $s+t \in R''$. The timed automaton $\mathcal{T}$ and the corresponding region graph $\widetilde{\mathcal{T}}$ are equivalent in the following sense.

PROPOSITION 4. *Let $\mathcal{T}$ be a timed automaton and $\widetilde{\mathcal{T}} = (\widetilde{Q},\widetilde{E})$ be its region graph. For every $s,s' \in S$ and $(t,a) \in \mathbb{R}_{\oplus} \times A$, we have $s \xrightarrow{a}_t s'$ if and only if $((s,[s]),(t,[s+t],a),(s',[s'])) \in \widetilde{E}$.*

Runs of Region Graphs. An infinite run of the closed region graph $\overline{\mathcal{T}}$ is an infinite sequence $\langle q_0,\tau_1,q_1,\tau_1,\ldots\rangle$, such that for all $i \geq 1$, we have $(q_{i-1},\tau_i,q_i) \in \overline{E}$. A finite run of the closed region graph $\overline{\mathcal{T}}$ is a finite sequence $\langle q_0,\tau_1,q_1,\tau_1,\ldots,q_n\rangle \in \overline{Q} \times ((\mathbb{R}_{\oplus} \times \mathcal{R} \times A) \times \overline{Q})^*$, such that for all $1 \leq i \leq n$, we have $(q_{i-1},\tau_i,q_i) \in \overline{E}$. Runs of the boundary region graph and the region graph are defined analogously. For a graph $\mathcal{G} \in \{\overline{\mathcal{T}},\widehat{\mathcal{T}},\widetilde{\mathcal{T}}\}$, we write $\mathrm{Runs}^{\mathcal{G}}_{\mathrm{fin}}$ for the set of its finite runs and $\mathrm{Runs}^{\mathcal{G}}_{\mathrm{fin}}(q)$ for the set of its finite runs from a configuration $q \in \overline{Q}$. Notice that for all $q \in \overline{Q}$ we have that $\mathrm{Runs}^{\widehat{\mathcal{T}}}(q) \subseteq \mathrm{Runs}^{\overline{\mathcal{T}}}(q)$ and $\mathrm{Runs}^{\widetilde{\mathcal{T}}}(q) \subseteq \mathrm{Runs}^{\overline{\mathcal{T}}}(q)$. For a finite run $r = \langle q_0,(t_1,R_1,a_1),q_1,(t_2,R_2,a_2),\ldots,q_n\rangle$ we define $\mathsf{time}(r) = \sum_{i=1}^{n} t_i$, and we denote the last configuration of the run by $\mathsf{last}(r) = q_n$.

Run Types of Region Graphs. Type of a finite run $\langle (s_0,R_0),(t_1,R'_1,a_1),(s_1,R_1),\ldots,(s_n,R_n)\rangle$ is the finite sequence $\langle R_0,(R'_1,a_1),R_1,(R'_2,a_2),\ldots,R_n\rangle$. The type of an infinite run is defined analogously. For a (finite or infinite) run r, we write $[\![r]\!]_{\mathcal{R}}$ for its type. We write $\mathrm{Types}_{\mathrm{fin}}$ and Types for the set of types of finite runs and the set of types of infinite runs, respectively.

4.2 Simple Functions and Boundary Timed Actions

A function $F : \overline{Q} \to \mathbb{R}$ is *simple* [3, 11] if either: there is $e \in \mathbb{Z}$, such that for every $(s,R) \in \overline{Q}$, we have $F(s,R) = e$; or there are $e \in \mathbb{Z}$ and $c \in C$, such that for every $(s,R) \in \overline{Q}$ we have $F(s,R) = e - s(c)$. We say that a function $F : \overline{Q} \to \mathbb{R}$ is *regionally simple* or *regionally constant*, respectively, if for every region $R \in \mathcal{R}$ the function F, over domain $\overline{Q}(R)$, is simple or constant, respectively.

Define the finite set of *boundary timed actions* $\mathbb{A} = (\!|k|\!)_{\mathbb{N}} \times C \times A \times \mathcal{R}$. For $q = (s,R) \in \overline{Q}$ and $\alpha = (b,c,a,R'') \in \mathbb{A}$, we define $t(s,\alpha) = b - s(c)$. If $s + t(s,\alpha) \in \mathsf{clos}(R'')$ then the

function $\mathsf{succ}(q,\alpha)$ is defined and we have $q' = (\mathsf{succ}(s,\tau(\alpha)), R')$, where $\tau(\alpha) = (t(s,\alpha), a)$ and $R'' \xrightarrow{a} R'$. We sometimes write $q \xrightarrow{\alpha} q'$ if $q' = \mathsf{succ}(q,\alpha)$.

4.3 Strategies

Let $\Gamma = (\mathcal{T}, L_{\text{Min}}, L_{\text{Max}})$ be an average-time game. The partition $(L_{\text{Min}}, L_{\text{Max}})$ naturally gives rise to average-time games on the closed region graph $\overline{\Gamma} = (\overline{\mathcal{T}}, \overline{Q}_{\text{Min}}, \overline{Q}_{\text{Max}})$, the boundary region graph $\widehat{\Gamma} = (\widehat{\mathcal{T}}, \widehat{Q}_{\text{Min}}, \widehat{Q}_{\text{Max}})$, and the region graph $\widetilde{\Gamma} = (\widetilde{\mathcal{T}}, \widetilde{Q}_{\text{Min}}, \widetilde{Q}_{\text{Max}})$.

In a closed region graph, a strategy of player Min μ is a (partial) function $\mu : \text{Runs}_{\text{fin}}^{\overline{\mathcal{T}}} \to \mathbb{R}_{\oplus} \times \mathcal{R} \times A$, such that for a run $r \in \text{Runs}_{\text{fin}}^{\overline{\mathcal{T}}}$, if $\mathsf{last}(r) = (s,R) \in Q_{\text{Min}}$ then $\mu(r) = (t, R', a)$ is defined, and it is such that $(s+t) \in \mathsf{clos}(R')$ and $(R,(R',a),R'') \in \mathcal{M}$, for some $R'' \in \mathcal{R}$. Strategies of player Max is defined analogously. We write $\overline{\Sigma}_{\text{Min}}$ and $\overline{\Sigma}_{\text{Max}}$ for the set of strategies of player Min and player Max, respectively. We say that a strategy σ is positional if for all runs $r_1, r_2 \in \text{Runs}_{\text{fin}}^{\overline{\mathcal{T}}}$, $\mathsf{last}(r_1) = \mathsf{last}(r_2)$ implies $\mu(r_1) = \mu(r_2)$. We define the run starting from configuration $q \in \overline{Q}$ and following strategies μ and χ, of player Max and player Min, respectively, in a straightfoward manner and we write $\mathsf{run}(q,\mu,\chi)$ to denote this run. For every $n \geq 1$, we write $\mathsf{run}_n(q,\mu,\chi)$ for the prefix of the run $\mathsf{run}(q,\mu,\chi)$ of length n.

We say that a strategy σ is an *admissible strategy* if for all finite runs $r \in \text{Runs}_{\text{fin}}^{\overline{\mathcal{T}}}$, we have $\sigma(r) = (t, R', a)$ such that $s + t \in R'$, where $(s,R) = \mathsf{last}(r)$. Note that both players have only admissible strategies on the region graph. We write $\widetilde{\Sigma}_{\text{Min}}$ and $\widetilde{\Sigma}_{\text{Max}}$ for the set of admissible strategies of player Min and player Max, respectively.

We say that a strategy μ of player Min is a *boundary strategy* if for all finite runs $r \in \text{Runs}_{\text{fin}}^{\overline{\mathcal{T}}}$, we have $\mu(r) = (t, R', a)$, such that $t = \inf\{t : s + t \in \mathsf{clos}(R')\}$, where $(s,R) = \mathsf{last}(r)$. We say that a strategy χ of player Max is a *boundary strategy* if for all finite runs $r \in \text{Runs}_{\text{fin}}^{\overline{\mathcal{T}}}$, we have $\chi(r) = (t, R', a)$, such that $t = \sup\{t : s + t \in \mathsf{clos}(R')\}$, where $(s,R) = \mathsf{last}(r)$. Both players have only boundary strategies in the boundary region graph. We write $\widehat{\Sigma}_{\text{Min}}$ and $\widehat{\Sigma}_{\text{Max}}$ for the set of boundary strategies of player Min and player Max, respectively.

PROPOSITION 5. *For every boundary strategy σ and for every run r, if $\sigma(r) = (t, R', a)$ then there exists a boundary timed action $\alpha = (b, c, a, R') \in \mathbb{A}$ such that $t(s,\alpha) = t$, where $(s,R) = \mathsf{last}(r)$.*

By Proposition 5 a run of the closed region graph in which both players use boundary strategies, can be represented as a sequence $\langle q_0, \alpha_1, q_1, \alpha_2, \ldots \rangle$. Such a run is called a *boundary run*. For a boundary strategy σ, we define the function $\widehat{\sigma} : \text{Runs}_{\text{fin}}^{\overline{\mathcal{T}}} \to \mathbb{A}$ as follows: if for a run r we have $\sigma(r) = (t, R', a)$, then $\widehat{\sigma}(r) = (b, c, a, R')$, such that $b - s(c) = t$, where $(s,R) = \mathsf{last}(r)$.

Type-Preserving Boundary Strategies. We say that a boundary strategy σ is *type-preserving*, if for all finite runs $r_1, r_2 \in \text{Runs}_{\text{fin}}^{\overline{\mathcal{T}}}$ such that $[\![r_1]\!]_{\mathcal{R}} = [\![r_2]\!]_{\mathcal{R}}$, we have that $\widehat{\sigma}(r_1) = \widehat{\sigma}(r_2)$. We write Ξ_{Min} and Ξ_{Max} for the sets of type-preserving boundary strategies of players Min and Max, respectively. Notice that for type-preserving boundary strategies $\mu \in \Xi_{\text{Min}}$ and $\chi \in \Xi_{\text{Max}}$, for every region $R \in \mathcal{R}$ and for all configurations $q, q' \in \overline{Q}(R)$, we have that $[\![\mathsf{run}(q,\mu,\chi)]\!]_{\mathcal{R}} = [\![\mathsf{run}(q',\mu,\chi)]\!]_{\mathcal{R}}$.

Note that the following inclusions hold.

$$\Xi_{\text{Min}} \subseteq \widehat{\Sigma}_{\text{Min}} \subseteq \overline{\Sigma}_{\text{Min}} \quad \text{and} \quad \widetilde{\Sigma}_{\text{Min}} \subseteq \overline{\Sigma}_{\text{Min}}, \quad \text{and}$$
$$\Xi_{\text{Max}} \subseteq \widehat{\Sigma}_{\text{Max}} \subseteq \overline{\Sigma}_{\text{Max}} \quad \text{and} \quad \widetilde{\Sigma}_{\text{Max}} \subseteq \overline{\Sigma}_{\text{Max}}$$

PROPOSITION 6. *For every $n \geq 1$, and for all type-preserving boundary strategies $\mu \in \Xi_{Min}$ and $\chi \in \Xi_{Max}$, the function $\mathsf{time}(\mathsf{run}_n(\cdot, \mu, \chi))$ is regionally simple.*

Given a type-preserving boundary strategy σ and $\varepsilon > 0$, we define an admissible strategy σ_ε as follows: for a finite run $r \in \text{Runs}_{\text{fin}}^{\overline{\mathcal{T}}}$, if $\widehat{\sigma}(r) = (b, c, a, R')$ then $\sigma_\varepsilon(r) = (t, R', a)$ such that $b - s(c) - \varepsilon \leq t \leq b - s(c) + \varepsilon$, where $(s, R) = \mathsf{last}(r)$.

Given a boundary strategy σ and a configuration $q \in \overline{Q}$, we define the type-preserving boundary strategy $\sigma[q]$, which agrees with the strategy σ on all the runs starting from the configuration q. Formally, for a given σ the type-preserving boundary strategy $\sigma[q]$ is such that for all runs $r \in \text{Runs}_{\text{fin}}(q)$, we have $\widehat{\sigma[q]}(r) = \widehat{\sigma}(r)$.

4.4 Value of Average-Time Game

For the strategies $\mu \in \overline{\Sigma}_{\text{Min}}$ and $\chi \in \overline{\Sigma}_{\text{Max}}$ of respective players and a configuration $q \in \overline{Q}$ we define $\mathcal{A}_{\text{Min}}(q, \mu, \chi) = \limsup_{n\to\infty}(1/n) \cdot \mathsf{time}(\mathsf{run}_n(q, \mu, \chi))$ and $\mathcal{A}_{\text{Max}}(q, \mu, \chi) = \liminf_{n\to\infty}(1/n) \cdot \mathsf{time}(\mathsf{run}_n(q, \mu, \chi))$. For average-time games on a graph $\mathcal{G} \in \{\overline{\mathcal{T}}, \widehat{\mathcal{T}}, \widetilde{\mathcal{T}}\}$ we define the lower-value $\underline{\mathsf{val}}^{\mathcal{G}}(q)$, the upper-value $\overline{\mathsf{val}}^{\mathcal{G}}(q)$ and the value $\mathsf{val}^{\mathcal{G}}(q)$ of a configuration $q \in \overline{Q}$ in a straightfoward manner.

4.5 Determinacy of Average-Time Games on the Boundary Region Graph

Positional determinacy of average-time games on the boundary region graph is immediate from Proposition 2 and Theorem 1.

THEOREM 7. *The average-time game on $\widehat{\mathcal{T}}$ is determined, and there are optimal positional strategies in $\widehat{\mathcal{T}}$, i.e., for every $q \in \overline{Q}$, we have:*

$$\mathsf{val}^{\widehat{\mathcal{T}}}(q) = \inf_{\mu \in \widehat{\Pi}_{Min}} \sup_{\chi \in \widehat{\Sigma}_{Max}} \mathcal{A}_{Min}(q, \mu, \chi) = \sup_{\chi \in \widehat{\Pi}_{Max}} \inf_{\mu \in \widehat{\Sigma}_{Min}} \mathcal{A}_{Max}(q, \mu, \chi).$$

PROPOSITION 8. *For all $\mu \in \Xi_{Min}$ and $\chi \in \Xi_{Max}$, the functions $\mathcal{A}_{Min}(\cdot, \mu, \chi)$ and $\mathcal{A}_{Max}(\cdot, \mu, \chi)$ are regionally constant.*

LEMMA 9. *In $\widehat{\mathcal{T}}$, if $\mu \in \widehat{\Sigma}_{Min}$ and $\chi \in \widehat{\Sigma}_{Max}$ are mutual best responses from $q \in \overline{Q}$, then $\mu[q] \in \Xi_{Min}$ and $\chi[q] \in \Xi_{Max}$ are mutual best responses from every $q' \in \overline{Q}([q])$.*

PROOF. We argue that $\chi[q]$ is a best response to $\mu[q]$ from $q' \in \overline{Q}([q])$ in $\widehat{\mathcal{T}}$; the other case is analogous. For all $\chi' \in \widehat{\Sigma}_{\text{Max}}$, we have the following:

$$\mathcal{A}_{\text{Min}}(q', \mu[q], \chi[q]) = \mathcal{A}_{\text{Min}}(q, \mu[q], \chi[q]) \geq \mathcal{A}_{\text{Min}}(q, \mu[q], \chi'[q']) = \mathcal{A}_{\text{Min}}(q', \mu[q], \chi'[q']) = \mathcal{A}_{\text{Min}}(q', \mu[q], \chi').$$

The first equality follows from Proposition 8; the inequality follows because χ is a best response to μ from q; the second equality follows from Proposition 8 again; and the last equality is straightforward. ∎

THEOREM 10. *There are optimal type-preserving boundary strategies in $\widehat{\mathcal{T}}$, i.e., for every $q \in \overline{Q}$, we have:*

$$\mathit{val}^{\widehat{\mathcal{T}}}(q) = \inf_{\mu \in \Xi_{Min}} \sup_{\chi \in \widehat{\Sigma}_{Max}} \mathcal{A}_{Min}(q,\mu,\chi) = \sup_{\chi \in \Xi_{Max}} \inf_{\mu \in \widehat{\Sigma}_{Min}} \mathcal{A}_{Max}(q,\mu,\chi).$$

PROOF. Let $\mu^* \in \Xi_{\mathrm{Min}}$ and $\chi^* \in \Xi_{\mathrm{Max}}$ be mutual best responses in $\widehat{\mathcal{T}}$; existence of such strategies follows from Lemma 9. Moreover, we can assume that the strategies μ^* and χ^* have finite memory; this can be achieved by taking positional strategies $\mu \in \widehat{\Sigma}_{\mathrm{Min}}$ and $\chi \in \widehat{\Sigma}_{\mathrm{Max}}$ in Lemma 9. We then have the following:

$$\inf_{\mu \in \Xi_{\mathrm{Min}}} \sup_{\chi \in \widehat{\Sigma}_{\mathrm{Max}}} \mathcal{A}_{\mathrm{Min}}(q,\mu,\chi) \leq \sup_{\chi \in \widehat{\Sigma}_{\mathrm{Max}}} \mathcal{A}_{\mathrm{Min}}(q,\mu^*,\chi) = \mathcal{A}_{\mathrm{Min}}(q,\mu^*,\chi^*) =$$
$$\mathcal{A}_{\mathrm{Max}}(q,\mu^*,\chi^*) = \inf_{\mu \in \widehat{\Sigma}_{\mathrm{Min}}} \mathcal{A}_{\mathrm{Max}}(q,\mu,\chi^*) \leq \sup_{\chi \in \Xi_{\mathrm{Max}}} \inf_{\mu \in \widehat{\Sigma}_{\mathrm{Min}}} \mathcal{A}_{\mathrm{Max}}(q,\mu,\chi).$$

The first and last inequalities are straightforward because $\mu^* \in \Xi_{\mathrm{Min}}$ and $\chi^* \in \Xi_{\mathrm{Max}}$. The first equality holds because χ^* is a best response to μ^* in $\widehat{\mathcal{T}}$, and the third equality holds because μ^* is a best response to χ^* in $\widehat{\mathcal{T}}$. Finally, the second equality holds because strategies μ^* and χ^* have finite memory. ∎

4.6 Determinacy of Average-Time Games on the Closed Region Graph

LEMMA 11. *In $\overline{\mathcal{T}}$, for every strategy in Ξ_{Min} there is a best response in Ξ_{Max}, and for every strategy in Ξ_{Max} there is a best response in Ξ_{Min}.*

THEOREM 12. *The average-time game on $\overline{\mathcal{T}}$ is determined, and there are optimal type-preserving boundary strategies in $\overline{\mathcal{T}}$, i.e., for every $q \in \overline{Q}$, we have:*

$$\mathit{val}^{\overline{\mathcal{T}}}(q) = \inf_{\mu \in \Xi_{Min}} \sup_{\chi \in \overline{\Sigma}_{Max}} \mathcal{A}_{Min}(q,\mu,\chi) = \sup_{\chi \in \Xi_{Max}} \inf_{\mu \in \overline{\Sigma}_{Min}} \mathcal{A}_{Max}(q,\mu,\chi) = \mathit{val}^{\widehat{\mathcal{T}}}(q).$$

PROOF. We have the following:

$$\inf_{\mu \in \Xi_{\mathrm{Min}}} \sup_{\chi \in \overline{\Sigma}_{\mathrm{Max}}} \mathcal{A}_{\mathrm{Min}}(q,\mu,\chi) = \inf_{\mu \in \Xi_{\mathrm{Min}}} \sup_{\chi \in \Xi_{\mathrm{Max}}} \mathcal{A}_{\mathrm{Min}}(q,\mu,\chi) =$$
$$\sup_{\chi \in \Xi_{\mathrm{Max}}} \inf_{\mu \in \Xi_{\mathrm{Min}}} \mathcal{A}_{\mathrm{Max}}(q,\mu,\chi) = \sup_{\chi \in \Xi_{\mathrm{Max}}} \inf_{\mu \in \overline{\Sigma}_{\mathrm{Min}}} \mathcal{A}_{\mathrm{Max}}(q,\mu,\chi),$$

where the first and last equalities follow from Lemma 11, and the second equality follows from Theorem 10. Now it is routine to show that $\underline{\mathsf{val}}^{\overline{\mathcal{T}}}(q) \geq \underline{\mathsf{val}}^{\widehat{\mathcal{T}}}(q)$ and $\overline{\mathsf{val}}^{\overline{\mathcal{T}}}(q) \leq \overline{\mathsf{val}}^{\widehat{\mathcal{T}}}(q)$. It concludes the proof that the average-time game on $\overline{\mathcal{T}}$ is determined, and there are optimal type-preserving boundary strategies in $\overline{\mathcal{T}}$. ∎

4.7 Determinacy of Average-Time Games on the Region Graph

LEMMA 13. *If the strategies $\mu^* \in \Xi_{Min}$ and $\chi^* \in \Xi_{Max}$ are optimal for respective players in $\overline{\mathcal{T}}$ then for every $\varepsilon > 0$, we have that*

$$\sup_{\chi \in \overline{\Sigma}_{Max}} \mathcal{A}_{Min}(q, \mu^*_\varepsilon, \chi) \leq \mathit{val}^{\overline{\mathcal{T}}}(q) + \varepsilon \text{ and } \inf_{\mu \in \overline{\Sigma}_{Min}} \mathcal{A}_{Max}(q, \mu, \chi^*_\varepsilon) \geq \mathit{val}^{\overline{\mathcal{T}}}(q) - \varepsilon.$$

THEOREM 14. *The average-time game on $\widetilde{\mathcal{T}}$ is determined, and for every $q \in \overline{Q}$, we have $\mathit{val}^{\widetilde{\mathcal{T}}}(q) = \mathit{val}^{\overline{\mathcal{T}}}(q)$.*

PROOF. Let $\mu^* \in \Xi_{\mathrm{Min}}$ be an optimal strategy of player Min in $\overline{\mathcal{T}}$. Let us fix an $\varepsilon > 0$.

$$\overline{\mathsf{val}}^{\widetilde{\mathcal{T}}}(q) = \inf_{\mu \in \widetilde{\Sigma}_{\mathrm{Min}}} \sup_{\chi \in \widetilde{\Sigma}_{\mathrm{Max}}} \mathcal{A}_{\mathrm{Min}}(q, \mu, \chi) \leq \sup_{\chi \in \widetilde{\Sigma}_{\mathrm{Max}}} \mathcal{A}_{\mathrm{Min}}(q, \mu^*_\varepsilon, \chi) \leq$$
$$\sup_{\chi \in \overline{\Sigma}_{\mathrm{Max}}} \mathcal{A}_{\mathrm{Min}}(q, \mu^*_\varepsilon, \chi) \leq \mathsf{val}^{\overline{\mathcal{T}}}(q) + \varepsilon.$$

The second inequality follows because $\mu^*_\varepsilon \in \widetilde{\Sigma}_{\mathrm{Min}}$ and the third inequality follows because $\widetilde{\Sigma}_{\mathrm{Max}} \subseteq \overline{\Sigma}_{\mathrm{Max}}$. The last inequality follows from Lemma 13 because $\mu^* \in \Xi_{\mathrm{Min}}$ is an optimal strategy in $\overline{\mathcal{T}}$. Similarly we show that for every $\varepsilon > 0$ we have that $\underline{\mathsf{val}}^{\widetilde{\mathcal{T}}}(q) \geq \mathsf{val}^{\overline{\mathcal{T}}}(q) - \varepsilon$. Hence it follows that $\mathsf{val}^{\widetilde{\mathcal{T}}}(q)$ exists and its value is equal to $\mathsf{val}^{\overline{\mathcal{T}}}(q)$. ∎

4.8 Determinacy of Average-Time Games on Timed Automata

THEOREM 15. *The average-time game on $\mathcal{T}$ is determined, and for every $s \in S$, we have:*

$$\mathit{val}^{\mathcal{T}}(s) = \mathit{val}^{\widetilde{\mathcal{T}}}(s, [s]) = \mathit{val}^{\overline{\mathcal{T}}}(s, [s]) = \mathit{val}^{\widehat{\mathcal{T}}}(s, [s]).$$

5 Complexity

The main decision problem for average-time game is as follows: given an average-time game $\Gamma = (\mathcal{T}, L_{\mathrm{Min}}, L_{\mathrm{Max}})$, a state $s \in S$, and a number $B \in \mathbb{R}_{\oplus}$, decide whether $\mathsf{val}(s) \leq B$.

From Theorem 15 we know that in order to solve an average-time game starting from an initial state of a timed automaton, it is sufficient to solve the average-time game on the set of states of the boundary region graph of the automaton that are reachable from the initial state. Observe that every region, and hence also every configuration of the game, can be represented in space polynomial in the size of the encoding of the timed automaton and of the encoding of the initial state, and that every move of the game can be simulated in polynomial time. Therefore, the value of the game can be computed by a straightforward alternating PSPACE algorithm, and hence the problem is in EXPTIME because APSPACE = EXPTIME.

One can prove EXPTIME-hardness of average-time games on timed automata with at least two clocks by a reduction from countdown games [10], similar to the reduction from countdown games to reachability-time games on timed automata [11].

THEOREM 16. *Average-time games are EXPTIME-complete on timed automata with at least two clocks.*

References

[1] R. Alur, M. Bernadsky, and P. Madhusudan. Optimal reachability for weighted timed games. In *International Colloquium on Automata, Languages and Programming, ICALP 2004*, volume 3142 of *LNCS*, pages 122–133. Springer, 2004.

[2] R. Alur and D. Dill. A theory of timed automata. In *Theoretical Computer Science*, volume 126, pages 183–235, 1994.

[3] E. Asarin and O. Maler. As soon as possible: Time optimal control for timed automata. In F. W. Vaandrager and J. H. van Schuppen, editors, *HSCC 1999*, volume 1569 of *LNCS*, pages 19–30. Springer, 1999.

[4] P. Bouyer, T. Brihaye, V. Bruyère, and J. Raskin. On the optimal reachability problem on weighted timed automata. *Formal Methods in System Design*, 31(2):135–175, 2007.

[5] P. Bouyer, E. Brinksma, and K. G. Larsen. Staying alive as cheaply as possible. In *Hybrid Systems: Computation and Control, HSCC 2004*, volume 2993 of *LNCS*, pages 203–218. Springer, 2004.

[6] P. Bouyer, F. Cassez, E. Fleury, and K. G. Larsen. Optimal strategies in priced timed game automata. In *FSTTCS'04*, volume 3328 of *LNCS*, pages 148–160. Springer, 2004.

[7] T. Brihaye, T. A. Henzinger, V. S. Prabhu, and J. Raskin. Minimum-time reachability in timed games. In *ICALP 2007*, volume 4596 of *LNCS*, pages 825–837. Springer, 2007.

[8] J. Filar and K. Vrieze. *Competitive Markov Decision Processes*. Springer, 1997.

[9] D. Gillette. Stochastic games with zero stop probabilities. In *Contributions to the Theory of Games*, volume 39 of *Annals of Mathematics Studies*, pages 179–187. Princeton University Press, 1957.

[10] M. Jurdziński, J. Sproston, and F. Laroussinie. Model checking probabilistic timed automata with one or two clocks. *Logical Methods in Computer Science*, 4(3):12, 2008.

[11] M. Jurdziński and A. Trivedi. Reachability-time games on timed automata. In *ICALP 2007*, volume 4596 of *LNCS*, pages 838–849. Springer, 2007.

[12] M. Jurdziński and A. Trivedi. Concavely-priced timed automata. In *FORMATS 2008*, volume 5215 of *LNCS*, pages 48–62. Springer, 2008.

[13] T. Liggett and S. Lipman. Stochastic games with perfect information and time average payoff. *SIAM Review*, 11:604–607, 1969.

[14] M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley, 1994.

[15] U. Zwick and M. Paterson. The complexity of mean payoff games on graphs. *Theoretical Computer Science*, 158:343–359, 1996.

Foundations of Software Technology and Theoretical Computer Science (Bangalore) 2008.
Editors: R. Hariharan, M. Mukund, V. Vinay; pp 352-356

About Models of security protocols (abstract)

Hubert Comon-Lundh
ENS Cachan and Research Center for Information Security
Advanced Industrial Science and Technology (AIST), Tokyo
h.comon-lundh@aist.go.jp

Framework

It is clear for everybody that security protocols need to be proven. Indeed, a security breach in one of these distributed programs, may have a dramatic impact. A typical example are the (electronic) voting protocols, which we fear will be soon widely used.

But what does "proven" mean? A formal proof requires a formal model, both for the protocol executions and for the security properties. There are however several such models and it may happen that a protocol is secure in one model and insecure in the other. Furthermore, there is no clear hierarchy in such models as the most accurate ones are not well-suited for formal proofs, because they require much too complicated proof steps, that are always performed in a very sketchy way.

There are two main classes of security models: the first one, later named *symbolic*, has been developed over years by the community of automated theorem proving and concurrency theory [22, 19, 3, 14, 2, 21]. The second one, later named *computational*, is more recent; it is an extension to protocols of the well-developed area of "provable security"[10, 7, 20, 17]. Symbolic models are much simpler, as an attacker is only given a fixed finite set of functionalities and there is no probabilistic choices or complexity issues. They are therefore better suited for (automated) formal proofs. However, because they are simpler, they might miss some attacks that rely on other attacker's capibilities, which are only considered in the computational model.

The relationship between these two classes of protocol models has been investigated in a series of recent works, starting with M. Abadi and Ph. Rogaway [4]. The idea is to explicit under which computational assumptions the symbolic models are *sound*; the soundness theorems show that reasoning in the symbolic model is sufficient: the extra attacker's capabilities are useless for mounting attacks.

Soundness results were first proven in the passive attacker case: such an attacker is not allowed to forge messages nor to send fake messages. Depending on the security primitives and the computational assumptions, there are several soundness results such as [4, 1, 9].

For protocol verification, it is however more relevant to consider an active attacker, who may control the communications. There is then a series of results showing a *trace mapping* property [18, 13, 15]. Roughly speaking, they show that a sequence of events in the computational model is, with overwhelming probability, also represented by a sequence of events in the symbolic model. Such trace mappings are showed for particular primitives and particular symbolic and computational models and assume computational properties

FSTTCS 2008
IARCS Annual Conference on
Foundations of Software Technology and Theoretical Computer Science
http://drops.dagstuhl.de/opus/volltexte/2008/1766

of the security primitives. Furthermore, on the security properties side, these results show that we may reason at the symbolic level, but only for some specific security properties, typically "trace properties".

Another series of soundness results are obtained in a series of papers on *simulatability* [7, 6, 8]. Simulatability implies trace mapping, as pointed out in [5], but the converse might not be true. Roughly speaking, what is missing in trace mapping is the *adaptative soundness* [16].

Finally, an even stronger result (which might be equivalent to simulatability) is the *soundness of observational equivalence* [11]: for a given set of cryptographic primitives, the authors show that trace mapping together with *tree soundness* implies that computational indistinguishability can be soundly abstracted as observational equivalence. This shows in particular, though in a different setting, what is missing in [5], in order to get a converse implication.

A few remarks on the state of the art

In all these works, the computational attacker is a polynomial time randomized interactive Turing machine. However, as discussed in [8, 17], there are several possible complexity notions for such machines. Also, using a Turing machine model yields often quite sketchy proofs, as the machines (in particular the simulators) are never constructed explicitly. Finally, there is no evidence that worst case polynomial time is an adequate complexity class, though it is convenient because of composition properties. It is actually not clear that Turing machines are an appropriate model. As a conclusion: we would like to abstract from this particular computation model.

On the formal model side, there are many schools, each promoting its own process calculus. There are also issues concerning the expressivity: are the result still valid if we consider protocols with branching tests? recursive protocols? We would like to avoid committing to a particular process calculus, while keeping the features that are essential for security definitions.

Concerning the relationships between the models, is there a general way of defining relationships between models? Is it possible to state something useful, independently of the models considered? Is there a methodology to decompose the tasks when proving a mapping property between two models?

Models of security protocols

In this paper, mostly consisting of definitions, we revisit the models of security protocols: we show that the symbolic and the computational models (as well as others) are instances of a same generic model. Our definitions are also parametrized by the security primitives, the notion of attacker and, to some extent, the process calculus.

We rely on a set of function symbols, predicate symbols and names (representing any randomized input), which are interpreted in some algebra. This can be a term algebra, or a computational algebra (as defined in [9]), whose domain is the set of bitstrings (or any other first-order structure on the given vocabulary).

A thread (also called a protocol role or a lightweight process) is any sequential program, generating data, receiving inputs from an environment and sending messages to the environment. The operational semantics of threads is defined through predicates that relate two consecutive states and a message (whether received or emitted). Again, this can be interpreted in a symbolic or computational world, depending on the interpretation structure that we consider.

Threads can be composed, using parallel composition (they may run concurrently), replication (a same program may be executed several times), name hiding and external inputs. This yields protocols.

Next, attackers are simply (deterministic) stateful functions that compute a message from a sequence of messages. They could be symbolic or computational and we may impose some restrictions, such as polynomial time computation bounds: this is again a model choice and we do not commit to any one in particular.

In order to define some asymptotic security notions, we need to include in the model families of probability distributions for the interpretation of names. In case of symbolic models, such distributions will be trivial: probabilities of events are either 0 or 1.

Finally, we define *indistinguishability*, without committing to any particular model: this yields for instance static equivalence in the case of a symbolic interpretation. This is also generalized to tree-indistinguishability, for families of term sequences.

Relationships between models

In this general setting, we define trace mapping: this is a relation between any two interpretations, each of which yielding some notion of possible sequences of events. In both interpretations, the attacker computes fake messages and send them to the network. However, he does not schedule the events according to his computation results. (The attacker is not "adaptative"). Among the models, the symbolic one has a universal property: there is always a trace mapping from the symbolic model to any other model. The converse implication depends however on the assumptions on the function interpretations.

The *tree soundness* property also relates two interpretations $\mathcal{M}_1$ and $\mathcal{M}_2$. This states that, if two trees, labeled with sequences of terms, are indistinguishable in the model $\mathcal{M}_1$, then they are also indistinguishable in the model $\mathcal{M}_2$. For this soundness notion, the attacker is adaptative, but cannot compute his own fake messages: he may only choose among the available directions.

In the most general case, the attacker is both allowed to compute fake messages and to schedule adaptatively the events. In that case, two programs (or protocols) are *observationally equivalent* if there is no attacker that can distinguish them. We show that relating observational equivalence in two models can be reduced to trace mapping and tree soundness in that models:

Trace mapping + Tree soundness $\Rightarrow$ Soundness of observational equivalence

This has been shown in [11], for a particular pair of models and process calculus. In [11] we further proved the trace mapping and the tree soundness for symmetric encryption, under some strong security assumptions.

Conclusion

We hope that revisiting the definitions will clarify what is relevant. We also believe that trace mapping and tree soundness are two (independent) relevant properties: this could be a guideline when trying to reduce security proofs in some model to symbolic security proofs. As a clue, the computational assumptions are often different for tree soundness and for trace mapping [12].

The main issue now is to decompose further the trace mapping property and the tree soundness property into more elementary tasks. Typically, we would like to get composition results, allowing to merge two sets of function symbols, instead of a having to restart from scratch each time we add a new primitive (which is the case in all current models).

Acknowledgments

I thank David Nowak for fruitful discussions.

References

[1] M. Abadi, M. Baudet, and B. Warinschi. Guessing attacks and the computational soundness of static equivalence. In L. Aceto and A. Ingólfsdóttir, editors, *FoSSaCS*, volume 3921 of *Lecture Notes in Computer Science*, pages 398–412. Springer, 2006.

[2] M. Abadi and C. Fournet. Mobile values, new names, and secure communication. In *Proceedings of the 28th ACM Symposium on Principles of Programming Languages (POPL'01)*, pages 104–115, January 2001.

[3] M. Abadi and A. Gordon. A calculus for cryptographic protocols: the spi calculus. *Information and Computation*, 148(1), 1999.

[4] M. Abadi and P. Rogaway. Reconciling two views of cryptography: the computational soundness of formal encryption. In *Proc. 1rst IFIP International Conference on Theoretical Computer Science*, volume 1872 of *Lecture Notes in Computer Science*, Sendai, Japan, 2000.

[5] M. Backes, M. Drmuth, and R. Ksters. On simulatability soundness and mapping soundness of symbolic cryptography. In *Proceedings of 27th International Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, December 2007.

[6] M. Backes and B. Pfitzmann. Symmetric encryption in a simulatable dolev-yao style cryptographic library. In *Proc. IEEE Computer Security Foundations workshop*, 2004.

[7] M. Backes, B. Pfitzmann, and M. Waidner. A composable cryptographic library with nested operations. In *Proc. 10th ACM Concerence on Computer and Communications Security (CCS'03)*, 2003.

[8] M. Backes, B. Pfitzmann, and M. Waidner. The reactive simulatability (rsim) framework for asynchronous systems. *Information and Computation*, 205(12), 2007.

[9] M. Baudet, V. Cortier, and S. Kremer. Computationally sound implementations of equational theories against passive adversaries. In *Proceedings of the 32nd International Colloquium on Automata, Languages and Programming (ICALP'05)*, volume 3580 of *Lecture Notes in Computer Science*, pages 652–663. Springer, July 2005.

[10] R. Canetti. Universal composable security: a new paradigm for cryptographic protocols. In *Proc. 42nd IEEE Symposium on Foundations of Computer Science*, 2001.

[11] H. Comon-Lundh and V. Cortier. Computational soundness of observational equivalence. In *Proc. ACM Conf. Computer and Communication Security (CCS)*, 2008.

[12] H. Comon-Lundh, Y. Kawamoto, and H. Sakurada. Symbolic and computational anonymity in an unbounded network. Submitted for publication.

[13] V. Cortier and B. Warinschi. Computationally sound, automated proofs for security protocols. In *Proc. 14th European Symposium on Programming (ESOP'05)*, volume 3444 of *Lecture Notes in Computer Science*, pages 157–171, 2005.

[14] F. T. Fabrega, J. Herzog, and J. Guttman. Strand spaces: Proving security protocol correct. *Journal of Computer Security*, 7:191–230, 1999.

[15] R. Janvier, Y. Lakhnech, and L. Mazaré. Completing the picture: Soundness of formal encryption in the presence of active adversaries. In *European Symposium on Programming (ESOP'05)*, volume 3444 of *Lecture Notes in Computer Science*, pages 172–185. Springer, 2005.

[16] S. Kremer and L. Mazaré. Adaptive soundness of static equivalence. In J. Biskup and J. Lopez, editors, *Proceedings of the 12th European Symposium on Research in Computer Security (ESORICS'07)*, volume 4734 of *Lecture Notes in Computer Science*, pages 610–625, Dresden, Germany, Sept. 2007. Springer.

[17] R. Küsters and M. Tuengerthal. Joint state theorems for public-key encryption and digital signature functionalities with local computations. In *Proc. IEEE Computer Security Foundations (CSF'08)*, 2008.

[18] D. Micciancio and B. Warinschi. Soundness of formal encryption in presence of an active attacker. In *Proc. Theory of Cryptography Conference (TCC'04)*, volume 2951 of *LNCS*, 2004.

[19] J. Millen and H. Rueß. Protocol independent secrecy. In *Proc. IEEE Symposium on Security and Privacy*, 2000.

[20] J. Mitchell, A. Ramanathan, and V. Teague. A probabilistic polynomial-time process calculus for the analysis of cryptographic protocols. *Theoretical Comput. Sci.*, 353:118–164, 2006.

[21] A. Roy, A. Datta, A. Derek, J. C. Mitchell, and J.-P. Seifert. Secrecy analysis in protocol composition logic. In *Proc. 11th Asian Computing Science Conference*, volume 4435 of *Lecture Notes in Computer Science*, Tokyo, Japan, Dec. 2006. Springer-Verlag.

[22] P. Ryan, S. Schneider, M. Goldsmith, G. Lowe, and B. Roscoe. *The Modelling and Analysis of Security Protocols*. Addison Wesley, 2000.

Foundations of Software Technology and Theoretical Computer Science (Bangalore) 2008.
Editors: R. Hariharan, M. Mukund, V. Vinay; pp 357-363

On Estimation Algorithms versus Approximation Algorithms

Uriel Feige[1]*
Weizmann Institute
Rehovot, Israel
uriel.feige@weizmann.ac.il

ABSTRACT.
In a combinatorial optimization problem, when given an input instance, one seeks a feasible solution that optimizes the value of the objective function. Many combinatorial optimization problems are NP-hard. A way of coping with NP-hardness is by considering approximation algorithms. These algorithms run in polynomial time, and their performance is measured by their approximation ratio: the worst case ratio between the value of the solution produced and the value of the (unknown) optimal solution.
In some cases the design of approximation algorithms includes a nonconstructive component. As a result, the algorithms become estimation algorithms rather than approximation algorithms: they allow one to estimate the value of the optimal solution, without actually producing a solution whose value is close to optimal.
We shall present a few such examples, and discuss some open questions.

1 Introduction

In a combinatorial optimization problem, when given an input instance, one seeks a feasible solution that maximizes (or minimizes) the value of the objective function. For example, in the Travelling Salesperson (TSP) problem, given an input graph with edge lengths, one is to find a tour (Hamiltonian cycle) of minimum length. Combinatorial optimization problems are very common in practice, and are also of great theoretical interest. Many combinatorial optimization problems are NP-hard (informally meaning that we know of no polynomial time algorithm that solves every instance optimally). A way of coping with NP-hardness is by considering *approximation algorithms*. These algorithms run in polynomial time (or sometimes, random polynomial time), but are not guaranteed to produce optimal solutions. Their performance is measured by their *approximation ratio*. For a maximization problem, an approximation algorithm is said to have approximation ratio $0 \leq \rho \leq 1$ if on every instance, the value of the solution output by the algorithm is at least ρ times the value of the optimal solution. (For minimization problems, $\rho \geq 1$, and the value of the solution output by the algorithm is at most ρ times the optimal.) It is often the case that the approximation ratio of an algorithm is not a fixed constant that holds for all input sizes n, but rather it deteriorates as the input size grows. In this case, rather than just saying that the approximation ratio is 0 (for maximization problems) or unbounded (for minimization problems), we measure the rate at which the the approximation ratio deteriorates (as a function of n). For example, the greedy algorithm for set cover has approximation ratio $\ln n$. The approximation ratio

*Supported in part by The Israel Science Foundation (grant No. 873/08)

of an optimization problem is the best approximation ratio achieved by any approximation algorithm for the problem. For more details see for example [24, 26, 6, 37].

We say that a combinatorial optimization problem has a *threshold* at ρ if there is a polynomial time (randomized) algorithm for it with approximation ratio ρ, and it is NP-hard to approximate it within a ratio better than ρ. (Here we ignore low order terms in the approximation ratio.) Problems that have approximation ratios arbitrarily close to 1 (a so called Polynomial Time Approximation Scheme, PTAS) have a threshold at 1. Perhaps surprisingly, many other problems (such as k-center, set cover, max coverage, max 3SAT) also have approximation thresholds, though the locations of the thresholds may differ among problems.

Needless to say, for many problems (such as metric TSP, max SAT, min bisection and dense k-subgraph) we do not know if they have a threshold or not. Problems with no known threshold are the ones relevant to the discussion that follows.

At this point it will be convenient to distinguish between notions that we shall call here *approximation* algorithms and *estimation* algorithms. For the approximation problem, one is required to find a feasible solution whose value is close to that of the value of the optimal solution. For estimation algorithms, one is required to estimate the value of the optimal solution, without necessarily outputting a solution that meets this estimate. This is potentially an easier task. It turns out that hardness of approximation results are essentially always also hardness of estimation results, within the same ratio. That is, our techniques for establishing hardness of approximation do not distinguish between approximation and estimation. On the algorithmic side, most positive results apply equally well to estimation and approximation. However, there are some exceptions where at the moment the known estimation ratios are better than the known approximation ratios.

2 Some research directions

The distinction between estimation algorithms and approximation algorithms offers interesting research directions.

Prove new estimation ratios. For some problems there are large gaps between the known approximation ratios and the known hardness of approximation results. For such problems, try to establish estimation ratios that are better than the known approximation ratios.

Close the gaps between estimation and approximation ratios. For some problems there are large gaps between the known approximation ratios and the known estimation ratios. For such problems, try to improve the approximation ratio (hopefully, replacing the nonconstructive arguments that lead to the estimation ratios by constructive arguments that lead to the same approximation ratio).

Relating between open questions. Introduce complexity classes that capture current gaps between estimation and approximation (similar in spirit to the work of [32]). That is, we would like to be able to show that if this gap is closed for one problem, this automatically implies that the gap will be closed for other problems.

Relating to external open questions. At the moment we do not have convincing evidence that there should be a gap between approximation ratios and estimation ratios. For

many optimization problems these ratios provably match (when there is a known approximation threshold, such as for max-3SAT or min set cover), in others they currently match (such as for min vertex cover or sparsest cut), and in the remaining cases the theory of NP-completeness does not appear to apply, because it deals with decision problems rather than search problems. Try to establish connections between previously defined concepts (such as PPAD-completeness) and gaps between approximation and estimation. (To appreciate the subtleties involved consider the following example. Finding a locally maximal cut is PLS-complete, but the known approximation ratios for max-cut [25] are better than those that local search gives. Hence PLS-completeness by itself is not an obstacle to bridging the gap between estimation and approximation.)

Development of techniques. There are some proof techniques that originally were nonconstructive, and algorithmic versions of them (or of special cases) were discovered only later. See for example [7] for the local lemma and [4] for the regularity lemma. Design algorithmic versions of nonconstructive arguments, regardless of any immediate applicability to combinatorial optimization.

Random instances. Nonconstructive arguments often show that random instances (such as random 3CNF formulas) are likely to either have or not have solutions (depending on the density of the underlying instance). Find algorithmic versions of these results. These type of questions have indirect connections to approximation algorithms, and may well require similar sets of techniques (see [15] for example).

3 Examples

Below we list some examples of current gaps between approximation ratios and estimation ratios (or conjectured estimation ratios).

Max-min allocation.

In max-min allocation problems, there is a threshold t, a set of m items, a set of n players, and nonnegative valuations v_{ij} that for every player i and item j specify the value of item j to player i. The goal is to allocate items to the players in a way that every player gets total value (sum of his values for the items allocated to him) at least t. This problem is NP-hard. A linear program relaxation of this problem provides an upper bound on the maximum possible value of t. It is known that the gap between this upper bound and true optimum may be $\Omega(\sqrt{n})$. However, in an interesting special case, the *restricted assignment* version, there is a nonconstructive proof (in fact, two different nonconstructive proofs by now, [18] using the local lemma, [5] using local search) that the gap is at most constant. Hence the value of the linear program provides a constant factor estimation for the restricted assignment version of the max-min allocation problem. No constant factor approximation ratio is known for this problem.

Metric TSP.

The Held-Karp conjecture states that the value of a certain linear program provides a $4/3$ estimation for metric TSP in undirected graphs. If true, this conjecture provides a $4/3$ estimation ratio for metric TSP, which is better than the known approximation ratio of $3/2$.

For undirected graphs it is known that the integrality gap of the LP is no better than $4/3$ and no worse than $3/2$. For directed graphs, the integrality gap is known to be no better

than 2, and no sublogarithmic approximation ratios are known.

Edge colorings in multigraphs.

There is a famous theorem by Vizing that states (and gives an algorithm) that in every simple graph there is a legal edge coloring with one more color than the maximum degree in the graph. This gives an approximation of the edge chromatic number within additive 1. It was conjectured (e.g., by Seymour) that a similar result can be extended to multigraphs, using a linear programming relaxation. If true this would provide an estimation algorithm for the edge chromatic number within an additive error of 1. There are nonconstructive proofs (using the local lemma) that give a $1 + \epsilon$ multiplicative estimation when the edge chromatic number of multigraphs is sufficiently large [29].

Discrepency.

Many discrepancy problems can be viewed as coloring problems on hypergraphs. The goal is to color the vertices such that every hyperedge remains nearly balanced (has roughly the same number of vertices of each color). Techniques used in the proofs that low discrepancy colorings exist are sometimes constructive (such as the Beck-Fiala theorem that iteratively uses basic feasible solutions of linear programs), and sometimes nonconstructive (such as the first use of the Lovasz local lemma, or Spencer's proof that "six standard deviations suffice" that uses the *pigeon hole principle* in a nonconstructive way). The reader is referred to [9, 31] were references to these and other results can be found. In general, it is often the case that statements involving discrepancy involve nonconstructive proofs (see also [2, 16]).

It would be desirable to replace some of the nonconstructive proofs in discrepancy theory by algorithmic proofs (as was done by Beck in the context of the local lemma). Perhaps more ambitiously, improve some of the known discrepancy bounds. (For example, it is conjectured that the Beck-Fiala theorem can be improved when the degrees are large.)

Graph bandwidth.

A linear arrangement of a graph is a numbering of its n vertices from 1 to n. The bandwidth of the linear arrangement is the maximum difference between numberings of endpoints of an edge. The *bandwidth* of a graph is the bandwidth of its minimum bandwidth linear arrangement. The *local density* of a graph is a natural lower bound on the bandwidth. It is known that the gap between bandwidth and local density can be $\Omega(\log n)$, and there is an algorithm that finds a linear arrangement of bandwidth $O(\log^{3.5} n)$ times the local density [14]. It is reasonable to conjecture that the maximum ratio between bandwidth and local density is $O(\log n)$. If true, then local density provides an $O(\log n)$ estimation ratio for the bandwidth. The best approximation ratio known for the bandwidth is currently $O(\log^3 n)$ [13].

Random 3CNF.

Work on refuting dense random 3CNF formulas offers a lot of interplay between existential and algorithmic arguments. For example, it is shown in [20] that formulas of density above $n^{0.4}$ are likely to have polynomial size witnesses for nonsatisfiability. There is no known efficient algorithm for finding these witnesses. Or another example, the notion of *even covers*, originally studied in coding theory, is used in [20, 17] as part of refutation algorithms and witnesses. Further progress is hampered because we are missing an existential result – we do not know how to prove that small even covers must exist at densities below

$\sqrt{n}$, and because we are missing an algorithmic result – we do not know how to find small even covers when they do exist.

4 Conclusions

The list of references is not based on a careful study of all related references. Hence it may miss some important references, and include some papers whose relevance to this manuscript is questionable. A short overview of the topics addressed by some of the references is provided.

A well known nonconstructive proof technique is the Lovasz local lemma (see for example [3]). It had been used in the design of estimation algorithms [30, 22, 19, 18]. In some cases, algorithmic versions of the local lemma are known [7, 12].

The use of linear programming relaxations is common in approximation algorithms. Sometimes general principles (such as the existence of basic feasible solutions) can be used in order to show show the existence of high quality integer solutions (as in [8]). In some cases the underlying linear programs are of exponential size (as in [2, 16]). These lead naturally to estimation algorithms rather than approximation algorithms. Sometimes, the result inferred from the exponential LP may be obtained by a more direct efficient algorithm (see [23] for one such example), leading to approximation algorithms.

In the context of random instances of CNF formulas there are many nonconstructive arguments that lack a constructive counterpart. See examples of work in this area in [1, 11, 15, 17, 20, 21].

Local search is a common algorithmic tool that does not always lead to polynomial time algorithms [27, 28, 33, 35]. When used for optimization problems, it might result in estimation algorithms rather than approximation algorithms [5].

There are certain complexity classes that attempt to capture nonconstructive principles. See [32, 10] for example.

In the context of counting problems [36] there are many randomized approximation algorithms (such as [34]). In our terminology, we would view them as estimation algorithms rather than approximation algorithms, since they are only required to output an estimation for the number of solutions, rather than to list the solutions (which in typical situations would require exponential output size).

In conclusion, the distinction between approximation and estimation algorithms has been an explicit or implicit part of research for many years. The purpose of this manuscript is to bring this distinction and the research opportunity that it offers to the awareness of more researchers.

References

[1] Dimitris Achlioptas, Yuval Peres. The threshold for random k-SAT is $2k(\ln 2 - O(k))$. *STOC 2003: 223–231*.

[2] Gagan Aggarwal, Amos Fiat, Andrew V. Goldberg, Jason D. Hartline, Nicole Immorlica, Madhu Sudan. Derandomization of auctions. *STOC 2005: 619–625*.

[3] N. Alon, J. Spencer. *The probabilistic Method.* Wiley Interscience.

[4] Noga Alon, Richard A. Duke, Hanno Lefmann, Vojtech Rodl, Raphael Yuster. The Algorithmic Aspects of the Regularity Lemma. *J. Algorithms 16(1): 80–109 (1994)*.
[5] A. Asadpour, U. Feige, A. Saberi. Santa Claus Meets Hypergraph Matchings. *Proceedings of APPROX-RANDOM 2008: 10–20*.
[6] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, M. Protasi. *Complexity and Approximation*. Springer Verlag, 1999.
[7] J. Beck. An algorithmic approach to the Lovasz Local Lemma. *Random Structures and Algorithms, 2 (1991), pp, 343–365*.
[8] J. Beck and T. Fiala. "Integer-making" theorems. *Discrete Applied Mathematics, 3:1–8, 1981*.
[9] B. Chazelle. *The Discrepancy Method: Randomness and Complexity*. Cambridge University Press, 2000
[10] Xi Chen, Xiaotie Deng. Settling the Complexity of Two-Player Nash Equilibrium. *FOCS 2006: 261–272*.
[11] Amin Coja-Oghlan, Andreas Goerdt, Andre Lanka: Strong Refutation Heuristics for Random k-SAT. *APPROX-RANDOM 2004: 310-321*.
[12] Artur Czumaj, Christian Scheideler. A new algorithm approach to the general Lovasz local lemma with applications to scheduling and satisfiability problems (extended abstract). *STOC 2000: 38–47*.
[13] John Dunagan, Santosh Vempala. On Euclidean Embeddings and Bandwidth Minimization. *RANDOM-APPROX 2001: 229-240*.
[14] Uriel Feige. Approximating the bandwidth via volume respecting embeddings. *Journal of Computer and System Sciences*, 60(3), 510–539, 2000.
[15] Uriel Feige. Relations between average case complexity and approximation complexity. *STOC 2002: 534-543.*
[16] Uriel Feige. You can leave your hat on (if you guess its color). *Technical report MCS04-03 of the Weizmann Institute, 2004.*
[17] Uriel Feige. Refuting smoothed 3CNF formulas. *Proc. of 48th FOCS*, 2007, 407–417.
[18] Uriel Feige. On Allocations that Maximize Fairness. *SODA 2008, 287–293*.
[19] Uriel Feige, Magnus M. Halldorsson, Guy Kortsarz, Aravind Srinivasan. Approximating the Domatic Number. *SIAM J. Comput. 32(1): 172-195 (2002).*
[20] Uriel Feige, Jeong Han Kim, Eran Ofek. Witnesses for non-satisfiability of dense random 3CNF formulas. *FOCS 2006: 497–508*.
[21] Uriel Feige, Eran Ofek. Easily Refutable Subformulas of Large Random 3CNF Formulas. *ICALP 2004: 519–530.*
[22] Uriel Feige, Christian Scheideler. Improved Bounds for Acyclic Job Shop Scheduling. *Combinatorica 22(3): 361–399 (2002).*
[23] Uriel Feige, Jan Vondrak. Approximation algorithms for allocation problems: Improving the factor of 1 - 1/e. *FOCS 2006: 667–676.*
[24] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.
[25] Michel X. Goemans, David P. Williamson. Improved Approximation Algorithms for Maximum Cut and Satisfiability Problems Using Semidefinite Programming. *J. ACM 42(6): 1115–1145 (1995).*

[26] Dorit Hochbaum (Ed.). *Approximation Algorithms for NP-hard Problems*. PWS Publishing Company, 1997.
[27] David S. Johnson, Christos H. Papadimitriou, Mihalis Yannakakis. How Easy is Local Search? *J. Comput. Syst. Sci. 37(1): 79–100 (1988).*
[28] Gil Kalai. Upper Bounds for the Diameter and Height of Graphs of Convex Polyhedra. *Discrete and Computational Geometry 8: 363–372 (1992).*
[29] Jeff Kahn. Asymptotics of the Chromatic Index for Multigraphs. *J. Comb. Theory, Ser. B 68(2): 233–254 (1996).*
[30] Frank Thomson Leighton, Bruce M. Maggs, Satish Rao. Packet Routing and Job-Shop Scheduling in O(Congestion + Dilation) Steps. *Combinatorica 14(2): 167-186 (1994).*
[31] J. Matousek. *Geometric Discrepancy*. Springer 1999.
[32] Christos H. Papadimitriou. On Graph-Theoretic Lemmata and Complexity Classes. *FOCS 1990: 794–801*.
[33] Christos H. Papadimitriou, Alejandro A. Schaeffer, Mihalis Yannakakis. On the Complexity of Local Search. *STOC 1990: 438–445.*
[34] Alistair Sinclair, Mark Jerrum. Approximate Counting, Uniform Generation and Rapidly Mixing Markov Chains. *Inf. Comput. 82(1): 93-133 (1989)*.
[35] Mihalis Yannakakis. The Analysis of Local Search Problems and Their Heuristics. *STACS 1990: 298–311.*
[36] Leslie G. Valiant. The Complexity of Enumeration and Reliability Problems. *SIAM J. Comput. 8(3): 410–421 (1979).*
[37] Vijay Vazirani. *Approximation Algorithms.* Springer 2001.

Foundations of Software Technology and Theoretical Computer Science (Bangalore) 2008.
Editors: R. Hariharan, M. Mukund, V. Vinay; pp 364-382

Banach-Mazur Games on Graphs

Erich Grädel
RWTH Aachen University
graedel@logic.rwth-aachen.de

ABSTRACT. We survey determinacy, definability, and complexity issues of Banach-Mazur games on finite and infinite graphs.
Infinite games where two players take turns to move a token through a directed graph, thus tracing out an infinite path, have numerous applications in different branches of mathematics and computer science. In the usual format, the possible moves of the players are given by the edges of the graph; in each move a player takes the token from its current position along an edge to a next position. In Banach-Mazur games the players instead select in each move a *path* of arbitrary finite length rather than just an edge. In both cases the outcome of a play is an infinite path. A winning condition is thus given by a set of infinite paths which is often specified by a logical formula, for instance from S1S, LTL, or first-order logic.
Banach-Mazur games have a long tradition in descriptive set theory and topology, and they have recently been shown to have interesting applications also in computer science, for instance for planning in nondeterministic domains, for the study of fairness in concurrent systems, and for the semantics of timed automata.
It turns out that Banach-Mazur games behave quite differently than the usual graph games. Often they admit simpler winning strategies and more efficient algorithmic solutions. For instance, Banach-Mazur games with ω-regular winning conditions always have positional winning strategies, and winning positions for finite Banach-Mazur games with Muller winning condition are computable in polynomial time.

1 Banach-Mazur Games

Game playing is a powerful metaphor that fits situations in which interaction between autonomous agents plays a central role. Indeed, numerous problems in computer science and other fields can be understood, mathematically treated, and solved in terms of appropriate mathematical models of games. There is of course a large variety of game models, leading to vastly different mathematical theories of games.

A prominent class of games, which is particularly useful for problems such as the synthesis and verification of interactive systems (with non-terminating behaviour and ongoing interaction between system and environment), or for the evaluation of fixed point logics and other important specification formalisms, are infinite games, where two players take turns to move a token through a directed graph thus tracing out an infinite path. The objectives of the players are given by suitable properties of infinite paths, often specified by logical formulae, for instance from monadic second order logic (S1S), linear-time temporal logic (LTL), or first-order logic (FO). Some central mathematical questions concerning such games are: Which games are determined (in the sense that from each position, exactly one player has a winning strategy)? How to compute winning positions? Are there optimal strategies, and if so, what is their complexity and how to compute them efficiently? How much knowledge of the play history is necessary to compute an optimal next action? In what logical formalisms can we define winning positions and winning strategies? And so on.

FSTTCS 2008
IARCS Annual Conference on
Foundations of Software Technology and Theoretical Computer Science
http://drops.dagstuhl.de/opus/volltexte/2008/1768

These questions are not just of theoretical interest. They are in fact standard design and verification problems (of interactive systems) in purified form. For background on such methodologies, based on the interplay between logic, automata, and games, see e.g. [8].

In the usual format of infinite games on graphs, the possible moves of the players are given by the edges of the graph; in each move a player takes the token from its current position along an edge to a next position. Here we study a different variant of graph games where, in each move, the players select a path of arbitrary finite length rather than just an edge. We call these games Banach-Mazur games on graphs.

DEFINITION 1. A Banach-Mazur game $\mathrm{BM}(G, v, \mathrm{Win})$ is given by an a directed graph $G = (V, E)$ without terminal nodes, an initial position $v \in V$, and a winning condition $\mathrm{Win} \subseteq \mathrm{Paths}(G, v)$ where $\mathrm{Paths}(G, v) \subseteq V^\omega$ denotes the set of infinite paths through G that start at node v.

The game $\mathrm{BM}(G, v, \mathrm{Win})$ is played by two players, called Player 0 and Player 1. In the opening move, Player 0 selects a finite, non-empty path x_0 from v through G. The players take turns, extending in each move the finite path $x_0 x_1 \dots x_{m-1}$ played so far by a new segment x_m (which again has to be a non-empty and finite path). In an infinite number of moves, the players thus trace out an infinite path $\pi \in \mathrm{Paths}(G, v)$. Player 0 wins the play, if $\pi \in \mathrm{Win}$, otherwise Player 1 wins.

In somewhat different forms, Banach-Mazur games have been extensively studied in descriptive set theory (see [13, Chapter 6] or [14, Chapter 8.H]) and topology (see e.g. [21]). In their original variant (see [15, pp. 113–117]), the winning condition is a set W of real numbers; in the first move, one of the players selects an interval d_1 on the real line, then her opponent chooses an interval $d_2 \subset d_1$, then the first player selects a further refinement $d_3 \subset d_2$ and so on. The first player wins if the intersection $\bigcap_{n\in\omega} d_n$ of all intervals contains a point of W, otherwise her opponent wins.

A similar game can be played on any topological space. Let $\mathcal{V}$ be a family of subsets of a topological space X such that each $V \in \mathcal{V}$ contains a non-empty open subset of X, and each nonempty open subset of X contains an element $V \in \mathcal{V}$. In the Banach-Mazur game defined on $X, \mathcal{V}$ with winning winning condition $W \subseteq X$, the players take turns to choose sets $V_0 \supset V_1 \supset V_2 \supset \dots$ in $\mathcal{V}$, and Player 0 wins the play if $\bigcap_{n<\omega} V_n \cap \mathrm{Win} \neq \emptyset$. We refer to [21] for a survey on topological games and their applications to set-theoretical topology. Notice that Banach-Mazur games on graphs are just a special case of this general topological setting. Indeed, the set $\mathrm{Paths}(G, v)$ of infinite paths through G from v is a topological space whose basic open sets are $\mathcal{O}(x)$, the sets of infinite prolongations of some finite path $x \in \mathrm{FinPaths}(G, v)$. Thus, when a player prolongs the finite path x played so far to a new path xy, she reduces the set of possible outcomes of the play from $\mathcal{O}(x)$ to $\mathcal{O}(xy)$, and she wins an infinite play $x_0 x_1 \dots$ if, and only if $\bigcap_{n<\omega} \mathcal{O}(x_0 \dots x_{n-1}) \cap \mathrm{Win} \neq \emptyset$.

Applications of Banach-Mazur games. Banach-Mazur games on graphs have recently appeared in several application areas in computer science. Pistore and Vardi used a variation of Banach-Mazur games for planning in nondeterministic domains [20]. In their scenario, the desired infinite behaviour of a system, which should be enforced by a plan, is specified by formulae in linear temporal logic LTL. It is assumed that the outcome of actions may be

nondeterministic. Hence a plan does not have only one possible execution path in the planning domain, but an execution tree. Between weak planning (some possible execution path satisfies the specification) and strong planning (all possible outcomes are consistent with the specification), there is a spectrum of intermediate cases such as strong cyclic planning: every possible partial execution of the plan can be extended to an execution reaching the desired goal. In this context, planning can be modelled by a game between a friendly player E and a hostile player A selecting the outcomes of nondeterministic actions. The game is played on the execution tree of the plan, and the question is whether the friendly player E has a strategy to ensure that the outcome (a path through the execution tree) satisfies the given LTL-specification. In contrast to the general scenario of Banach-Mazur games, the main interest here are games with finitely many alternations between players. Pistore and Vardi show that the planning problems in this context can be solved by automata-based methods in 2EXPTIME.

Banach-Mazur games appear also in the characterisation of fair behaviour in concurrent systems. There are many different notions of fairness. A very convincing one [23] defines a fairness property in a transition system as a set of (infinite) runs that is topologically large (co-meager). This is equivalent to say that, in an associated Banach-Mazur game, the first player (the scheduler) has a winning strategy to ensure fairness. It is a consequence of the positional determinacy of Banach-Mazur games with ω-regular winning conditions (see Theorem 17 below) that, on finite graphs, ω-regular fairness properties coincide with ω-regular properties that are probabilistically large under positive Markov measures. Hence , any ω-regular fairness property has probability one under randomised scheduling. As a further consequence, one can use results about finite Markov chains for checking whether a finite system is fairly correct with respect to LTL or ω-regular specifications.

Finally, Banach-Mazur games have recently been used to describe the semantics of timed automata [1, 2]. Timed automata are an important model for verification, but for many purposes, its idealized mathematical features such as infinite precision, instantaneous events lead to violations of specifications due to unlikely sequences of events. Therefore alternative semantics for the satisfaction of LTL specifications have been proposed, based on probability or on topological largeness, to rule out unlikely runs. By means of Banach-Mazur games, it has been established, that the two semantics coincide.

Here we study Banach-Mazur games on graphs, and focus on the above-mentioned central mathematical questions, such as determinacy, the structure and algorithmic properties of winning strategies, and the definability of winning regions.

Acknowledgement. This survey is based on joint research with Dietmar Berwanger and Stephan Kreutzer.

2 Topology and determinacy

For any arena (G, v) of a Banach-Mazur game, the space $\mathrm{Paths}(G, v)$ is endowed with a topology whose basic open sets are $\mathcal{O}(x)$, the sets of infinite prolongations of some finite path $x \in \mathrm{FinPaths}(G, v)$. A set $X \subseteq \mathrm{Paths}(G, v)$ is *open* if it is a union of basic open sets $\mathcal{O}(x)$, i.e., if $X = W \cdot V^\omega \cap \mathrm{Paths}(G, v)$ for some set $W \subseteq V^*$. A tree $T \subseteq \mathrm{FinPaths}(G, v)$ is a

set of finite paths that is closed under prefixes. It is easily seen that $X \subseteq \mathrm{Paths}(G, v)$ is *closed* (i.e., the complement of an open set) if, and only if, it is the set of infinite branches of some tree T, denoted $X = [T]$. Notice that $\mathrm{Paths}(G, v)$ itself is a closed set in the space V^ω, the set of all infinite sequences on V.

The class of *Borel sets* is the closure of the open sets under countable union and complementation. Borel sets form a natural hierarchy of classes $\mathbf{\Sigma}^0_\eta$ for $1 \leq \eta < \omega_1$, whose first levels are $\mathbf{\Sigma}^0_1$ (or G), the collection of all open sets, $\mathbf{\Pi}^0_1$ (or F), the closed sets, $\mathbf{\Sigma}^0_2$ (or F_σ), the countable unions of closed sets, and $\mathbf{\Pi}^0_2$ (or G_δ), the countable intersections of open sets. In general, $\mathbf{\Pi}^0_\eta$ contains the complements of the $\mathbf{\Sigma}^0_\eta$-sets, $\mathbf{\Sigma}^0_{\eta+1}$ is the class of countable unions of $\mathbf{\Pi}^0_\eta$-sets, and $\mathbf{\Sigma}^0_\lambda = \bigcup_{\eta<\lambda} \mathbf{\Sigma}^0_\eta$ for limit ordinals λ.

We recall that a set X in a topological space is *dense*, if its intersection with every (basic) non-empty open set is non-empty.

LEMMA 2. *For any strategy g of Player 1 in a Banach-Mazur game on a graph (G, v), the set $\mathrm{Plays}(g)$ of all plays that are consistent with g is a countable intersection of dense open sets.*

PROOF. Clearly, $\mathrm{Plays}(g) = \bigcap_{n\in\omega} \mathrm{Plays}_n(g)$ where $\mathrm{Plays}_n(g)$ is the set of all plays that may arise if Player 1 moves according to g during her first n moves. Obviously, $\mathrm{Plays}_n(g)$ is open. But it is also dense, since every finite path x can be used by Player 0 as her opening move, so there must be a prolongation of x in $\mathrm{Plays}_n(g)$, which means that $\mathcal{O}(x) \cap \mathrm{Plays}_n(g) \neq \emptyset$. ∎

Notice that, if $X \subseteq \mathrm{Paths}(G, v)$ is a dense open set, then any finite path x has a finite prolongation xy such that $\mathcal{O}(xy) \subseteq X$. In a topological sense, the dense open sets are large sets, and so is any countable intersection of such. Hence, by any strategy in a Banach-Mazur game, Player 1 can exclude only a topologically small set of plays. This means that she can only have a winning strategy if the set Win of winning plays for Player 0 is small, and her own set of winning plays, $\mathrm{Paths}(G, v) \setminus W$, is large.

For strategies of Player 0, the situtation is slightly different, since she starts the play. Hence, for any strategy f of Player 0, $\mathrm{Plays}(f) \subseteq \mathcal{O}(x)$ where x is the opening move by f. After the first move, the remaining game is one where the role of the players have been switched (i.e. Player 1 now moves first). By the same argument as in the previous lemma we infer that the set of plays consistent with a strategy of Player 0 is large inside some basic open set of plays.

LEMMA 3. *For any strategy f of Player 0 in a Banach-Mazur game, $\mathrm{Plays}(f)$ is a countable intersection of dense open subsets of $\mathcal{O}(x)$, where x is the opening move by f.*

The observations that we made on the set of plays that are consistent with strategies in Banach-Mazur games give a quite precise characterisation, in term of topological notions, of the games for which Players 0 and 1 have winning strategies.

A set in a topological space is *nowhere dense* if it is not dense in any open set or, equivalently, if its complement contains a dense open set. A set is *meager* (or topologicaly small) if it is a union of countably many nowhere dense sets, and *co-meager* (or topologically large) if its complement is meager. A topological space is called a *Baire space* if no non-empty set is both open and meager, or equivalently, if any countable intersection $X = \bigcap_{n<\omega} X_n$ of dense open sets X_n is dense. The spaces $\mathrm{Paths}(G, v)$ are Baire spaces since, for any finite path x,

we find an infinite extension $xy_0y_1\cdots \in X$ by choosing, for each n, a finite prolongation $xy_0\dots y_n$ of $xy_0\dots y_{n-1}$ such that $\mathcal{O}(xy_0\dots y_n) \subseteq X_n$. In Baire spaces a set is co-meager if, and only if, it contains a dense $\mathbf{\Pi}^0_2$ set.

Hence we have shown that, in Banach-Mazur games, $\mathrm{Plays}(g)$ is co-meager for every strategy g of Player 1 and $\mathrm{Plays}(f)$ is co-meager in some basic open set for every strategy f of Player 0. Conversely, for any meager set $W \subseteq \mathrm{Paths}(G,v)$, Player 1 has a strategy g such that $\mathrm{Plays}(G) \cap W = \emptyset$. Indeed, if $W = \bigcup_{n<\omega} X_n$ with X_n nowhere dense, then in her n-th move, Player 1 prolongs the path constructed so far to a path x_n such that $\mathcal{O}(x_n) \cap X_n = \emptyset$ which is always possible since the complement of X_n contains a dense open set. Clearly every play consistent with this strategy avoids W. Analogously, for every set that is co-meager in some basic open set, Player 0 has a strategy f such that $\mathrm{Plays}(f) \subseteq W$.

Our observations are summarized by the Banach-Mazur-Theorem which gives a precise characterisation of the games where Player 0 or Player 1 has a winning strategy.

THEOREM 4. *[Banach-Mazur]*

(1) Player 1 *has a winning strategy for the game* $\mathrm{BM}(G,v,\mathrm{Win})$ *if, and only if,* $\mathrm{Win} \subseteq \mathrm{Paths}(G,v)$ *is meager.*

(2) Player 0 *has a winning strategy for* $\mathrm{BM}(G,v,\mathrm{Win})$ *if, and only if, there exists a finite path* $x \in \mathrm{FinPaths}(G,v)$ *such that* $\mathcal{O}(x) \setminus \mathrm{Win}$ *is meager in* $\mathrm{Paths}(G,v)$ *(i.e.,* Win *is co-meager in some basic open set).*

This result appears, in different terms, in the Scottish Book [15, Problem 43] where it is mentioned as a conjecture due to Mazur, with an addendum by Banach, dated August 4, 1935 saying that "Mazur's conjecture is true". The Banach-Mazur-Theorem was published for the first time by Mycielski, Świerczkowski, and Zieba [18], without proof; the first published proof is due to Oxtoby [19].

¿From Theorem 4 we easily get strong results on determinacy of Banach-Mazur games.

COROLLARY 5. *Every Banach-Mazur game* $\mathrm{BM}(G,v,\mathrm{Win})$ *such that* $\mathrm{Win} \subseteq \mathrm{Paths}(G,v)$ *has the Baire property is determined.*

Recall that a set X in a topological space has the *Baire property* if its symmetric difference with some open set is meager. Since Borel sets have the Baire property, it follows that Banach-Mazur games are determined for Borel winning conditions. Standard winning conditions used in computer science applications (in particular the ω-regular winning conditions) are contained in very low levels of the Borel hierarchy.

A converse to Corollary 5 in terms of specific games does not hold. Indeed one can construct determined games with winning conditions of arbitrary complexity by combining a trivial game won by Player 0 with an arbitrarily complex game in such a way that Player 0 can avoid the complicated part.

A more interesting question is whether one can prove a converse for winning conditions that *guarantee determinacy* in the following sense. Let $W \subseteq C^\omega$ be a set of infinite words on some alphabet C. On every graph $G = (V,E)$ equipped with a function $\Omega : V \to C$, the set W defines a winning condition $\Omega^{-1}(W) := \{\pi \in \mathrm{Paths}(G,v) : \Omega(\pi) \in W\}$. We then say that W guarantees determinacy for Banach-Mazur games if *all* games with a winning condition $\Omega^{-1}(W)$ are determined.

We can link the Baire property with the determinacy of Banach-Mazur game in the following class-wise sense.

THEOREM 6. *For every class $\Gamma \subseteq \mathcal{P}(C^\omega)$ the following are equivalent.*
(1) All winning conditions $W \in \Gamma$ guarantee determinacy for Banach-Mazur.
(2) All sets $W \in \Gamma$ have the Baire property.

PROOF. If $W \subseteq C^\omega$ has the Baire property then so has $\Omega^{-1}(W)$, for all functions $\Omega : V \to C$ that label the nodes of a graph $G = (V, E)$ with elements of C. Thus, by Corollary 5 W guarantees determinacy.

For the converse, suppose that $W \subseteq C^\omega$ does not have the Baire property. To construct a non-determined game, let $G(C)$ be the complete directed graph on C itself (and let Ω be the identity function on C). We do not use W directly as a winning condition, but modify it as follows. Let $S := \{x \in C^* : \mathcal{O}(x) \setminus W \text{ is meager}\}$ and let Z be the symmetric difference of W with the open set $Y = \bigcup_{x \in S} \mathcal{O}(x)$.

We claim that the Banach-Mazur game on $G(C)$ with winning condition Z is not determined. Since Z is the symmetric difference of W with an open set, it cannot be meager (otherwise W would have the Baire property), hence Player 1 does not have a winning strategy. So suppose that Player 0 has a winning strategy. This can only happen if Z is co-meager in some basic open set $\mathcal{O}(x)$. For $x \in S$, this is impossible since $\mathcal{O}(x) \cap Z = \mathcal{O}(x) \setminus W$ is meager. Hence $x \in C^* \setminus S$. But then $\mathcal{O}(x) \cap Y = \emptyset$. Otherwise we would have some $y \in S$ such that $\mathcal{O}(x) \cap \mathcal{O}(y) \neq \emptyset$, which means that $\mathcal{O}(x) \subseteq \mathcal{O}(y)$ or $\mathcal{O}(y) \subseteq \mathcal{O}(x)$. In either case, since $\mathcal{O}(y) \cap Z = \mathcal{O}(y) \setminus W$ is meager, Z cannot be co-meager in $\mathcal{O}(x)$.

Now, since $\mathcal{O}(x) \cap Y = \emptyset$, we have $\mathcal{O}(x) \cap Z = \mathcal{O}(x) \cap W$, and if this set were co-meager in $\mathcal{O}(x)$ then $x \in S$, a contradiction.

Thus, none of the players has a winning strategy. ∎

A specific example of a non-determined Banach-Mazur game can be obtained by modifying a well-known construction on the basis of ultrafilters. Let G_2 be the complete directed graph with vertices 0,1, and for any set $U \subseteq \mathcal{P}(\omega)$, let let W_U be the set of infinite sequences $x_0x_1x_2 \cdots \in \{0,1\}^\omega$ such that $\{n : x_n = 0\} \in U$.

An ultrafilter over ω is a set $U \subseteq \mathcal{P}(\omega)$ that does not contain $\emptyset$, that includes with any set also all its supersets, with any two sets also their intersection, and such that for any set $x \subseteq \omega$ either $x \in U$ or $\omega \setminus x \in U$. An ultrafilter is free if it contains all co-finite sets. As a consequence, it does not contain any finite set. The Boolean Prime Ideal Theorem (a weak form of the Axiom of Choice) implies that free ultrafilters exist.

PROPOSITION 7. *If U is a free ultrafilter, then the Banach-Mazur game on G_2 with winning condition W_U is not determined.*

PROOF. Without loss of generality, we may assume that Player σ plays in each move a finite word in σ^+. Hence the game is equivalent to the game where the players play a strictly increasing sequence $a_0 < a_1 < a_2 < \ldots$ and Player 0 wins the resulting infinite play if, and only if, the set $[0, a_0) \cup [a_1, a_2) \cup [a_3, a_4) \cup \ldots$ belongs to U.

Assume that Player 0 has a winning strategy f which maps any increasing sequence $a_0 < a_1 < \cdots < a_{2n-1}$ of even length to $a_{2n} = f(a_0a_1 \ldots a_{2n-1}) > a_{2n-1}$. We consider two intertwined counter-strategies of Player 1, essentially forcing Player 0 to simultaneously

perform two plays against herself. In reply to the first move a_0, Player 1 selects an arbitrary $a_1 > a_0$ and then sets up the two plays as follows: In the first one she replies to a_0 by a_1 and waits for the answer $a_2 = f(a_0a_1)$ by Player 0. She then uses a_2 as her own reply to a_0 in the second play and gets the answer $a_3 = f(a_0a_2)$ by Player 0, which she now uses as her next move in the first play. There Player 0 responds by $a_4 = f(a_0a_1a_2a_3)$ which is again used by Player 1 as her answer to $a_0a_2a_3$ in the second play. And so on.

In this way, the two infinite plays result in sequences $a_0 < a_1 < a_2 < \dots$ and $a_0 < a_2 < a_3 < \dots$. Since Player 0 plays with her winning strategy in both plays. it follows that $X = [0, a_0) \bigcup_{n\in\omega} [a_{2n+1}, a_{2n+2}) \in U$, but also $X' = [0, a_0] \cup \bigcup_{n>0} [a_{2n}, a_{2n+1}) \in U$. By closure under intersection, it follows that $X \cap X' = [0, a_0) \in U$. But U is a free ultrafilter, so it cannot contain a finite set.

It follows by the same argument that Player 1 cannot have a winning strategy. ∎

3 Determinacy by simple strategies

In general, strategies can be very complicated as they may depend on the entire history of a play. However, there are interesting classes of games that are determined via relatively simple winning strategies. We will discuss several kinds of restricted strategies:

(1) **Decomposition invariant strategies** are strategies that depend only on the finite path that has been produced so far, and not on its decomposition into the moves of the players. Thus, a decomposition invariant strategy is a function assigning to each finite path a finite prolongation. We will show that, whenever a player has a winning strategy in a Banach-Mazur game, then she also has one that is decomposition invariant.

(2) **Positional strategies** (also called memoryless strategies) depend only on the current position, and not on the history of the play. On a game graph $G = (V, E)$ a positional strategy is a function $f : V \to V^*$ assigning to every node v a finite path $f(v) \in \text{FinPaths}(G, v)$. It is easy to find determined games that require non-positional winning stategies, but we will prove that all Banach-Mazur games with ω-regular objectives are determined via positional winning strategies.

(3) More generally, **strategies with memory** $\mathfrak{M}$ depend on the history of the play in a restricted way, via a memory structure $\mathfrak{M}$, consisting of a set of memory locations and an update function that changes the memory location as the play proceeds. Strategies with a finite memory structure can be implemented by a finite automaton. We will show that, for Banach-Mazur games, finite memory structures are irrelevant in the sense that winning strategies with finite memory can always be transformed into positional winning strategies. This is in sharp contrast to the usual graph games where already quite simple ω-regular winning conditions (such as, in particular, Muller conditions) lead to games that are determined by finite-memory strategies , but not by positional ones.

3.1 Decomposition invariant strategies

DEFINITION 8. A decomposition invariant strategy in a Banach-Mazur game on a graph (G, v) is a function $f : \text{FinPaths}(G, v) \to \text{FinPaths}(G, v)$ such that $x \leq f(x)$ for all x.

THEOREM 9. *Every Banach-Mazur game that is determined is also determined via a decomposition invariant strategy.*

PROOF. Suppose that Player 1 has a winning strategy for the game $\text{BM}(G, v, \text{Win})$. Then $\text{Win} = \bigcup_{n<\omega} X_n$ with X_n nowhere dense. This means that the complement of each X_n contains a dense open set Y_n. Hence there exists a function g_n assigning to each finite path y a prolongation $g_n(y)$ such that $\mathcal{O}(g_n(y)) \subseteq Y_n$. We define a decomposition-invariant strategy g as follows. Given a finite path $x \in \text{FinPaths}(G, v)$, there are only finitely many $n < \omega$ such that $g_n(y) \leq x$ for some $y \in \text{FinPaths}(G, v)$. Take the minimal n such that this is not the case and set $g(x) = g_n(x)$.

It remains to show that g is a winning strategy for Player 1. Let π be any infinite play that is consistent with g. For every $n < \omega$ there exists a prefix y such that $g_n(y) < \pi$. Hence $\pi \in Y_n$ for all n, which means that π is won by Player 1.

The argument for Player 0 is analogous ∎

3.2 Positional determinacy

To start, we present a simple example of a Banach-Mazur game that is determined, but does not admit a positional winning strategy.

Example 10 Let G_2 be the completely connected directed graph with nodes 0 and 1, and let the winning condition for Player 0 be the set of infinite sequences with infinitely many initial segments that contain more ones than zeros. Clearly, Player 0 has a winning strategy for this game, but not a positional one.

Note that this winning condition is on the $\mathbf{\Pi}_2$-level of the Borel hierarchy. As we show next, this is the lowest level with such an example.

PROPOSITION 11. *If Player 0 has a winning strategy for a Banach-Mazur game with a winning condition* $\text{Win} \in \mathbf{\Sigma}^0_2$*, then she also has a positional winning strategy.*

PROOF. Suppose that Player 0 has a winning strategy f for the Banach-Mazur game $\text{BM}(G, v, \text{Win})$ such that Win is a countable union of closed sets. We have $\text{Win} = \bigcup_{n<\omega}[T_n]$ where each $T_n \subseteq \text{FinPaths}(G, v)$ is closed under prefixes. Further, we can assume that the winning strategy f is decomposition invariant. We claim that, in fact, Player 0 can win with one move, i.e. there is a finite path x such that $\mathcal{O}(x) \subseteq \text{Win}$.

We construct this move by induction. Let x_1 be the initial path chosen by Player 0 according to f. Let $i \geq 1$ and suppose that we have already constructed a finite path $x_i \notin \bigcup_{n<i} T_n$. If $x_i y \in T_i$ for all finite y, then all infinite plays extending x_i remain in Win, hence Player 0 wins with the initial move $w = x_i$. Otherwise choose some y_i such that $x_i y_i \notin T_i$, and suppose that Player 1 prolongs the play from x_i to $x_i y_i$. Let $x_{i+1} := f(x_i y_i)$ the result of the next move of Player 0, according to her winning strategy f.

If this process did not terminate, then it would produce an infinite play that is consistent with f and won by Player 1. Since f is a winning strategy, this is impossible. Hence there

exists some $m < \omega$ such that $x_m y \in T_m$ for all y. Thus, if Player 0 moves to x_m in her opening move, then she wins, no matter how the play proceeds afterwards. In particular, Player 0 wins with a positional strategy. ∎

While many important winning conditions are outside $\mathbf{\Sigma}^0_2$, they may well be Boolean combinations of $\mathbf{\Sigma}^0_2$-sets. For instance, this is the case for parity conditions, Muller conditions, and more generally, all ω-regular winning conditions. In the classical framework of infinite games on graphs (where moves are along single edges rather than paths) it is well-known that parity games admit positional winning strategies [6, 17, 9], whereas there are simple games with Muller conditions that require strategies with some memory. We will see that for Banach-Mazur games, the class of winning conditions guaranteeing positional winning strategies is much larger than for classical graph games.

3.3 Banach-Mazur games with Muller winning conditions

A Muller condition is any property of infinite sequences $x \subseteq C^\omega$ that depends only on which symbols $c \in C$ occur infinitely often in x. Muller conditions are of crucial importance in automata theory and in the theory of infinite games. It is one of the standard acceptance conditions for automata on infinite words or infinite trees

DEFINITION 12. A Muller condition on a set C is written in the form $(\mathcal{F}_0, \mathcal{F}_1)$ where $\mathcal{F}_0 \subseteq \mathcal{P}(C)$ and $\mathcal{F}_1 = \mathcal{P}(C) \setminus \mathcal{F}_0$. Given a game graph $G = (V, E)$ whose nodes are labelled by a function $\Omega : V \to C$, a play $\pi \in \mathrm{Paths}(G, v)$ is won by Player σ if, and only if, the set of colours occurring infinitely often on π belongs to $\mathcal{F}_\sigma$.

Usually it is assumed that the set C of colours is finite. In that case there is a precise characterisation, due to Zielonka [24] of the Muller winning conditions that guarantee positional determinacy for the classical form of graph games. It sates that all games with winning condition $(\mathcal{F}_0, \mathcal{F}_1)$ are positionally determined if, and only if, neither $\mathcal{F}_0$ nor $\mathcal{F}_1$ contains a strong split, which means that there do not exist two sets $X, Y \in \mathcal{F}_\sigma$ such that $X \cap Y \neq \emptyset$ and $X \cup Y \in F_{1-\sigma}$.

However, as we show now, *all* Muller conditions (on a finite set of colours) guarantee positional determinacy for Banach-Mazur games.

PROPOSITION 13. *All Banach-Mazur games* $\mathrm{BM}(G, v_0, (\mathcal{F}_0, \mathcal{F}_1))$ *with a Muller winning condition on a finite set of colours are positionally determined.*

PROOF. We write $w \geq v$ to denote that position w is reachable from position v. For every position $v \in V$, let $C(v)$ be the set of colours reachable from v, that is, $C(v) := \{\Omega(w) : w \geq v\}$. Obviously, $C(w) \subseteq C(v)$ whenever $w \geq v$. In case $C(w) = C(v)$ for all $w \geq v$, we call v a *stable* position. Note that from every $u \in V$ some stable position is reachable. Further, if v is stable, then every reachable position $w \geq v$ is stable as well.

We claim that Player 0 has a winning strategy in $\mathrm{BM}(G, v, (\mathcal{F}_0, \mathcal{F}_1))$ if, and only if, there is a stable position w that is reachable from the initial position v, so that $C(w) \in \mathcal{F}_0$.

To see this, let us assume that there is such a stable position v with $C(w) \in \mathcal{F}_0$ for a stable position $w \geq v$. Then, for every $u \geq w$, we choose a path p from u so that, when moving along p, each colour of $C(u) = C(w)$ is visited at least once, and set $f(u) := p$. In

case v is not reachable from w, let $f(v)$ be some path that leads from v to w. Now f is a positional winning strategy for Player 0 because, after the first move, no colours other then those in $C(w)$ are seen. Moreover, every colour in $C(w)$ is visited at each move of Player 0, hence, infinitely often.

Conversely, if for every stable position w reachable from v we have $C(w) \in \mathcal{F}_1$, we can construct a winning strategy for Player 1 in a similar way. ∎

Note that in a finite arena all positions of a strongly connected component that is terminal, i.e., with no outgoing edges, are stable. Thus, the above characterisation translates as follows: Player 0 wins the game if, and only if, there is a terminal component whose set of colours belongs to $\mathcal{F}_0$. Obviously this can be established in linear time w.r.t. the size of the arena and a suitable description of the Muller condition.

COROLLARY 14. *On a finite arena G, Banach-Mazur games with a Muller winning condition $(\mathcal{F}_0, \mathcal{F}_1)$ can be solved in time $O(|G| \cdot |\mathcal{F}_\sigma|)$.*

We remark that solving single-step graph games with Muller winning conditions is PSPACE-complete. This is not too difficult to derive from the analysis presented in [5]. A detailed complexity analysis, for a number of different presentations of Muller conditions, can be found in [11].

3.4 Elimination of finite memory

We introduce a general notion of a memory structure and of a strategy with memory. The memory can be finite, as in the finite memory strategies studied for instance in [5], or infinite is in the strategies used in [7].

DEFINITION 15. A *memory structure* for a game graph $G = (V, E)$ is given by a triple $\mathfrak{M} = (M, m_0, \text{update})$, with a set of *memory states* M, an initial state m_0 and a *memory update function* $\text{update} : M \times V \to M$. The *size* of the memory is the cardinality of the set M. A *strategy with memory* $\mathfrak{M}$ for a Banach-Mazur game on G is given by a next-move function $f : V \times M \to V^*$ such that $f(v, m) \in \text{FinPaths}(G, v)$ for all $v \in V, m \in M$.

Notice that the local memory update function extends to a function $\text{memory} : M \times V^* \to V$, where $\text{memory}(m, x)$ is the memory state that is reached by a sequence of updates along a path x, starting with memory state m. This function is defined inductively by

$$\text{memory}(m, \varepsilon) = m, \qquad \text{memory}(m, xv) := \text{update}(\text{memory}(m, x), v).$$

In particular, if a play has gone from initial position v_0 through a finite path $x \in \text{FinPaths}(G, v_0)$ ending at node v, then the memory state is $m = \text{memory}(m_0, x)$, and the strategy defined by $\mathfrak{M}$ and F will prolong x by the path $F(v, m)$.

We will say that a game is determined via memory $\mathfrak{M}$ if one of the players has a winning strategy with memory $\mathfrak{M}$.

THEOREM 16. *A Banach-Mazur game that is determined via a finite-memory winning strategy is in fact positionally determined.*

PROOF. Let us assume that Player 0 wins a Banach-Mazur game on (G, v_0) with a strategy $f : V \times M \to V^*$ based on a finite memory structure $\mathfrak{M} = (M, m_0, \text{update})$. For any node $v \in V$, we denote by $M(v)$ the set of memory locations $\text{memory}(m_0, x)$ such that x is path from v_0 to v that may arise as an initial segment of some play consistent with f.

$$M(v) := \{\text{memory}(m_0, x) : x \text{ prolongs } f(v_0, m_0) \text{ and leads to } v\}.$$

Let $\{m_1, m_2, \ldots, m_n\}$ be an enumeration of $M(v)$, in which the initial memory m_0 is taken first, in case it belongs to $M(v)$. We construct paths $y_1 < y_2 < \cdots < y_n \in \text{FinPaths}(G, v)$. First, set $y_1 := f(v, m_1)$. Then, for $1 \leq i < n$, let y_{i+1} be the concatenation of y_i with the path $f(v_i, \text{memory}(m_i, y_i))$ where v_i is the end node of y_i. Finally, set $f'(v) := y_n$.

Clearly f' is a positional strategy. We claim that it is a winning strategy for Player 0. Consider any play π that is consistent with f'. Cleary $f(v_0, m_0)$ is an initial segment of π. Further, suppose that after some finite number of moves, an initial segment x ending at position v has been produced. Player 0 now prolongs x by the path $f'(v)$.

We claim that the path $f'(v)$ can be written in the form $z_1 z_2 z_3$ such that there exist $v' \in V$ and $m' \in M$ with

- z_1 ends at node v'
- $\text{memory}(m_0, xz_1) = m'$,
- $z_2 = f(v', m')$

Indeed, if $M(v) = \{m_1, \ldots, m_n\}$, we have $\text{memory}(m_0, x) = m_i$ for some $i \leq n$. Let $z_1 := y_i$. Then $v' = v_i$, $m' = \text{memory}(m_0, xy_i) = \text{memory}(m_i, y_i)$, and $f'(v) = z_1 f(v', m') z_3$ for appropriate z_3.

In other words, every move of Player 0 has some "good part" z_2 that would also have been produced by the strategy f if Player 0 had to choose at the position v' with current memeory state m'. But this means that the play cannot be distinguished from a play where Player 0 always moved according to the strategy f while all the "bad parts" were produced by Player 1. Hence the play is also consistent with f and therefore won by Player 0.

The same construction works for Player 1, if we define $M(v) := \{\text{memory}(m_0, x) : x \text{ is a path from } v_0 \text{ to } v\}$. ∎

This result has very interesting consequences for Banach-Mazur games with ω-regular winning conditions. Let $G = (V, F)$ be a game graph with a colouring $\Omega : V \to C$ of the nodes by a finite number of colours and consider winning conditions given by an ω-regular set $W \subseteq C^\omega$. Such conditions can by defined by a formula in some appropriate logic over infinite paths. In the most general case, we have S1S-formulae (i.e. MSO-formulae on infinite paths with vocabulary $\{<\} \cup \{P_c : c \in C\}$). It is well known that every S1S-definable class of infinite words can be recognised by a deterministic Muller or parity automaton (see e.g. [8]). Hence, by a standard construction, any game on a graph G with an ω-regular winning condition can be reformulated as a game on a graph $G \times \mathfrak{M}$, for a finite memory structure $\mathfrak{M}$, with a Muller (or parity) winning condition. This means that we get for G a winning strategy with memory $\mathfrak{M}$ for one of the players. Theorem 16 tells us that in the case of Banach-Mzur games, we can get rid of this finite memory.

THEOREM 17. *All Banach-Mazur games with ω-regular winning conditions are positionally determined.*

4 Definability

We now discuss the question in what logics (MSO, μ-calculus, FO, CTL*, ...) winning positions of Banach-Mazur games with ω-regular winning conditions can be defined. Given any formula φ from a logic on infinite paths (like S1S or LTL), we define the game formula $\eth\varphi$, to be evaluated over game graphs, with the meaning that

$$G \models \eth\varphi(v) \iff \text{Player 0 wins the Banach-Mazur game BM}(G, v, \varphi).$$

Note that the operation $\varphi \mapsto \eth\varphi$ maps a formula over infinite paths to a formula on graphs. Given a logic L over infinite paths, and a prefix let Game-$L := \{\eth\varphi : \varphi \in L\}$. As usual we write $L \leq L'$ to denote that every formula in the logic L is equivalent to some formula from the logic L'.

Our main definability result can be stated as follows.

THEOREM 18.
(1) Game-S1S $\leq L_\mu$
(2) Game-LTL $\equiv$ Game-FO $\leq$ CTL*.

Obviously, the properties expressed by formulae $\eth\varphi$ are invariant under bisimulation. This has two relevant consequences:

(a) We can restrict attention to trees (obtained for instance by unravelling the given game graph from the start node).
(b) It suffices to show that, on trees, Game-S1S $\leq$ MSO, and Game-FO $\leq$ MPL where MPL is *monadic path logic*, i.e., monadic second-order logic where second-order quantification is restricted to infinite paths.

Indeed, it has been proved by Janin and Walukiewicz [12] that every bisimulation-invariant class of trees that is MSO-definable is also definable in the modal μ-calculus. Similarly, it is known from results by Hafer and Thomas [10] and by Moller and Rabinovitch [16], that every bisimulation invariant property of trees expressible in MPL is also expressible in CTL*.

PROPOSITION 19. *On trees,* Game-S1S $\leq$ MSO.

PROOF. Let $x \leq y$ denote that y is reachable from x. A (decomposition-invariant) strategy for Player 0 in a game BM(T, r, Win) on a tree $T = (V, E)$ with root r is a partial function $f : V \to V$, such that $w < f(w)$ for every w; it is winning if every infinite path through T containing $r, f(r), y_1, f(y_1), y_2, f(y_2) \ldots$, where $f(y_i) < y_{i+1}$ for all i, is contained in Win. An equivalent description can be given in terms of the set $X = f(V)$. A set $X \subseteq V$ defines a winning strategy for Player 0 in the game BM(T, r, Win) if

(1) $(\forall x \in X)\forall y(x < y \to (\exists z \in X)(y < z))$
(2) every path hitting X infinitely often is in Win (i.e. it is winning for Player 0)
(3) X is non-empty.

Clearly these conditions are expressible in MSO. ∎

To deal with winning conditions defined in first-order logic (or equivalently, LTL), we use a normal form for first-order logic on infinite paths (with $<$) that has been established by Thomas [22]. A first-order formula $\varphi(\overline{x})$ is *bounded* if it only contains bounded quantifiers of form $(\exists y \leq x_i)$ or $(\forall y \leq x_i)$.

PROPOSITION 20. *On infinite paths, every first-order formula is equivalent to a formula of the form*

$$\bigvee_i \Big(\exists x(\forall y \geq x)\varphi_i \wedge \forall y(\exists z \geq y)\vartheta_i\Big)$$

where φ_i and ϑ_i are bounded.

THEOREM 21. *On trees,* Game-FO $\leq$ FO.

PROOF. Let $\psi = \bigvee_i \Big(\exists x(\forall y \geq x)\varphi_i \wedge \forall y(\exists z \geq y)\vartheta_i\Big)$ be a first-order formula on infinite paths describing a winning condition. We claim that, on trees, $\Game\psi$ is equivalent to the first-order formula

$$\psi^* := (\exists p_1)(\forall p_2 \geq p_1)(\exists p_3 \geq p_2) \bigvee_{i\in I} \psi_i^{(b)} \quad \text{where}$$

$$\psi_i^{(b)} := (\exists x \leq p_1)(\forall y\,.\, x \leq y \leq p_2)\varphi_i \wedge (\forall y \leq p_2)(\exists z\,.\, y \leq z \leq p_3)\vartheta_i.$$

Let $T = (V, E)$ and suppose first that Player 1 has a winning strategy for $\mathrm{BM}(T, r, \psi)$. We prove that $T \models \neg\psi^*$. To see this we have to define an appropriate Skolem function $g : p_1 \mapsto p_2$ such that, for all $p_3 \geq p_2$ and all $i \in I$,

$$T \models \neg\psi_i^{(b)}(p_1, p_2, p_3).$$

Fix any p_1 that we can consider as the first move of Player 0 in the game $\mathrm{BM}(T, r, \psi)$ and any play P (i.e., any infinite path through T) that prolongs this move and that is consistent with the winning strategy of Player 1. Since Player 1 wins, we have that $P \models \neg\psi$. Hence, there exists some $J \subseteq I$ such that

$$P \models \bigwedge_{i\in J} \forall x(\exists y \geq x)\neg\varphi_i \wedge \bigwedge_{i\in I-J} \exists y(\forall z \geq y)\neg\vartheta_i.$$

To put it differently, there exist

- for every $i \in J$ and every $a \in P$, a witness $h_i(a) \in P$ such that $P \models \neg\varphi_i(a, h_i(a))$, and
- for every $i \in I - J$, an element b_i such that $P \models (\forall z \geq b_i)\neg\vartheta_i(b_i, z)$.

Now set

$$p_2 := \max(\{h_i(a) : a \leq p_1, i \in J\} \cup \{b_i : i \in I - J\}).$$

For any p_3 we now obviously have that $T \models \neg\psi_i^{(b)}(p_1, p_2, p_3)$.

For the converse, let $f : V \to V$ be a winning strategy for Player 0 in $\mathrm{BM}(T, r, \psi)$. We claim that $T \models \psi^*$. Toward a contradiction, suppose that $T \models \neg\psi^*$. Hence there exists a Skolem function $g : V \to V$ assigning to each p_1 an appropriate $p_2 \geq p_1$ such that $T \models$

$\neg\psi_i^{(b)}(p_1, p_2, p_3)$ for all $p_3 \geq p_2$ and all $i \in I$. We can view g as a strategy for Player 1 in the game $\mathrm{BM}(T, r, \psi)$. If Player 0 plays according to f and Player 1 according to g, then the resulting infinite play $f \hat{} g = q_1 q_2 q_3 \ldots$ satisfies ψ (because f is a winning strategy). Hence there exists some $i \in I$ such that

$$f \hat{} g \models \exists x (\forall y \geq x) \varphi_i \wedge \forall y (\exists z \geq y) \vartheta_i.$$

Let a be a witness for x so that $f \hat{} g \models (\forall y \geq a)\varphi_i(a, y)$. Choose the minimal odd k, such that $a \leq q_k$, and set $p_1 := q_k$. Then $q_{k+1} = g(q_k) = g(p_1) = p_2$. Since $f \hat{} g \models \forall y (\exists z \geq y)\vartheta_i(y, z)$, we have, in particular, for every $b \leq p_2$ a witness $h(b) \geq b$ on $f \hat{} g$ such that $f \hat{} g \models \vartheta_i(b, h(b))$. Choose $p_3 = \max\{h(b) : b \leq p_2\}$. It follows that $f \hat{} g \models \psi_i^{(b)}(p_1, p_2, p_3)$. Since $\psi_i^{(b)}$ is bounded, its evaluation on T is equivalent to its evaluation on $f \hat{} g$. Hence we have shown that there exists p_1 such that for $p_2 = g(p_1)$, given by the Skolem function g, we can find a p_3 with $T \models \psi_i^{(b)}(p_1, p_2, p_3)$. But this contradicts the assumption that g is an appropriate Skolem function for $\neg\psi^*$.

We have shown that whenever Player 0 has a winning strategy for $\mathrm{BM}(T, r, \psi)$ then $T \models \psi^*$ and whenever Player 1 has a winning strategy, then $T \models \neg\psi^*$. By contraposition and determinacy, the reverse implications also hold. ∎

Theorem 18 is implied by Proposition 19 and Theorem 21.

We have seen that for every fixed winning condition expessible in S1S, the winner of the associated Banach-Mazur games is uniformly definable in the μ-calculus. Notice however that this requires that we consider games with a fixed number of local parameters (colours) by which this winning condition is defined. But in the theory of infinite game, a number of algorithmic question concerns classes of games where the number of coulours to define the winning condition is not fixed, but may depend on the game graph.

The most important example is the parity condition: Given a function $\Omega : V \to \omega$, Player 0 wins those infinite plays in which the least value appearing infintely often is even. For the usual format of graph games, one of the most prominent open problems is the question whether the winning regions of parity games are computable in polynomial time. This problem is equivalent to the question whether the modal μ-calculus admits a polynomial-time model checking algorithm. Even if the range of Ω, i.e. the number of colours, is assumed to be finite, it is not bounded.

For parity games with a fixed number d of colours, which can be viewed as structures $(V, E, P_0, \ldots, P_{d-1})$, it is well-known that the winner is computable in polynomial-time and definable by a μ-calculus formula (with d alternations between least and greatest fixed points). The interesting problem concerns the case of an unbounded number of priorities, and the current algorithms for solving parity games only have upper time-complexity bounds that are exponential in the number of colours.

What about the definability of winning positions in parity games with an unbounded number of colours? First, of all we have to represent the structures in a different way, to avoid an infinite vocabulary. For instance we can describe game graphs as structures

$$(V, E, \prec, \mathrm{Odd})$$

where $u \prec v$ means that u has a smaller colour than v, and Odd is the set of nodes with an odd colour. We denote this class of structures by $\mathcal{PG}$.

The descriptive complexity of parity games, i.e. the question in which logics, winning regions of parity games are definable, has been considered in [4]. The descriptive complexity of a problem provides an insight into the structure of the problem, and the sources of algorithmic difficulty, as the logical resources needed to specify the problem are closely tied to its structure. In the case of parity games, the questions that naturally arise are whether the problem is definable in the least fixed-point logic (LFP) and in monadic second-order logic (MSO), as these are logics with which it is closely associated.

It has been proved in [4] that on arbitrary (finite or infinite) game graphs, parity games are not definable in the least fixed point logic LFP. On finite games graphs, it turned out that the winning regions are LFP-definable if, and only if, they are computable in polynomial-time (despite the fact that, on unordered finite structures, LFP is weaker than PTIME).

Again, it turns out that the analogous question for Banach-Mazur games is simpler.

THEOREM 22. *Winning regions of Banach-Mazur games with the parity winning condition are uniformly definable in least fixed-point logic LFP.*

PROOF. In the proof of Proposition 13 we have shown that Player 0 wins a Banach-Mazur game on (G, v) with a Muller condition $(\mathcal{F}_0, \mathcal{F}_1)$ if, and only if, there is a stable position w, reachable from v, such that $C(w) \in \mathcal{F}_0$. For parity games $C(w) \in \mathcal{F}_0$ means that the least colour in $C(w)$ is even. Clearly, this condition is uniformly definable in least-fixed point logic on $\mathcal{PG}$. ∎

This result in fact applies to weaker logics than LFP. It suffices that reachability statements "there is a path from x to y" are expressible. Also, the result may apply to stronger classes of Muller conditions, but it depends on how these are described. In what ever way, such a condition $(\mathcal{F}_0, \mathcal{F}_1)$ is presented on the given game graphs, the necessary condition to express is that $C(w) \in \mathcal{F}_0$.

5 Path games with bounded alternations

Banach-Mazur games have an infinite sequence of alternating moves of the two players. There is an interesting variation of such games where one of the player only makes finitely many moves and eventually one player plays alone. To describe the alternation patterns of such games, we now call the players Ego and Alter, and denote a move where Ego selects a finite path by E, and an ω-sequence of such moves by E^ω; for Alter, we use corresponding notation A and A^ω.

Hence, for any graph G initial position v, and winning condition Win we have the following games.

- $(EA)^\omega(G, v, \mathrm{Win})$ is the ususal Banach-Mazur games with infinite alternation between the two players. By exchanging the roles of the players, we get the game $(AE)^\omega(G, v, \mathrm{Win})$.
- $(EA)^k E^\omega(G, v, \mathrm{Win})$ and $A(EA)^k E^\omega(G, v, \mathrm{Win})$, for arbitrary $k \in \omega$, are the games ending with an infinite path extension by Ego.
- $(AE)^k A^\omega(G, v, W)$ and $E(AE)^k A^\omega(G, v, W)$ are the games where Alter chooses the final infinite lonesome ride.

All these games together form the collection $\mathrm{Path}(G, v, W)$ of *path games*. (Obviously two consecutive finite path moves by the same players correspond to a single move, so there is no need for quantifier strings containing EE or AA.)

Pistore and Vardi [20] used path games of this form for task planning in nondeterministic domains.

5.1 Comparison of path games

For two games $\mathcal{G}$ and $\mathcal{H}$ we write $\mathcal{G} \preceq \mathcal{H}$ if, from the point of view of Ego, $\mathcal{H}$ is better than $\mathcal{G}$. More precisely, $\mathcal{G} \preceq \mathcal{H}$ if, whenever Ego has a winning strategy for $\mathcal{G}$ then he also has one for $\mathcal{H}$, and if Alter has a winning strategy for $\mathcal{H}$ then he has one also for $\mathcal{G}$. Finally, $\mathcal{G} \equiv \mathcal{H}$ if $\mathcal{G} \preceq \mathcal{H}$ and $\mathcal{H} \preceq \mathcal{G}$.

It turns out that this infinite collection of games defined by the game quatifier prefixes over E and A collapses in a uniform way to a finite lattice of just eight different games. This has been observed independently in [3] and [20].

THEOREM 23. *For every arena G and every winning condition* Win*, we have*

$$\begin{array}{ccccc}
E^\omega(G,v,\mathrm{Win}) & \succeq & EAE^\omega(G,v,\mathrm{Win}) & \succeq & AE^\omega(G,v,\mathrm{Win}) \\
 & & \curlyvee| & & \curlyvee| \\
 & & (EA)^\omega(G,v,\mathrm{Win}) & \succeq & (AE)^\omega(G,v,\mathrm{Win}) \\
 & & \curlyvee| & & \curlyvee| \\
 & & EA^\omega(G,v,\mathrm{Win}) & \succeq & AEA^\omega(G,v,\mathrm{Win}) \quad \succeq \quad A^\omega(G,v,\mathrm{Win})
\end{array}$$

Further, every path game $\mathcal{H} \in \mathrm{Path}(G, v, \mathrm{Win})$ is equivalent to one of these eight games.

PROOF. The comparison relations in the diagram follow by trivial arguments. We just illustrate them for one case. To show that $\mathcal{G} \succeq \mathcal{H}$ for $\mathcal{G} = EAE^\omega(G, v, \mathrm{Win})$ and $\mathcal{H} = (EA)^\omega(G, v, \mathrm{Win})$, consider first a winning strategy f of Ego in $\mathcal{H}$. Ego can use this strategy also for $\mathcal{G}$: he just plays as if he would play $\mathcal{G}$, making an arbitrary move whenever it would be Alter's turn in $\mathcal{H}$. Any play in $\mathcal{G}$ that is consistent with this strategy, is also a play in $\mathcal{H}$ that is consistent with f, and is therefore won by Ego. Second, consider a winning strategy g of Alter in $\mathcal{G}$. In $\mathcal{H} = (EA)^\omega(G, v, \mathrm{Win})$, Alter answers the first move of Ego as prescribed by g, and moves arbitrarily in all further moves. Again, every play that can be produced against this strategy is also a play of $\mathcal{G}$ that is consistent with g, and is therefore won by Alter. In all other cases the arguments are analogous.

To see that any other path game over (G, v, Win) is equivalent to one of those displayed, it suffices to show that

(1) $(EA)^k E^\omega(G, v, \mathrm{Win}) \equiv EAE^\omega(G, v, \mathrm{Win})$, for all $k \geq 1$, and
(2) $A(EA)^k E^\omega(G, v, \mathrm{Win}) \equiv AE^\omega(G, v, \mathrm{Win})$, for all $k \geq 0$.

By duality, we can then infer that $(AE)^k A^\omega(G, v, \mathrm{Win}) \equiv AEA^\omega(G, v, \mathrm{Win})$ for $k \geq 1$ and $E(AE)^k A^\omega(G, v, \mathrm{Win}) \equiv EA^\omega(G, v, \mathrm{Win})$ for all $k \geq 0$.

The equivalences (1) and (2) follow with similar reasoning. Ego can modify a strategy f for $EAE^\omega(G, v, \mathrm{Win})$ to a strategy for $(EA)^k E^\omega(G, v, \mathrm{Win})$. He chooses the first move according to f and makes arbitrary moves the next $k-1$ times; he then considers the entire $A(EA)^{k-1}$-sequence of moves, which were played after his first move, as one single move of A in $EAE^\omega(G, v, \mathrm{Win})$ and completes the play again according to f. The resulting play of $(EA)^k E^\omega(G, v, \mathrm{Win})$ is also consistent with f in $EAE^\omega(G, v, \mathrm{Win})$. Conversely a strategy of Ego for $(EA)^k E^\omega$ also works if his opponent lets Ego move for him in all moves after the first one, i.e., in the game $EAE^\omega(G, v, \mathrm{Win})$. All other equalities are treated in a similar way. ∎

The question arises whether the eight games displayed in the diagram are really different or whether they can be collapsed further. The answer depends on the game graph and the winning condition but for each comparison $\succeq$ in the diagram we find simple cases where it is strict. Indeed, standard winning conditions $\mathrm{Win} \subseteq \{0,1\}^\omega$ on the completely connected graph G_2 with nodes 0 and 1 show that the eight games in the diagram are distinct.

If the winning condition requires a particular initial segment then Ego wins the path games where he moves first and loses those where Alter moves first. Thus, starting conditions separate the left half of the diagram from the right one. Games with reachability conditions and safety conditions separate games in which only one player moves, i.e. with prefix E^ω or A^ω respectively, from the other ones. A game with a Büchi condition is won by Ego if he has infinite control and lost if he only has a finite number of finite moves (prefix ending with A^ω). Similarly, Co-Büchi conditions separate the games which are controlled by Ego from some time onwards (with prefix ending in E^ω) from the others.

5.2 Positional determinacy of ω-regular path games

We have seen that Banach-Mazur games are positionally determined for any ω-regular winning condition. Does this also hold for path games with bounded alternation between the players?

To establish positional determinacy for ω-regular Banach-Mazur games, we first noticed that the reduction of *S1S* to deterministic parity automata gives us determinacy by finite-memory strategies. In a second step, we proved that for Banach-Mazur games we can eliminate the finite memory, and reduce finite memory strategies to positional ones. The first of these two steps does not depend on the alternation pattern in the game, and therefore also holds for path games with bounded alternation.

PROPOSITION 24. *For any winning condition $\psi \in$ S1S and any game prefix γ, the path games $\gamma(G, \psi)$ admit finite-memory winning strategies.*

However, the reduction from finite memory strategies to positional ones in the proof of Theorem 16 does rely on infinite alternation between the players. For games where the players alternate only finitely often the situation changes. Intuitively, a winning strategy of the solitaire player eventually forms an infinite path which may not be broken apart into finite pieces to serve as a positional strategy.

PROPOSITION 25. *For any prefix γ with finitely many alternations between the players, there are arenas G and winning conditions $\psi \in$ S1S so that no positional strategy is winning in the game $\gamma(G, v, \psi)$.*

PROOF. Consider, for instance, the arena G_2 from Example 10 and a winning condition $\psi \in$ S1S that requires the number of zeroes occurring in a play to be odd. When starting from position 1 Ego obviously has winning strategies for each of the games $E^\omega(G, \psi)$, $AE^\omega(G, \psi)$, and $EAE^\omega(G, \psi)$, but no positional ones. ∎

Nevertheless, these games are positionally determined for one of the players. Indeed, if a player wins a game $\gamma(G, v, \psi)$ that is finally controlled by his opponent, he always has a positional winning strategy. This is trivial when $\gamma \in \{E^\omega, A^\omega, AE^\omega, EA^\omega\}$; for the remaining cases EAE^ω and AEA^ω a positional strategy can be constructed as in the proof of Theorem 16.

Finally we consider winning conditions that do not depend on initial segments. We say that ψ is prefix independent, if, for any ω-word π and any finite words x and y, we have $x\pi \models \psi$ if, and only if, $y\pi \models \psi$.

THEOREM 26. *For any prefix-independent winning condition $\psi \in$ S1S and every γ, the games $\gamma(G, v, \psi)$ admit positional winning strategies.*

References

[1] C. BAIER, N. BERTRAND, P. BOUYER, T. BRIHAYE, AND M. GRÖSSER, *Probabilistic and topological semantics for timed automata*, in Proceedings of the 27th Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS'07, 2007, pp. 179–191.

[2] C. BAIER, N. BERTRAND, P. BOUYER, T. BRIHAYE, AND M. GRÖSSER, *Almost-sure model checking of infinite paths in one-clock timed automata*, in Proceedings of 23rd Annual IEEE Symposium on Logic in Computer Science LICS'08, 2008, pp. 217–226.

[3] D. BERWANGER, E. GRÄDEL, AND S. KREUTZER, *Once upon a time in the west. Determinacy, complexity and definability of path games*, in Proceedings of the 10th International Conference on Logic for Programming and Automated Reasoning, LPAR 2003, Almaty, Lecture Notes in Computer Science Nr. 2850, Springer-Verlag, 2003, pp. 226–240.

[4] A. DAWAR AND E. GRÄDEL, *The descriptive complexity of parity games*, in Proceedings of 22th Annual Conference of the European Association for Computer Science Logic CSL 2008, 2008, pp. 354–368.

[5] S. DZIEMBOWSKI, M. JURDZIŃSKI, AND I. WALUKIEWICZ, *How much memory is needed to win infinite games?*, in Proceedings of 12th Annual IEEE Symposium on Logic in Computer Science (LICS 97), 1997, pp. 99–110.

[6] A. EMERSON AND C. JUTLA, *Tree automata, mu-calculus and determinacy*, in Proc. 32nd IEEE Symp. on Foundations of Computer Science, 1991, pp. 368–377.

[7] E. GRÄDEL AND Ł. KAISER, *What kind of memory is needed to win infinitary muller games?*, in Interactive Logic, J. van Benthem, B. Lwe, and D. Gabbay, eds., vol. 1 of Texts in Logic and Games, Amsterdam University Press, 2007, pp. 89–116.

[8] E. GRÄDEL, W. THOMAS, AND T. WILKE, eds., *Automata, Logics, and Infinite Games*, Lecture Notes in Computer Science Nr. 2500, Springer, 2002.

[9] E. GRÄDEL AND I. WALUKIEWICZ, *Positional determinacy of games with infinitely many priorities*, Logical Methods in Computer Science, (2006).

[10] T. HAFER AND W. THOMAS, *Computation tree logic CTL* and path quantifiers in the monadic theory of the binary tree*, in Automata, Languages and Programming, 14th International Colloquium, ICALP87, Lecture Notes in Computer Science Nr. 267, Springer, 1987, pp. 269–279.

[11] P. HUNTER, *Complexity and Infinite Games on Finite Graphs*, PhD thesis, University of Cambridge, 2007.

[12] D. JANIN AND I. WALUKIEWICZ, *On the expressive completeness of the propositional mu-calculus with respect to monadic second order logic*, in Proceedings of 7th International Conference on Concurrency Theory CONCUR '96, no. 1119 in Lecture Notes in Computer Science, Springer-Verlag, 1996, pp. 263–277.

[13] A. KANAMORI, *The Higher Infinite*, Springer, 1991.

[14] A. KECHRIS, *Classical Descriptive Set Theory*, Springer, 1995.

[15] R. MAULDIN, ed., *The Scottish Book. Mathematics from the Scottish Café*, Birkhäuser, 1981.

[16] F. MOLLER AND A. RABINOVITCH, *Counting on CTL*: On the expressive power of monadic path logic*, Information and Computation, (2003). To appear.

[17] A. MOSTOWSKI, *Games with forbidden positions*, Tech. Rep. Tech. Report 78, University of Gdansk, 1991.

[18] J. MYCIELSKI, S. ŚWIERCZKOWSKI, AND A. ZIEBA, *On infinite positional games*, Bull. Acad. Polon. Sci, 4 (1956), pp. 485–488.

[19] J. OXTOBY, *The banach-mazur game and banach category theorem*, in Contributions to the Theory of Games, Vol. III, vol. 39 of Annals of Math. Studies, Princeton, 1957, pp. 159–163.

[20] M. PISTORE AND M. VARDI, *The planning spectrum — one, two, three, infinity*, in Proc. 18th IEEE Symp. on Logic in Computer Science, 2003.

[21] R. TELGÁRSKY, *Topological games: On the 50th anniversary of the Banach-Mazur game*, Rocky Mountain J. Math., 17 (1987), pp. 227–276.

[22] W. THOMAS, *A combinatorial approach to the theory of omega-automata*, Information and Control, 48 (1981), pp. 261–283.

[23] D. VARACCA AND H. VÖLZER, *Temporal logics and model checking for fairly correct systems*, in Proc. 21st IEEE Symp. on Logic in Computer Science, 2006, pp. 389–398.

[24] W. ZIELONKA, *Infinite games on finitely coloured graphs with applications to automata on infinite trees*, Theoretical Computer Science, 200 (1998), pp. 135–183.

Foundations of Software Technology and Theoretical Computer Science (Bangalore) 2008.
Editors: R. Hariharan, M. Mukund, V. Vinay; pp 383-414

Harnessing the Multicores: Nested Data Parallelism in Haskell

Simon Peyton Jones[1], **Roman Leshchinskiy**[2], **Gabriele Keller**[2], **Manuel M. T. Chakravarty**[2]

[1]Microsoft Research Ltd, Cambridge, England, simonpj@microsoft.com

[2]Programming Languages and Systems, School of Computer Science and Engineering, University of New South Wales, {rl,keller,chak}@cse.unsw.edu.au

ABSTRACT. If you want to program a parallel computer, a purely functional language like Haskell is a promising starting point. Since the language is pure, it is by-default safe for parallel evaluation, whereas imperative languages are by-default unsafe. But that doesn't make it easy! Indeed it has proved quite difficult to get robust, scalable performance increases through parallel functional programming, especially as the number of processors increases.
A particularly promising and well-studied approach to employing large numbers of processors is data parallelism. Blelloch's pioneering work on NESL showed that it was possible to combine a rather flexible programming model (nested data parallelism) with a fast, scalable execution model (flat data parallelism). In this paper we describe Data Parallel Haskell, which embodies nested data parallelism in a modern, general-purpose language, implemented in a state-of-the-art compiler, GHC. We focus particularly on the vectorisation transformation, which transforms nested to flat data parallelism.

1 Introduction

Computers are no longer getting faster; instead, we will be offered computers containing more and more CPUs, each of which is no faster than the previous generation. As the number of CPUs increases, it becomes more and more difficult for a programmer to deal with the interactions of large numbers of threads. Moreover, the physical limitations of bus bandwidth will mean that memory access times will be increasingly non-uniform (even if the address space is shared), and locality of reference will be increasingly important.

In the world of massively-parallel computing with strong locality requirements there is already a well-established, demonstrably successful brand leader, namely *data parallelism*. In a data-parallel computation one performs the *same* computation on a large collection of *differing* data values. Well-known examples of data-parallel programming environments are High Performance Fortran (HPF) [For97], the collective operations of the Message Passing Interface (MPI) [GHLL+98], NVIDIA's Compute Unified Device Architecture (CUDA) API for graphics processors [NVI07], and Google's map/reduce framework [DG04].

All these systems support only *flat* data parallelism, in which the computation that is performed on each data element must itself be (a) sequential and (b) of a similar execution time to the computation on the other data elements. In practice, this severely limits the applications of data-parallel computing, especially for sparse or irregular problems [PCS99].

FSTTCS 2008
IARCS Annual Conference on
Foundations of Software Technology and Theoretical Computer Science
http://drops.dagstuhl.de/opus/volltexte/2008/1769

```
(!:)        :: [:a:] -> Int -> a
sliceP      :: [:a:] -> (Int,Int) -> [:a:]
replicateP  :: Int -> a -> [:a:]
mapP        :: (a->b) -> [:a:] -> [:b:]
zipP        :: [:a:] -> [:b:] -> [:(a,b):]
zipWithP    :: (a->b->c) -> [:a:] -> [:b:] -> [:c:]
filterP     :: (a->Bool) -> [:a:] -> [:a:]

concatP     :: [:[:a:]:] -> [:a:]
concatMapP  :: (a -> [:b:]) -> [:a:] -> [:b:]
unconcatP   :: [:[:a:]:] -> [:b:] -> [:[:b:]:]
transposeP  :: [:[:a:]:] -> [:[:a:]:]
expandP     :: [:[:a:]:] -> [:b:] -> [:b:]

combineP    :: [:Bool:] -> [:a:] -> [:a:] -> [:a:]
splitP      :: [:Bool:] -> [:a:] -> ([:a:], [:a:])
```

Figure 1: Type signatures for parallel array operations

Thus motivated, Blelloch and Sabot developed the idea of *nested* data parallelism in the early 90's, and embodied it in their language NESL [BS90].

NESL was a seminal breakthrough but, fifteen years later it remains largely un-exploited. Our goal is to adopt the key insights of NESL, embody them in a modern, widely-used functional programming language, namely Haskell, and implement them in a state-of-the-art Haskell compiler (GHC). The resulting system, Data Parallel Haskell, will make nested data parallelism available to real users.

Doing so is not straightforward. NESL a first-order language, has very few data types, was focused entirely on nested data parallelism, and its implementation is an interpreter. Haskell is a higher-order language with an extremely rich type system; it already includes several other sorts of parallel execution; and its implementation is a compiler.

This paper makes two main contributions:

- We give a tutorial, programmer's-eye view of what programming in Data Parallel Haskell is like. Rather than a series of tiny examples, we give a serious application that is very hard to fully parallelise in a flat data-parallel setting, namely the Barnes-Hut algorithm for N-body simulation.
- We give a detailed tutorial overview of the key vectorisation transformation. There are two major innovations over NESL: one is the non-parametric representation of arrays (Section 4) and one is the treatment of first-class functional values (Section 5).

All the technical innovations in this paper have appeared, piecemeal, in our earlier publications. Our hope, however, is that this paper draws together a somewhat-complex set of technical strands into a comprehensible whole.

2 The programmer's view on DPH

We begin by describing Data Parallel Haskell (DPH) purely from the point of view of the programmer, illustrating the description with a non-trivial example, the Barnes-Hut algorithm [BH86]. GHC supports other forms of concurrency besides data parallelism, but we focus here exclusively on the latter. Singh [SJ08] gives a tutorial covering a broader scope, including semi-implicit parallelism (`par`), explicit threads, transactional memory, as well as Data Parallel Haskell.

DPH is simply Haskell with the following extra features:

- A type of *parallel arrays*, denoted `[:e:]` for arrays of type `e`. These arrays are indexed by values of type `Int`. From a semantic point of view an array `[:a:]` is very similar to a list `[a]` – the difference is in the execution pragmatics. An array can contain elements of any type, including arrays and functions.
- A large number of *parallel operations* that operate collectively on entire arrays. As far as possible, these operations have the same names as Haskell's standard list functions, but with the suffix `P` added—i.e., `mapP`, `filterP`, `unzipP`, and so forth. Figure 1 lists the operations that we will use in this paper.
- Syntactic sugar, called *parallel array comprehensions*, which are similar to list comprehensions but operate on parallel arrays.

In addition to the parallel evaluation semantics, lists and parallel arrays also differ with respect to strictness: more precisely, demand for *any* element of a parallel array results in the evaluation of *all* elements.

2.1 *N*-Body Barnes-Hut Simulation Algorithm

We will demonstrate the use of DPH features using the Barnes-Hut n-body simulation algorithm as an example. We discuss the algorithm in some detail because it is a particularly striking example of the power of nested data parallelism, and of the utility of user-defined data types in data-parallel programs. We will, for the sake of clarity, restrict ourselves to two dimensions and neglect complications such as bodies that are very close to each other.

An n-body simulation computes the motion under gravitational forces of n bodies, or *particles*. A naive solution is to compute the force between every pair of particles which requires n^2 calculations in each time step. The Barnes-Hut algorithm reduces the work complexity to the order of $n \log n$ interactions by grouping together particles which are close to each other and calculating the centre of gravity, or *centroid* of the cluster. The centroids are then used to approximate the effect the particles have on other particles which are sufficiently far away. The stricter we are in determining what exactly constitutes "sufficiently far away", the more precise the final result is, and the algorithm can be parametrised accordingly.

The first phase of the algorithm determines the hierarchical grouping of the particles, computes the centroids of the clusters, and stores the result in a tree structure. To be more precise, the area is split into four subareas of equal size, the particles are grouped according to the subarea they are located in. We repeat this step for each subarea, and terminate if an area contains either none or only a single particle. Figure 2 illustrates the tree construction process for particles $p_1 \ldots p_9$. In the first iteration, the particles are split into four groups,

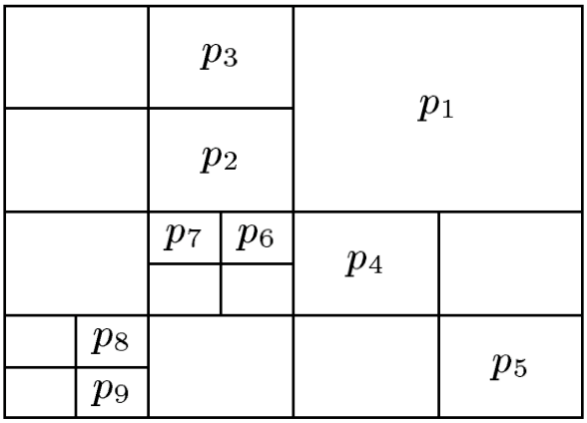


Figure 2: Subdivision of area

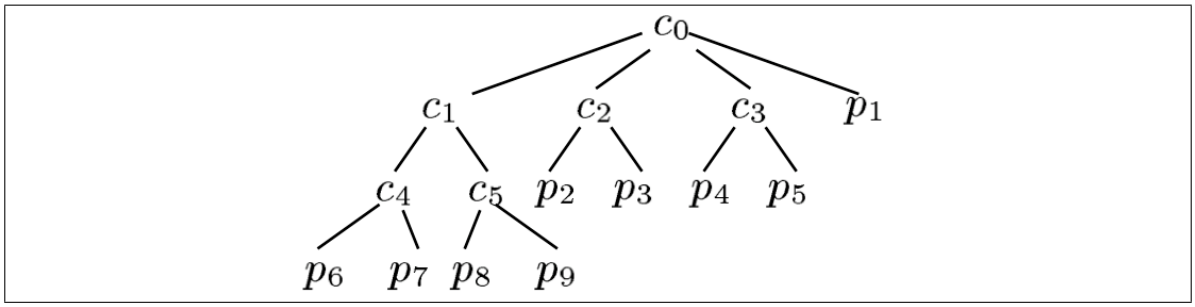


Figure 3: Rosetree

depending on which quadrant they are located in. The upper right quadrant already contains only a single particle, so it isn't divided up any further. Both the upper left and the lower right quadrant require only one more iteration, the lower left two iterations.

Figure 3 shows the resulting hierarchical tree structure: the root node contains the centroid of all particles and four subtrees (since all subareas contain at least one particle). Each of the subtrees contains the centroid of the corresponding subarea – which is the particle itself in case of a singular particle.

The second phase of the algorithm now calculates the forces that affect each particle p, by traversing the tree from the root downwards: for every subtree, if the particle p is sufficiently far away from the centroid stored in the root of that subtree, the force on p is calculated using this centroid without looking at the rest of the tree. Otherwise, we add up the forces on p from the subtrees of the current root – and so on recursively.

2.2 Encoding Barnes-Hut in DPH

Since the only way to express parallelism in DPH is to apply collective operations to parallel arrays, we need to store all data that we want to process in parallel in such an array. For instance, the function `oneStep`, which computes one step in the simulation, takes a parallel array of particles as arguments, and returns an array of the same length, with the position and velocity of each particle adjusted according to the gravitational forces:

```
-- Compute one step of the n-body simulation
oneStep :: [:Particle:] -> [:Particle:]
oneStep particles = moveParticles particles forces
  where
    tree   = buildTree initialArea particles
    forces = calcForces (lengthOf initialArea) tree particles

buildTree     :: Area -> [:Particle:] -> Tree
calcForces    :: Float -> Tree -> [:Particle:] -> [:Force:]
moveParticles :: [:Particle:] -> [:Force:] -> [:Particle:]
lengthOf      :: Area -> Float
```

The function `oneStep`, as discussed before, is comprised of three data parallel phases. First, `buildTree` decomposes the particles into sub-areas, returning the resulting `Tree`. This tree is then used by `calcForces` to compute the forces on the particles, returning a new array of forces with one element for each particle. Finally, `moveParticles` uses these forces to adjust the positions and velocities the particles.

The data types involved in the computation are defined exactly as the would be in regular Haskell. For example, a `Particle` is a record of its mass, its location, and its velocity:

```
type Vector   = (Float, Float)
type Area     = (Vector, Vector)
type Force    = Vector
type Velocity = Vector
type Location = Vector

data Particle = Particle { mass     :: Float
                         , location :: Location
                         , velocity :: Velocity}
```

Some functions are conveniently defined using the parallel array counterparts of ordinary list processing functions (see Figure 1). For example, we can define `moveParticles` like this:

```
moveParticles :: [:Particle:] -> [:Force:] -> [:Particle:]
moveParticles ps fs = zipWithP moveParticle ps fs

moveParticle :: Particle -> Force -> Particle
moveParticle (Particle { mass     = m
                       , location = loc
                       , velocity = vel })
             force
  = Particle { mass     = m
             , location = loc + vel   * timeStep
             , velocity = vel + accel * timeStep }
  where
    accel = force / m
```

Now we turn our attention to the `Tree` data type and its construction. When building and traversing a `Tree`, we want to process its sub-trees in parallel, and so we must use a parallel array for the children:

```
data Tree = Node Mass Location [:Tree:]
            -- Rose tree for spatial decomposition
```

This time, unlike the flat array of particles (which may be very long), the array of sub-trees has at most four elements at any level (recall that we are working with only 2 dimensions). To build a tree, we perform recursive descent over the area:

```
-- Perform spatial decomposition and build the tree
buildTree :: Area -> [:Particle:] -> Tree
buildTree area [: p :]   = Node (mass p) (location p) [::]
buildTree area particles = Node m l subtrees
  where
    (m,l)    = calcCentroid subtrees
    subtrees = [: buildTree a ps
                | a <- splitArea area
                , let ps = [:p | p <- particles, inArea a p:]
                , lengthP ps > 0 :]


inArea :: Area -> Particle -> Bool
inArea ((lx,ly),(hx,hy)) (Particle { location = (x,y) })
  = lx <= x && x <= hx && ly <= y && y <= hy

splitArea    :: Area -> [:Area:]
   -- splitArea returns the four sub-areas in a parallel array
calcCentroid :: [:Tree:] -> (Mass, Location)
```

The first equation deals with the case of a single particle: we simply record its mass and location. In the recursive case, the array comprehension for `subtrees` iterates in parallel over `(splitArea area)`, an array of exactly four elements. For each such area `a`, we compute the set of particles `ps` that lie inside `a` and, if that set is non-empty, we recursively call `buildTree`. The "if non-empty" test discards sub-areas which do not contain any particles at all, so the length of `subtrees` can be anything between 1 and 4. We omit the implementations of `inArea` and `calcCentroid`, since they are straightforward.

The nested comprehension in the where clause of `buildTree` makes sure that `inArea` is called on every subarea/particle combination in a single parallel step. Another source of nested parallelism in `buildTree` are the recursive calls to the parallel function `buildTree`, which are performed simultaneously on however many sub-areas contain particles (from one to four). The number of parallel steps is hence proportional to the depth of the rose tree.

Lastly, we have to write the function `calcForces`, which, given a `Tree` and an array of particles, calculates the forces applied by the `Tree` on those particles. It can do so by dividing the particles into two groups: those that are "far" from the centre of gravity of the `Tree` (as determined by a function `isFar`), and those that are "near". Here is the code:

```
calcForces :: Float -> Tree -> [:Particle:] -> [:Force:]
calcForces len (Node m l ts) ps
  = let
      far_forces     = [: forceOn p m l | p <- ps, isFar len l p :]
      near_ps        = [: p | p <- ps, not (isFar len l p) :]
      near_forces_s  = [: calcForces (len / 2) t near_ps | t <- ts :]
      near_forces    = [: sumForces p_forces
                        | p_forces <- transposeP near_forces_s :]
    in
    combineP [:isFar len l p | p <- ps:] far_forces near_forces
```

```
forceOn    :: Particle -> Mass -> Location -> Force
isFar      :: Float -> Location -> Particle -> Bool
sumForces  :: [:Force:] -> Force
```

The function `calcForces` divides the particles into two groups: those that are "near" the centroid `l` of tree, and those that are not. For the far particles, we simply use `forceOn` to compute the force on each such particle from the tree, giving `far_forces`. For particles near to `l`, `near_ps`, we recursively use `calcForces` (in parallel) to compute the force on each particle from each sub-tree giving `near_forces_s`, a short vector with one element for each sub-tree. Each element is a vector with one element for each particle, giving the force on that particle from the sub-tree. All that remains is to transpose this nested structure, and add up the forces on each particle. Finally, we must re-combine the near and far forces, using `combineP`, which interleaves two vectors as directed by a boolean mask.

2.3 Communication and locality

Here is an alternative, simpler way to write `calcForces`:

```
calcForces :: Tree -> [:Particle:] -> [:Force:]
calcForces tree ps = mapP (calc t) ps
  where
    calc (Node m l ts) p
      | isFar l p = forceOn p m l
      | otherwise = sumForces [: calc t p | t <- ts :]
```

For each particle (the `mapP`), it recurses down the tree, stopping when the centroid of the sub-tree is far away from the particle.

Which version should we prefer? Different ways of writing the code give rise to different patterns of data communication. In this latter version you can see that every particle needs a copy of (at least the top part of) the tree, so the danger here is that most of the tree ends up being copied to most of the processors. In the earlier version, the particles migrate (in smaller and smaller groups) to the tree, rather than the other way around.

It undoubtedly complicates the programmer's life to have to think about these matters, but there is no silver bullet. Parallel programming is complicated, and programmers *must* think about concurrency and communication, as well as correctness. However, one of the advantages of the data-parallel style is that it gives us a much better handle on the program's *cost model* (both computation and communication) than un-structured parallel programming [Ble96].

2.4 Summary

The algorithm we have described makes extensive use of data parallelism. For example, `buildTree` is called in parallel on the four sub-areas of the area under consideration; and for each of those sub-areas we compute the relevant subset of the particles in parallel. Similarly `calcForces` is called in parallel on the four sub-trees; and the computation of `far_forces` is done in parallel over all the particles. In each case, the recursive calls over the sub-trees express *nested* data parallelism, because the computation that is performed

on the sub-tree is itself a data-parallel computation. This is really quite difficult to express using flat data parallel frameworks; indeed tree construction is often not parallelised.

The rest of this paper uses the parallel Barnes-Hut algorithm as a running example to explain the successive steps through which the program is compiled to run efficiently on parallel shared-memory machines.

3 Compiling DPH programs

The compiler must translate high-level nested data parallel programs, as described in the previous section, into efficient low-level code. This translation consists of four main steps:

- *Desugaring* removes syntactic sugar, reducing the program to a simple lambda language. This intermediate language, GHC's "Core" language, is still strongly typed.
- *Vectorisation* transforms *nested* data parallelism into *flat* data parallelism; it is a Core-to-Core transformation.
- *Fusion* optimises the Core program, by eliminating redundant synchronisation points and intermediate arrays, thus dramatically improves locality of reference;
- *Gang parallelism* divides the parallel operations spatially into chunks, each chunk being executed by a thread from a *gang* of threads. Typically a gang contains a thread for each CPU. Gang parallelism is expressed by giving library implementations of the "vector instructions", rather than by built-in compiler support.

GHC implements these steps using a large number of Core-to-Core program transformations. Many of these transformations have been part of GHC's optimiser for a long time, in particular a sophisticated inliner, worker-wrapper unboxing, and constructor specialisation [Pey96, PM02, PL91, PTH01]. In the course of the Data Parallel Haskell project, we are adding more, array-specific transformations. Due to GHC's generic support for program transformations — specifically, the inliner and rewrite rules [PM02, PTH01] — we can implement most of these new transformations as library code, as opposed to extending the compiler itself. Indeed, apart from the vectorisation pass, the rest of the optimisation pipeline operates in ignorance of the fact that the program being optimised is a data parallel one.

In this paper we focus mainly on vectorisation, starting at Section 3.2, after taking a brief diversion to describe how array comprehensions are desugared (Section 3.1).

3.1 Desugaring array comprehensions

In the Barnes-Hut code we used both *array comprehensions*, and *ordinary functions* over parallel arrays such as `zipP` and `mapP`. However, just as in the case of list comprehensions, the former is just a convenient syntactic sugar for the latter. More precisely, Figure 4 gives rules for desugaring array comprehensions. They are quite standard [JW07], and practically identical to those for lists, so we do not discuss them further. These rules are simple, but they should be thought of as a specification rather than an implementation, because they generate somewhat inefficient code. In GHC's actual implementation we use slightly more complicated rules.

$$
\begin{array}{lrcl}
\textbf{Expressions} & e & ::= & \ldots \mid \texttt{[:}e \mid q\texttt{:]} \\
\textbf{Qualifiers} & p, q & ::= & x\texttt{<-}e \mid e \mid p\texttt{,}q \mid p \texttt{|} q
\end{array}
$$

$$\mathcal{D}[\![\texttt{[:}e \texttt{|} q\texttt{:]}]\!] = \texttt{mapP}\ (\lambda q_v.e)\ \mathcal{Q}[\![q]\!]$$

$\mathcal{Q}[\![q]\!]$ computes the parallel array of the tuples generated by q

$$
\begin{array}{rcl}
\mathcal{Q}[\![x\texttt{<-}e]\!] & = & e \\
\mathcal{Q}[\![e]\!] & = & \texttt{if}\ e\ \texttt{then [:():] else [::]} \\
\mathcal{Q}[\![p\texttt{,}q]\!] & = & \texttt{concatMapP}\ (\lambda p_v.\texttt{mapP}\ (\lambda q_v.(p_v, q_v))\ \mathcal{Q}[\![q]\!])\mathcal{Q}[\![p]\!] \\
\mathcal{Q}[\![p \texttt{|} q]\!] & = & \texttt{zipP}\ \mathcal{Q}[\![p]\!]\ \mathcal{Q}[\![q]\!]
\end{array}
$$

q_v is a tuple of the variables bound by q

$$
\begin{array}{rcl}
(x\texttt{<-}e)_v & = & x \\
(g)_v & = & () \\
(p\texttt{,}q)_v & = & (p_v, q_v) \\
(p \texttt{|} q)_v & = & (p_v, q_v)
\end{array}
$$

Figure 4: Desugaring rules for array comprehensions

3.2 Informal overview of vectorisation

The purpose of vectorisation is to take a program that uses nested data parallelism, and transform it into a program that uses only flat data parallelism. Consider this tiny example

```
f :: Float -> Float
f x = x*x + 1
```

For every such function we build its *lifted* version $\texttt{f}_\texttt{L}$ thus:

```
f_L :: [:Float:] -> [:Float:]
f_L x = (x *_L x) +_L (replicateP n 1)
        where
          n = lengthP x
```

Internally, $\texttt{f}_\texttt{L}$ uses "vector instructions" like $+_\texttt{L}$ to do its work, where

```
+_L :: [:Float:] -> [:Float:] -> [:Float:]
```

Notice that it must also replicate the constant 1 so that argument has the type that $+_\texttt{L}$ expects. So, roughly speaking (we give the true story later), to form the definition of $\texttt{f}_\texttt{L}$ we transform the body of `f` in the following way:

- Replace a constant by a call to `replicateP`.
- Replace a function by its lifted versions (e.g. + becomes $+_\texttt{L}$).
- Replace a parameter (e.g. `x`) by itself.

This new definition obeys the equation $\texttt{f}_\texttt{L}$ = `mapP f`, so it takes an array to an array. In effect, it is a specialised variant of `mapP` – specialised by fixing the function argument. The idea is that whenever we see the call (`mapP f`) we will replace it by $\texttt{f}_\texttt{L}$. But there is a problem! Suppose we have

```
g :: [:Float:] -> [:Float:]
g xs = mapP f xs
```

First we replace (`mapP f`) by f_L to get:

```
g :: [:Float:] -> [:Float:]
g xs = fL xs
```

But now we must lift `g` too, in case there are calls to (`mapP g`). If we try, we get this:

```
gL :: [:[:Float:]:] -> [:[:Float:]:]
gL xs = fLL xs
```

Not good: we need the *doubly-lifted* version of `f`! If the depth of nesting is not statically bounded (and it isn't in Barnes-Hut) then we are in trouble. Blelloch's clever solution is to observe that we can define f_{LL} in terms of f_L, thus:

```
fLL :: [:[:Float:]:] -> [:[:Float:]:]
fLL xss = unconcatP xss (fL (concatP xss))
```

That is: first concatenate all the rows of `xss` to make a single flat vector; then map `f` over that vector; then chop up the result to form a vector of vectors again, guided by the original shape of `xss`. (Note that the incoming vector might well be "ragged", so that not all the sub-vectors have the same length.) At first, this idea looks terribly inefficient, because of all the flattening and un-flattening but, as we shall see, if we choose the right data representation, `concatP` and `unconcatP` take constant time and involve no copying.

This is the core of the vectorisation transformation. We have left many details vague. What about higher order functions? What about user-defined data types? We now start to tighten our description up. We begin by discussing how to represent arrays (Section 4) and functions (Section 5) in vectorised code. These representation choices in turn drive the vectorisation transformation (Section 6). More details are given in previous work [KC98, CK00, LCK06, CLP+07].

4 Representing arrays in vectorised code

Standard arrays in Haskell are parametric; i.e., the array representation is independent of the type of array elements. This is achieved by using arrays of pointers referring to the actual element data. Such a *boxed* representation is very flexible, but it is also detrimental to performance. The indirections consume additional memory, increase memory traffic, and decrease locality of memory access. The resulting runtime penalty can be very significant.

The parallel arrays `[:a:]` offered by the DPH *source* language are also parametric, as can be seen from the polymorphic type signatures in Figure 1. One of the tasks of the vectoriser is to change the array representation, by systematically transforming a function that manipulates values of type `[:(Int,Int):]`, say, to one that manipulates values of type `PA (Int,Int)`. These new `PA` arrays have a *non-parametric* representation; that is, *the representation depends on the element type* [CK00]. For example, a value of type `PA Int` is held as a contiguous memory area containing unboxed 32-bit integer values — not as a block of pointers to `Int`-valued thunks, as is the case in vanilla Haskell.

Although `PA` is not visible to the user, such non-parametric data types are an indepenently-useful source-language feature, already implemented in GHC, which we call an *associated data type* [CKPM05]. We will therefore explain `PA` using the notation of associated data types.

Since the representation of an array depends on the type of its elements, there can be no useful polymorphic functions over `PA`. For example, we cannot define

```
lengthPA :: PA a -> Int    -- WRONG!
```

because `lengthPA`, being polymorphic in `a`, knows nothing about the representation of `PA a`. This is just what type classes are for. So we declare the type `PA` in association with a class `PAElem` that defines operations over the type, thus:

```
class PAElem a where
  data PA a
  indexPA     :: PA a -> Int -> a
  lengthPA    :: PA a -> Int
  replicatePA :: Int -> a -> PA a
  ...more operations...
```

Given a type `a` that is allowed to be an element of a parallel array, there is a corresponding data type `PA a`, and operations `indexPA`, `lengthPA`, `replicatePA`, and so on. These operations therefore have overloaded types, thus:

```
indexPA  :: PAElem a => PA a -> Int -> a
lengthPA :: PAElem a => PA a -> Int
...etc...
```

All the parametric operations of Figure 1 have `PA` variants with the same types apart from the additional `(PAElem a)` constraint. (Our concrete implementation is more complex with more operations, but the code shown here conveys the basic idea.)

An instance declaration fills in an implementation for each of these elements. For example, the instance declaration for integers takes the following form:

```
class PAElem Int where
  data PA Int = AInt ByteArray
  indexPA (AInt ba) i = indexIntArray ba i
  lengthPA (AInt ba) = lengthIntArray ba
  replicatePA n i = AInt (replicateIntArray n i)
  ...more operations...
```

We represent the array by a contiguous region of bytes (aka `ByteArray`) with primitives such as `indexIntArray` that operate on individual 32-bit integers from a `ByteArray`. (The code again simplifies the concrete implementation by omitting the use of unboxed types.)

4.1 Arrays of structured data

The `PAElem` instance for `Float`, and other primitive types, follows the same pattern. But what about more complex data structures, such as an array of pairs? It is quite unacceptable to represent it by an array of pointers to (heap-allocated) records, because the indirection costs would be too heavy. Instead, we represent it by a *pair of arrays*:

```
class (PAElem a, PAElem b) => PAElem (a, b) where
  data PA (a,b) = ATup2 Int (PA a) (PA b)
  indexPA (ATup2 _ arr1 arr2) i = (indexPA arr1 i, indexPA arr2 iv)
  lengthPA (ATup2 n _ _) = n
```

Thus, a `PA (Float,Float)` is represented by a pair of unboxed arrays, each storing a vector of floating point values. Crucially, the two arrays must have the same length; and we record that length in the `Int` field of the $\texttt{ATup}_2$ constructor. This length field is convenient, but usually redundant — but not always! Consider an array of `()` elements:

```
class PAElem () where
  data PA () = ATup0 Int
  indexPA (ATup0 _) i = ()
  lengthPA (ATup0 n) = n
```

We need no data storage to store a vector of `()` values, but we must still remember its length.

Notice that the representation is *compositional*; that is, the representation of an array of pairs is given by combining the representations of an array of the first and second elements of the pair, and so on recursively.

The representation also allows us to combine two arrays element-wise into an array of pairs *in constant time*, with unzipping being equally easy:

```
zipPA :: PAElem a => PA a -> PA b -> PA (a,b)
zipPA as bs = ATup2 (lengthPA as) as bs

unzipPA :: PA (a,b) -> (PA a, PA b)
unzipPA (ATup_2 _ as bs) = (as,bs)
```

This stands in contrast to lists, where zipping and unzipping take linear time.

Lastly, since records are converted into product types by the desugarer, the `Particle` arrays in Barnes-Hut are represented by tuples of arrays.

4.2 Nested arrays

Even more interesting is the representation of nested arrays. A classic example is that of *sparse matrices*, in which we represent a sparse matrix as a vector of rows, each row consisting of a vector of (index,value) pairs, where only the non-zero values in the row are represented. Thus

```
type SparseMatrix a = [:[:(Int,a):]:]
```

Since our ultimate goal is to eliminate nested parallelism, it is not surprising that we also want to represent nested arrays in terms of flat ones. Indeed, a nested array `PA (PA a)` can be encoded by

- a flat *data array* of type `PA a` which contains the data elements and
- a *segment descriptor* of type `PA (Int, Int)` which stores the starting position and length of the subarrays embedded in the flat data array.

This is captured by the following instance:

```
class PAElem a => PAElem (PA a) where
  data PA (PA a) = AArr (PA a) (PA (Int, Int))
  indexPA (AArr arr segd) i = slicePA arr (indexPA segd i)
  lengthPA (AArr _ seg) = lengthPA seg
```

where `sliceP` extracts a subarray from a larger array in constant time. Thus, the sparse matrix

```
[:[:(0,15),(2,9),(3,20):], [::], [:(3,46):]:]
```

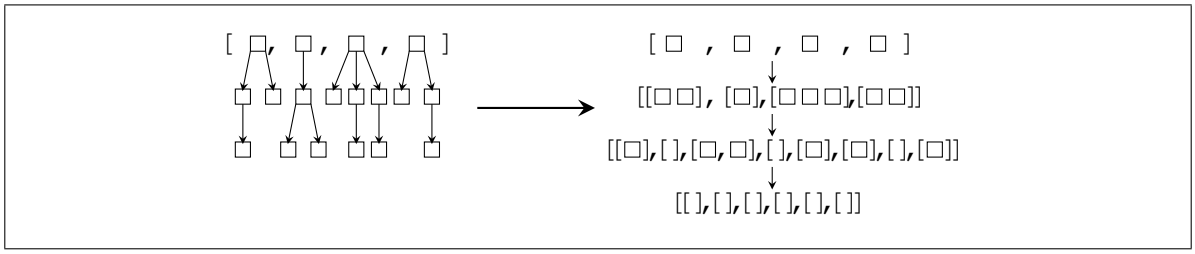


Figure 5: Value of type `[:Tree:]` and its vectorised representation

will be represented as

```
AArr (ATup2 [#0,2,3,3#] [#15,9,20,46#]) -- Data
     (ATup2 [#0,3,3#] [#3,0,1#])        -- Segment descriptor
```

where we write `[#...#]` for a literal `ByteArray`. The first `ByteArray` contains all the column indexes, the second one all the `Float`s, and the third and fourth the start indexes and lengths of the segments, respectively. Since all four `ByteArray` are unboxed, programs which process such matrices can be compiled to highly efficient code.

Remarkably, we can now give constant-time implementations of the two functions `concatPA` and `unconcatPA`, as promised in Section 3.2:

```
concatPA :: PA (PA a) -> PA a
concatPA (AArr cts _) = cts

unconcatPA :: PA (PA a) -> PA b -> PA (PA b)
unconcatPA (AArr _ shape) cts = AArr cts shape
```

4.3 Recursive types

If the array elements are recursive, the non-parametric representation of the array has to be recursive, too. For instance, arrays of `Tree` from Section 2.2 are represented as follows:

```
instance PAElem Tree where
  data PA Tree = ATree Int (PA Mass) (PA Location) (PA (PA Tree))
```

Since `Tree` is a product, the representation is similar to arrays of tuples. It stores the masses and locations of the centroids and a nested array containing the subtrees of each node. As described in the previous section, the latter is encoded by a flat array of trees together with a segment descriptor. In effect, this means that an array of trees is represented by a list with each element containing the centroids and segmentation information for one tree level. This allows all data in one level to be processed in parallel, although the levels have to be processed one after another. Figure 5 illustrates this representation. User-defined types are discussed in more detail in Section 6.4.

4.4 Polymorphism

If we were only interested in *monomorphic* code, or if we would use a whole-program compiler that specialises a polymorphic to a monomorphic program, as was the case in NESL, then life would have been much easier. We could implement the non-parametric array type

by *statically replacing* `PA Int` by `PAInt`, say, where the latter is a perfectly ordinary data type, defined as

```
data PAInt = AInt ByteArray
```

Now we do not need non-parametric types; in the vectorised code, original types `[:Int:]`, `[:(Int,Float):]`, etc, are simply replaced by `PAInt`, `PAPair PAInt PAFloat`, and so on, where all these are ordinary data types.

Alas, this does not work for *polymorphic* functions. For example, how could we translate the type of this function? What would we statically replace `[:a:]` by?

```
firstRow :: [:[:a:]:] -> [:a:]
```

We also cannot turn polymorphic functions into families of monomorphic functions, as we support separate compilation and polymorphic recursion. No — if we want polymorphism, we must use something akin to type classes, as we have described in this section. A key component of our work is the extension of the non-parametric representation idea to work in a polymorphic setting. In particular, our Core language regards `PA` as a type-level function from types to types [SCPD07].

5 Representing functions in vectorised code

Haskell is a higher order language, so we have to consider how to vectorise programs that manipulate functions. Vectorising higher-order programs raises two distinct problems.

5.1 Functions are pairs

Consider this (contrived) definition:

```
ho :: (Int->Bool) -> (Bool, [:Bool:])
ho f = (f 2, mapP f [:1,2,3:])
```

In our overview (Section 3.2), we said that we should replace `(mapP f)` with a call to $\texttt{f}_\text{L}$, the lifted version of `f`. But since `f` is lambda-bound, it is not so easy to call "the lifted version of `f`". Clearly the caller must pass the lifted version of `f` as a parameter to `ho`. But there is an ordinary, scalar call `(f x)` in the body of `ho`, so we can't pass *only* the lifted version. The obvious alternative is to pass a pair that gives *both* the lifted *and* unlifted versions. With such a representation, `mapP` can just extract the lifted variant of its argument (a pair), while the vanilla application of `f` extracts the unlifted variant.

5.2 Functions are closures

There is a second challenge. In Section 4 we discussed the efficient, non-parametric representation of data-parallel arrays. A higher order language forces us to confront the question of how to represent an array of *functions*. For example:

```
distance :: [:Float:] -> [: Float->Float :]
distance xs = mapP (λx y. sqrt (x*x + y*y)) xs

distY :: [:Float:] -> Float -> Float
distY xs y = sumP [: d y | d <- distance xs :]
```

There are more direct ways to write `distY`, of course, but arrays of functions also arise inevitably when we think about lifting. If we start with

```
f :: (Int->Int) -> Int
```

then the lifted version of `f` has the type

```
f_L :: [: Int->Int :] -> [:Int:]
```

How should we represent such an array of functions? A possible answer is "as an array of pointers to function closures". Bad answer! In data-parallel computations, all the processors are supposed to execute the same code in parallel, but if every function closure in the array is potentially different, they clearly cannot do that. Furthermore, a pointer-based representation destroys locality.

Happily, there is no need for the generality of a distinct function pointer for each array element. Consider the result of `distance`, for example. Every element of this array is a function *with the same code*, but a different value for the function's free variable `x`. This makes the solution obvious: we must represent an array of functions by a pair of a *single* code pointer and an *array* of environment records, which give the per-element bindings for the free variables.

5.3 Putting it together

Putting our two solutions together, we see that in vectorised code a function must be represented by a triple:

1. The scalar version of the function
2. The lifted version of the function
3. An environment record of the free variables of the function

To be concrete, here is the data type declaration for vectorised functions:

```
data (a :-> b) = forall e. PAElem e =>
                 Clo { env  :: e
                     , clo_s :: e -> a -> b
                     , clo_l :: PA e -> PA a -> PA b }
```

This declaration says that `(:->)` is an algebraic data type (written infix), with a single constructor `Clo`. The constructor `Clo` has an existentially-quantified type variable `e`, and three fields, `env`, $\texttt{clo}_s$, and $\texttt{clo}_l$. The vectorisation transformation will transform every function type $\tau_1\texttt{->}\tau_2$ to $\tau_1'\texttt{:->}\tau_2'$, where τ_1' is the transformed version of τ_1 and similarly for τ_2. In effect, the vectorisation transform performs closure conversion [AJ89].

With this definition in hand, we can now explain how arrays of values of type `(a :-> b)` are represented:

```
instance PAElem (a :-> b) where
  data PA (a :-> b) = forall e. PAElem e
                    => AClo { aenv  :: PA e
                            , aclo_s :: e -> a -> b
                            , aclo_l :: PA e -> PA a -> PA b }
  lengthPA (AClo env f_s f_l) = lengthPA env
  indexPA (AClo env f_s f_l) n = Clo (indexPA env n) f_s f_l
  replicatePA n (Clo env f_s f_l) = AClo (replicatePA n env) f_s f_l
```

To represent an array of functions, we keep a *single* code pointer for each of the scalar and lifted code, but have an *array* of environment records. Notice that `Clo` and `AClo` differ *only* in the type of the environment field; their $\texttt{clo}_s$ and $\texttt{clo}_l$ fields are identical.

As in the case of other types, it is worth noting that this representation supports very simple and direct implementations of indexing, replication, and so on. It does *not* efficiently support literal arrays of various different functions, such as `[:sin,cos:]`. This is quite deliberate: in a data-parallel computation all the processors should be performing the same computation at the same time. Nevertheless, such arrays can be handled, essentially using conditionals which ensure that different functions are executed one after another.

In Section 6.2, we will see how `Clo` and `AClo` are used in the transformation of the `ho` example. Before we can do so, we must first specify the vectorisation transformation more precisely.

6 Vectorisation

We are finally ready to discuss the vectorisation transformation itself. Consider a top-level function definition $f :: \tau = e$, where τ is the type of f. The *full vectorisation* transformation produces a definition for the vectorised version of f called f_V, thus:

$$f_\text{V} :: \mathcal{V}_t[\![\tau]\!] = \mathcal{V}\,[\![e]\!]$$

Here, f_V is the *fully vectorised* variant of f, whose right-hand side is generated by the *full vectorisation transform* $\mathcal{V}\,[\![\cdot]\!]$. As we have already seen, vectorisation returns an expression of a different type to the input, so the type of f_V is obtained by vectorising the type τ, thus $\mathcal{V}_t[\![\tau]\!]$. In general, if $e :: \tau$ then $\mathcal{V}\,[\![e]\!] :: \mathcal{V}_t[\![\tau]\!]$. Figure 6 gives the functions for both type and term vectorisation. In our compiler, the transformation applies to an explicitly-typed program, but we omit all type information in Figure 6, in order to concentrate on the essentials.

Vectorisation is applied separately to each top level function in the program, so it is a whole-program transformation. In real programs, only a part will be data-parallel, while much of it is not (e.g. input/output, user interaction etc). We ignore this issue here, but in reality our compiler performs *selective* vectorisation – see Section 6.5 and [CLJK08].

The type transformation $\mathcal{V}_t[\![\tau]\!]$ transforms a source-program type to the corresponding type in the vectorised program. As can be seen in Figure 6, its effect is simple: it transforms every function arrow `(->)` to a vectorised function arrow `(:->)`, and every parametric array constructor `[::]` to a non-parametric parallel array constructor `PA`. A user-defined algebraic data type might have nested uses of `(->)` or `[::]` — for example, `Tree` does so — and for these we must generate a vectorised variant ($\texttt{Tree}_\text{V}$) of the data type itself. We elaborate this point in Section 6.4.

This type transformation forces the vectorised program to differ quite radically from the original. In particular, since a "function" is now a triple constructed with `Clo`, we need an infix application operator `$:` to extract the scalar copy:

```
($:) :: (a :-> b) -> a -> b
($:) (Clo env fs fl) = fs env
```

and a lifted version

$\mathcal{V}_t[\![\tau]\!] :: \mathit{Type} \to \mathit{Type}$ is the vectorisation transformation on types

$$
\begin{array}{rcll}
\mathcal{V}_t[\![\tau_1 \texttt{->} \tau_2]\!] & = & \mathcal{V}_t[\![\tau_1]\!] \texttt{ :->} \mathcal{V}_t[\![\tau_2]\!] & \text{Functions} \\
\mathcal{V}_t[\![\texttt{[:}\tau\texttt{:]}]\!] & = & \mathcal{L}_t[\![\tau]\!] & \text{Parallel arrays} \\
\mathcal{V}_t[\![\texttt{Int}]\!] & = & \texttt{Int} & \text{Primitive scalar types} \\
\mathcal{V}_t[\![\texttt{Float}]\!] & = & \texttt{Float} & \\
\mathcal{V}_t[\![T\ \tau_1 \ldots \tau_n]\!] & = & T_\mathrm{V}\ \mathcal{V}_t[\![\tau_1]\!] \ldots \mathcal{V}_t[\![\tau_n]\!] & \text{Algebraic data types (e.g. lists)}
\end{array}
$$

$$
\mathcal{L}_t[\![\tau]\!] = \mathit{PA}\ \mathcal{V}_t[\![\tau]\!]
$$

$\mathcal{V}[\![e]\!] :: \mathit{Term} \to \mathit{Term}$ is the full vectorisation transformation on terms

Invariant: if $\overline{x_i : \sigma_i} \vdash e : \tau$ then $\overline{x_i : \mathcal{V}_t[\![\sigma_i]\!]} \vdash \mathcal{V}[\![e]\!] : \mathcal{V}_t[\![\tau]\!]$

$$
\begin{array}{rll}
\mathcal{V}[\![k]\!] & = k & k \text{ is a literal} \\
\mathcal{V}[\![f]\!] & = f_\mathrm{V} & f \text{ is bound at top level} \\
\mathcal{V}[\![x]\!] & = x & x \text{ is locally bound (lambda, let, etc)} \\
\mathcal{V}[\![e_1\ e_2]\!] & = \mathcal{V}[\![e_1]\!]\ \texttt{\$:}\ \mathcal{V}[\![e_2]\!] & \\
\mathcal{V}[\![\lambda x.e]\!] & = \texttt{Clo}\ \{\texttt{env} = (y_1, \ldots, y_k) & \\
 & \quad , \texttt{clo}_s = \lambda e\, x.\, \texttt{case}\ e\ \texttt{of}\ (y_1, \ldots, y_k) \to \mathcal{V}[\![e]\!] & \\
 & \quad , \texttt{clo}_l = \lambda e\, x.\, \texttt{case}\ e\ \texttt{of}\ \texttt{ATup}_k\ n\ y_1 \ldots y_k \to \mathcal{L}[\![e]\!]\ n\} & \\
 & \text{where} \{y_1, \ldots, y_k\} = \text{free variables of } \lambda x.e &
\end{array}
$$

$$
\mathcal{V}\left[\!\!\left[\begin{array}{l} \texttt{if}\ e_1 \\ \texttt{then}\ e_2 \\ \texttt{else}\ e_3 \end{array} \right]\!\!\right] = \texttt{if}\ \mathcal{V}[\![e_1]\!]\ \texttt{then}\ \mathcal{V}[\![e_2]\!]\ \texttt{else}\ \mathcal{V}[\![e_3]\!]
$$

$\mathcal{L}[\![e]\!]\ n :: \mathit{Term} \to \mathit{Term} \to \mathit{Term}$ is the lifting transformation on terms

Invariant: if $\overline{x_i : \sigma_i} \vdash e : \tau$ then $\overline{x_i : \mathcal{L}_t[\![\sigma_i]\!]} \vdash \mathcal{L}[\![e]\!]\ n : \mathcal{L}_t[\![\tau]\!]$

where n is the length of the result array

$$
\begin{array}{rll}
\mathcal{L}[\![k]\!]\ n & = \texttt{replicatePA}\ n\ k & k \text{ is a literal} \\
\mathcal{L}[\![f]\!]\ n & = \texttt{replicatePA}\ n\ f_\mathrm{V} & f \text{ is bound at top level} \\
\mathcal{L}[\![x]\!]\ n & = x & x \text{ is locally bound (lambda, let, etc)} \\
\mathcal{L}[\![e_1\ e_2]\!]\ n & = \mathcal{L}[\![e_1]\!]\ n\ \texttt{\$:}_\mathrm{L}\ \mathcal{L}[\![e_2]\!]\ n & \\
\mathcal{L}[\![\lambda x.e]\!]\ n & = \texttt{AClo}\ \{\texttt{aenv} = \texttt{ATup}_k\ n\ y_1 \ldots y_k, & \\
 & \quad , \texttt{aclo}_s = \lambda e\, x.\, \texttt{case}\ e\ \texttt{of}\ (y_1, \ldots, y_k) \to \mathcal{V}[\![e]\!] & \\
 & \quad , \texttt{aclo}_l = \lambda e\, x.\, \texttt{case}\ e\ \texttt{of}\ \texttt{ATup}_k\ n'\ y_1 \ldots y_k \to \mathcal{L}[\![e]\!]\ n'\} & \\
 & \text{where} \{y_1, \ldots, y_k\} = \text{free variables of } \lambda x.e &
\end{array}
$$

$$
\begin{array}{rll}
\mathcal{L}\left[\!\!\left[\begin{array}{l} \texttt{if}\ e_1 \\ \texttt{then}\ e_2 \\ \texttt{else}\ e_3 \end{array} \right]\!\!\right] n & = \texttt{combinePA}\ e'_1\ e'_2\ e'_3 & \\
 & \text{where}\ e'_1 = \mathcal{L}[\![e_1]\!]\ n & \\
 & \quad e'_2 = \texttt{case}\ ys_2\ \texttt{of}\ \texttt{ATup}_k\ n_2\ y_1 \ldots y_k \to \mathcal{L}'[\![e_2]\!]\ n_2 & \\
 & \quad e'_3 = \texttt{case}\ ys_3\ \texttt{of}\ \texttt{ATup}_k\ n_3\ y_1 \ldots y_k \to \mathcal{L}'[\![e_3]\!]\ n_3 & \\
 & \quad (ys_2, ys_3) = \texttt{splitPA}\ e'_1\ (\texttt{ATup}_k\ n\ y_1 \ldots y_k) & \\
 & \quad \{y_1, \ldots, y_k\} = \text{free variables of } e_2, e_3 & \\
\mathcal{L}'[\![e]\!]\ n & = \texttt{if}\ n\texttt{==0}\ \texttt{then emptyPA else}\ \mathcal{L}[\![e]\!]\ n &
\end{array}
$$

Figure 6: The vectorisation transformation

```
($:L) :: PA (a:->b) -> PA a -> PA b
($:L) (AClo env fs fl) = fl env
```

The transformation rules in Figure 6 are given with their type invariants, which make the rules much more comprehensible. For example, consider the rule for $\mathcal{V}[\![e_1\ e_2]\!]$. Since $e_1 : \tau_1 \text{->} \tau_2$, we know that $\mathcal{V}[\![e_1]\!] : \mathcal{V}_t[\![\tau_1]\!] \text{:->} \mathcal{V}_t[\![\tau_2]\!]$; that is why we need the application function (`$:`) to transform the application to an expression of type $\mathcal{V}_t[\![\tau_2]\!]$.

Similarly, the rule for $\mathcal{V}[\![\lambda x.e]\!]$ must produce a value of type $\mathcal{V}_t[\![\tau_1]\!] \text{:->} \mathcal{V}_t[\![\tau_2]\!]$, and that in turn must be built with a `Clo` constructor. We build an environment tuple $(y_1, \ldots, y_k)$, of the free variables of $(\lambda x.e)$. Now the type of the arguments of `Clo` tell us what functions we must build. The scalar function simply requires a recursive use of $\mathcal{V}[\![e]\!]$, while the lifted function requires us to generate a lifted version of the code for e, $\mathcal{L}[\![e]\!]\ n$.

The rules for $\mathcal{L}[\![e]\!]\ n$ can be read in the same way. The main new complication is with conditionals. First we compute in parallel e'_1, the vector of booleans (of length n) for the discriminant of the conditional. Then we use that vector to split the a vector of environment tuples into two parts, ys_2 (for which corresponding elements of e'_1 is true), and ys_3 (for which e'_1 is false). The lengths n_2, n_3 of these vectors will sum to n. Then we compute each of e'_2 and e'_3 in parallel, and finally interleave them together with `combinePA`.

Why do we need to pack and split the free variables in the conditional rule? Each free variable y_i is bound to an n-vector; but in the `then` branch we need a (shorter) n_2-vector (namely ys_2) of the elements of y_i for which e is True; and dually for the `else` branch. We must also test for n_2 or n_3 being zero (done by $\mathcal{L}'[\![e]\!]\ n$), otherwise when transforming a recursive function we would generate a program that recurses infinitely deep. The *operational* behaviour of the translated function will compute e'_1, e'_2 and e'_3 *in sequence*; as in any data-parallel machine, the "then" and "else" branches of a conditional are computed separately.

Figure 6 is the core of this paper. Our real system handles `let` expressions, `case` expressions, and constructors, and hence is a bit more complicated. But Figure 6 describes all the essential ideas.

Since $\mathcal{V}$ invokes both $\mathcal{V}$ and $\mathcal{L}$ (as does $\mathcal{L}$) you might worry about a code explosion. But notice that the $\texttt{clo}_s$ field in $\mathcal{V}$ is identical to the $\texttt{aclo}_s$ field in $\mathcal{L}$, and both are closed functions that can be named, and bound at top level; and similarly for the $\texttt{clo}_l$ and $\texttt{aclo}_l$ fields. Hence, as we will see in the examples that follow, we can avoid the code explosion simply by naming and sharing these functions.

6.1 A simple example

Here is the simplest possible example:

```
inc :: Float -> Float
inc = λx. x + 1
```

The full vectorisation transformation in Figure 6 gives us this:

```
incV :: Float :-> Float
incV = Clo () incS incL

incS :: () -> Float -> Float
incS = λe x. case e of () -> (+)V $: x $: 1
```

```
incL :: PA () -> PA Float -> PA Float
incL = λe x. case e of ATup0 n -> (+)V $:L x $:L (replicatePA n 1)
```

To aid explanation we have named $\texttt{inc}_S$ and $\texttt{inc}_L$, but otherwise we have simply applied Figure 6 blindly. Notice the way we have systematically transformed `inc`'s type, replacing `(->)` by `(:->)`. Notice too that this transformation neatly embodies the idea that we need two versions of every top-level function `inc`, a *scalar version* $\texttt{inc}_S$ and a *lifted version* $\texttt{inc}_L$. These two versions paired together to form the *fully vectorised version* $\texttt{inc}_V$.

The vectorised code makes use of vectorised addition `(+)`, which is part of a fixed, hand-written library of vectorised primitives:

```
(+)V :: Float :-> Float :-> Float
(+)V = Clo () (+)S (+)L

(+)S :: () -> Float -> Float :-> Float
(+)S = λe x. Clo x addFloatS addFloatL

(+)L :: PA () -> PA Float -> PA (Float:->Float)
(+)L = λe xs. AClo xs addFloatS addFloatL

-- Implemented in the back end
addFloatS :: Float -> Float -> Float
addFloatL :: PA Float -> PA Float -> PA Float
```

The intermediate functions $\texttt{(+)}_S$ and $\texttt{(+)}_L$ deal with partial applications of `(+)`. Finally we reach ground truth: invocations of $\texttt{addFloat}_S$ and $\texttt{addFloat}_L$, which are implemented by the back end. The former is the ordinary floating point addition instruction; the latter is a "vector instruction", which will be implemented differently on different targets. On a sequential machine it will be implemented as a loop; on a GPU it will be implemented using vector hardware; on a cluster it will be implemented using a loop on each CPU with barrier synchronisation at the end. Section 7 elaborates.

These functions look grotesquely inefficient, especially considering how trivial the original function `inc` was. Fortunately, most of the clutter is introduced to account for the *possibility* of higher order programming, and can be removed by straightforward optimisations.

For example, consider the sub-term $\texttt{(+)}_V$ `$: x $: 1` in the definition of $\texttt{inc}_S$. We can simplify it in the following way:

```
  (+)V $: x $: 1)
⟹ Inline (+)V
  (Clo () (+)S (+)L) $: x $: 1
⟹ Definition of $:
  (+)S () x $: 1
⟹ Inline (+)S
  (Clo x addFloatS addFloatL) $: 1
⟹ Definition of $:
  addFloatS x 1
```

All the intermediate closure data structures are removed. (To save generating huge intermediates during compilation, we are exploring whether the vectorisation transformation could have special cases to avoid introducing them in the first place.)

6.2 The higher order example again

It is instructive to see a case where the use of higher order functions prevents complete removal of intermediate closures. Let us return to the `ho` example of Section 5.1:

```
ho :: (Int->Bool) -> (Bool, [:Bool:])
ho f = (f 2, mapP f [:1,2,3:])
```

Again applying the vectorisation transformation blindly we get this:

```
hoV :: (Int :-> Bool) :-> (Bool, PA Bool)
hoV = Clo () hoS hoL

hoS :: () -> (Int :-> Bool) -> (Bool, PA Bool)
hoS () f = (f $: 2, mapPV $: f $: [:1,2,3:])

hoL :: PA () -> PA (Int :-> Bool) -> PA (Bool, PA Bool)
hoL (ATup_0 n) fs
  = (,)L (fs $:L replicatePA n 2)
         (replicatePA n mapPV $:L fs $:L replicatePA n [:1,2,3:])
```

We have taken a short-cut here by using optimised transformation rules for pairs:

$$\begin{array}{rcl} \mathcal{V}[\![(e_1,e_2)]\!] & = & (\mathcal{V}[\![e_1]\!], \mathcal{V}[\![e_2]\!]) \\ \mathcal{L}[\![(e_1,e_2)]\!]\ n & = & (,)_L\ n\ (\mathcal{L}[\![e_1]\!]\ n)\ (\mathcal{L}[\![e_2]\!]\ n) \end{array}$$

The reader may verify the correctness of this optimised rule by seeing what happens instead if we use the normal translation $(,)_V$ `$:` $\mathcal{V}[\![e_1]\!]$ `$:` $\mathcal{V}[\![e_1]\!]$, and the definition of $(,)_V$, which in turn is very like that for `(+)`. Because of our array representation, the lifted pairing function $(,)_L$ is a constant-time operation:

```
(,)L :: Int -> PA a -> PA b -> PA (a,b)
(,)L n xs ys = ATup2 n xs ys
```

6.3 How flattening happens

In our informal overview (Section 3.2) we said that we "replace a call (`mapP f`) by f_L". Higher order flattening takes that *static* decision and makes it *dynamic*, by representing `f` by a pair of functions, thereby allowing `mapP` to select at runtime. (With the usual compile-time optimisations when `f` is known, of course.) The code for `mapP` itself is therefore the heart of the way in which nested data parallelism is transformed to flat data parallelism. Here it is:

```
mapPV :: (a :-> b) :-> PA a :-> PA b
mapPV = Clo () mapP1 mapP2

mapP1 :: () -> (a :-> b) -> PA a :-> PA b
mapP1 _ f = Clo f mapPS mapPL
```

```
mapP2 :: PA () -> PA (a :-> b) -> PA (PA a :-> PA b)
mapP2 _ fs = AClo fs mapPS mapPL

mapPS :: (a :-> b) -> PA a -> PA b
mapPS (Clo env fs fl) xss
  = fl (replicatePA (lengthPA xss) env) xss

mapPL :: PA (a :-> b) -> PA (PA a) -> PA (PA b)
mapPL (AClo env _ fl) xss
  = unconcatPA xss (fl (expandPA xss env) (concatPA xss))
      -- xss :: PA (PA a)
      -- env :: PA e
      -- fl  :: PA e -> PA a -> PA b
```

In `mapPS` we exploit the key observation from Section 3.2, namely that we can define the doubly-lifted function using the singly-lifted one `fl`, using constant-time reshaping operations on the data. Unfortunately, to account for free variables, we face a small complication: the environment `env` contains one element for each *subarray* of `xss`. Thus, before applying `fl` we must expand `env`, i.e., repeat each element as many times as the corresponding subarray of `xss` has elements. For top-level functions, the environment will be empty and `expandPA` performs no work.

Of course, `mapP` is not the only function that the library must implement. All of (the `PA` versions of) the functions in Figure 1 must be provided in vectorised form. For example, here is the lifted version of `zipP` (the definition of `zipPA` is given in Section 4.1):

```
zipPL :: PA (PA a) -> PA (PA b) -> PA (PA a,b)
zipPL (AArr segd xs) (AArr _ ys) = AArr segd (zipPA xs ys)
```

These library functions are the heart of flattening: they make nested data parallelism "go". Everything is organised to make their implementation, especially their lifted variants, work efficiently.

6.4 User-defined data types

One of Haskell's strengths is the ease with which programmers can declare new algebraic data types, and process them using pattern matching. DPH allows all of this expressiveness in fully-vectorised code as well. There are two main complications: occurrences of `(->)` and `[::]` in user-defined data types; and representing arrays of values drawn from such types. We discuss each in turn.

Vectorising user-defined data types

In Figure 6, the type transform $\mathcal{V}_t[\![\tau]\!]$ replaces a user-defined data type T by its vectorised counterpart T_V. But what exactly is T_V? Consider

```
data Fun = MkFun (Int -> Int)
```

Remember that in the vectorised program, each function arrow `(->)` must be replaced by a function closure `(:->)` — and of course that must also happen inside data types. So we must generate a vectorised version of `Fun`, thus:

```
data Fun_V = MkFun_V (Int :-> Int)
```

This must be done recursively: if a constructor of data type `T` mentions `Fun`, then `T` too must have a vectorised version. So the vectorised variant of each data type obtained by simply applying the $\mathcal{V}_t[\![]\!]$ transform to every type in the data type declaration. The `Tree` type of Section 2.2 is another good example, because we must replace `[::]` by `PA`:

```
data Tree_V = Node_V Mass Location (PA Tree)
```

While we *can* generate a vectorised version of every data type, it is *unnecessary* to do so for data types that do not mention functions or parallel arrays. Happily, almost all data types fall into this category; for example `Bool`, `Maybe`, lists, tuples, and so on. We quietly took advantage of this in the `Tree` example, by not transforming `Mass` to $\texttt{Mass}_V$ (and similarly `Location`) because $\texttt{Mass} = \texttt{Mass}_V$. In Section 6.5 we will see a second reason to avoid vectorising a data type unless it is absolutely necessary to do so.

Arrays of user-defined data types

The ideas of Section 4.1 can readily be extended to work for arbitrary user-defined algebraic data types. We have already seen how this works for `Tree` in Section 4.3. Here is another example, a sum type:

```
data Maybe a = Nothing | Just a
```

How can we represent an array of `Maybe Float` values? The natural dense representation is as a pair of (a) an array of booleans (`True` for `Nothing`, and `False` for `Just`), and (b) an array of `Float` containing only the `Just` values:

```
instance PAElem a => PAElem (Maybe a) where
  data PA Maybe a = AMaybe (PA Bool) (PA a)
  indexPA (AMaybe bs vs) i
    | indexPA bs i = Nothing
    | otherwise    = indexPA vs (indexPA just_indices i)
    where
      just_indices = scanPA (+) 0 (mapPA boolToInt bs)
  lengthPA (AMaybe bs _) = lengthPA bs
```

In practice, to avoid computing `just_indices` on each indexing operation we precompute the index vector, and cache it in an extra field of the `AMaybe` constructor.

In our real implementation, we avoid generating a big instance declaration for every such user-defined data type, by instead generating code to convert it to a simple sum-of-products representation, and then using a set of fixed instances for `PAElem` at those representation types.

6.5 What we have swept under the carpet

Vectorisation is a complicated transformation, and to keep it comprehensible we have simplified several aspects. In this section we briefly mention some of them.

Types and dictionaries

The alert and Haskell-savvy reader will have noticed the following discontinuity in our presentation. We described `PA` type in association with a *type class*, `PAElem`. However, type classes are dealt with by the type inference system, right at the front end of the compiler, and are completely translated out in the passage to the Core intermediate language. In this desugaring, a function with an overloaded type, such as `nub :: Eq a => [a] -> [a]` is given a second parameter which is a record, or "dictionary", of the functions that implement the operations of the `Eq` class.

In the desugared program, `nub` has type `EqD a -> [a] -> [a]`, where `EqD` is an ordinary data type, thus:

```
data EqD a = EqD { (==) :: a -> a -> Bool
                 , (/=) :: a -> a -> Bool }
```

Correspondingly, the desugarer injects an extra argument at every call to `nub`, namely the correct method suite for that particular call site.

The vectoriser generates many calls to `replicatePA`, `splitPA`, etc, which have type-class-constrained types, *yet the vectoriser runs after typechecking and desugaring are complete*. So the vectoriser cannot take advantage of the implicit injection of extra arguments; instead it must insert them itself. In the real implementation of Figure 6, the vectoriser therefore adds appropriate dictionary abstractions and applications. (In fact, since GHC's Core language is an explicitly-typed variant of System F, we also inject type abstractions and applications.) All this is tiresome but routine; showing the implicit abstractions and applications in Figure 6 would have dramatically obfuscated an already-dense figure.

Selective vectorisation

As mentioned earlier, we do not really vectorise the *whole* program; rather, we selectively vectorise parts of it. We must also generate marshaling code to allow us to "cross the border" between vectorised and unvectorised code. For example, in Barnes-Hut, we presumably want to vectorise the `oneStep` function, which will give us

```
oneStepV :: PA Particle :-> PA Particle
```

If we want to be able to call `oneStep` from ordinary scalar code, we must generate the following marshalling code:

```
oneStep :: [:Particle:] -> [:Particle:]
oneStep ps = fromPA (oneStepV $: (toPA ps))

toPA   :: PAElem a => [:a:] -> PA a
fromPA :: PAElem a => PA a -> [:a:]
```

Marshaling may also be necessary for user-defined data types.. For example, suppose we vectorise a function `f :: Int -> Fun`, so that `fV :: Int :-> FunV` (cf. Section 6.4 for the definition of `Fun`). If we want to call `f` from normal scalar code, we must generate:

```
f :: Int -> Fun
f n = case fV n of
        MkFunV tf -> MkFun (($:) tf)
```

Of course, it gets worse if the data type is recursive, because the marshaling code has to traverse the whole structure. On the other hand, no marshaling is needed for types that have have no functions or arrays inside them, which is a strong reason for exploiting that special case (Section 6.4).

Marshaling has a run-time cost. In particular, the calls to `toPA` and `fromPA` change the data representation for parallel arrays, and so are potentially *very* expensive. In fact, it is possible to choose a representation for `[:a:]` that mitigates these costs somewhat but in general, marshaling data across the border should be avoided.

The question of just which parts of the program to vectorise is therefore an interesting one. We want to vectorise code that can run in parallel; we want to reduce marshaling to a minimum; and we do not want to vectorise code where there is little or no benefit. We suggest automatic approaches in [CLJK08], but it may also be reasonable to seek help from the programmer (e.g. "vectorise module X but not module Y").

Laziness

Consider this function:

```
f :: Int -> Int
f x = h x (1/x)
```

Although `x` might be zero, let us assume that `h` only evaluates its second argument if its first argument is non-zero. Haskell's lazy evaluation therefore ensures that no divide-by-zero exception is raised.

The lifted version will look something like this:

```
f_L :: PA Int -> PA Int -> PA Int
f_L xs = h_L xs (replicatePA (lengthPA xs) 1 /_L xs)
```

The trouble is that a demand for *any* element of h_L's second argument will force *all* the elements to be evaluated, including the divisions by zero. Something very similar arises in a more local context when we have `let` expressions:

```
f x = let y = 1/x in
      if x==0 then 0 else y+1
```

Although this is something of a corner case, we do not yet have a very satisfying solution. We currently simply ignore the problem, and accept the slight change in semantics. A better solution might be to reify the exception into an exceptional value (like a IEEE NaN); but that carries an efficiency cost. Lastly, we might treat the argument as a nullary function, accepting the loss of sharing that would result.

7 Multicore execution model

The vectorisation transformation turns all nested data parallelism into parallel operations on flat arrays, as used by the instances of the `PAElem` class. The transformation is crucial to express parallel algorithms on a high-level of abstraction and in a modular fashion. However, purely functional array operations, even if restricted to flat arrays, are still a far cry from the hardware model of multicore CPUs.

In particular, the vectorised code uses many superfluous intermediate arrays, which increase the overhead of memory management and whose creation involves extra synchronisation between parallel CPUs. Even worse, the repeated traversal of large structures compromises locality of reference, and so, has a very negative effect on execution performance. Finally, we need to map the data parallel array operations onto the multi-threaded execution model of multicore CPUs by way of the Single Program Multiple Data (SPMD) model [Dar01].

In contrast to the vectorisation transformation, we can implement the mapping from flat data-parallel code to SPMD code using existing transformation and optimisation phases of GHC; in particular, we make heavy use of GHC's inliner and rewrite rules [PM02, PTH01], which enable library-specific optimisations as part of library source code, in the form of compiler pragmas. Consequently, we can implement these transformations without altering GHC's source code, which greatly simplifies experimentation with different transformations.

In the reminder of this section, we illustrate the transformation of flat data-parallel code into SPMD code by way of an example. Further details are in [KC99, CK03, CSL07, CLS07, CLP+07].

7.1 Running example

As an example, we consider the computation of the value `far_forces`, in the function `calcForces` of Section 2.2, by way of the array comprehension,

```
[: forceOn p m l | p <- ps, isFar len l p :]
```

After vectorisation and simplification to remove intermediate closure data structures, we have

```
forceOn'L (filterPS (isFar len l) ps) m l
```

The code performs two collective operations on the input array `ps` in sequence. Firstly, the application of $\texttt{filterP}_S$ to remove all particles that are not far, and secondly, a computation that corresponds to lifting `forceOn` only on its first argument (here called, $\texttt{forceOn'}_L$):

```
forceOn'L :: PA Particle -> Mass -> Location -> PA Force
```

Such pipelines of collective operations are typical for data-parallel code.

7.2 The SPMD execution model

The implementation of collective array operations, such as `mapP` and `filterP`, needs to distribute the workload evenly across the the available processing elements (PEs), such as multiple cores and CPUs. In the data-parallel model, the workload of a PE is dependent on the number of array elements residing on that PE. Hence, we balance work by suitably distributing the array elements. By default, we choose an even distribution; i.e., given p PEs and an array of length n, each PE gets about n/p array elements.

In the SPMD model, the individual PEs process local array elements until they arrive at a point in the computation where they require non-local data, and need to cooperate with other PEs. In our example, the result of $\texttt{filterP}_S$ is such a point. Even if the *input* to $\texttt{filterP}_S$ is an array that is evenly spread across the PEs, the *output* of $\texttt{filterP}_S$ might

be wildly unbalanced, depending on which elements of the array are selected by the predicate. If so, any further processing of that array would have an equally unbalanced work distribution.

To avoid a work imbalance, arrays need to be re-distributed when their size changes. Redistribution is a cooperative process in which all PEs need to coordinate. However, redistribution is not the only such operation in an SPMD implementation of data parallelism. Other prominent cooperative operations are reductions (such as `foldP`), pre-scans (such as `scanlP`), and permutation operations. Overall, a parallel program executing in SPMD-style alternates between *processing phases*, where the PEs operate independently on local data, and *communication phases*, where the processing elements interact and exchange data.

Communication phases are typically expensive because they include data exchange and blocking to allow any slower PEs to catch up. Hence, compiler optimisations that remove communication phases in favour of longer-running processing phases are often worthwhile. In particular, the redistribution of arrays after operations that change the array length, such as $\texttt{filterP}_S$, does not necessarily improve overall runtime. An inexpensive, purely local operation may be faster, even if work is not ideally balanced, than an expensive redistribution followed by the same local operation with a perfectly balanced workload.

7.3 Gang parallelism

Our implementation of the SPMD model for data parallelism is based on the coordinated execution of a *gang of threads*, with one thread per PE. GHC includes a Haskell library for concurrent programming with explicit thread forking and thread communication primitives. It forms the lowest level of abstraction in our data-parallel array framework and enables us to implement the entire library in Haskell without any special compiler support or the need to resort to C code.

We need to make the distributed nature of computations in the SPMD model explicit to further compile the code resulting from vectorisation, such as

```
forceOn'L (filterPS (isFar len l) ps) m l
```

In this context, distribution does not imply that the data is necessarily located on physically distinct memory banks, but that different threads are responsible for the processing of different portions of parallel arrays. By being explicit about distribution, we are automatically also explicit about the distinction of processing phases versus communication phases.

Our main vehicle for distinguishing between these two phases and making distribution explicit is the type `Dist a` of *distributed values.* For instance, `Dist Int`, pronounced "distributed `Int`", denotes a collection of *local* integers, such that there is one local integer value per gang thread. Arrays can be distributed, too: `Dist [:Float:]` is a collection of local array *chunks*, again one per gang member, which together make up the array. Arrays are distributed across gang members and joined back together by the following functions:

```
splitD :: PA a -> Dist (PA a)
joinD  :: Dist (PA a) -> PA a
```

Distributed values support a number of operations, most importantly mapping:

```
mapD :: (a -> b) -> Dist a -> Dist b
```

While `splitD` and `joinD` denotes communication, `mapD` is the main means of implementing parallel processing phases: the gang members concurrently apply the (purely sequential) function to their respective local values.

7.4 Inlining of gang code

Given a sequential filter function operating on a *single chunk* of a parallel array

```
filterS :: (a -> Bool) -> PA a -> PA b
```

we can define $\texttt{filterP}_\texttt{S}$ as a distributed gang computation as follows:

```
filterPS p arr = joinD (mapD (filterS p) (splitD arr))
```

Given a global parallel array, which is not distributed, `splitD` distributes the array across the gang, `mapD (filterS p)` applies the sequential filter function in parallel to all chunks of the distributed array, and finally, `joinD` combines the various chunks, which may now be of varying length, into one global array.

Similarly, $\texttt{forceOn'}_\texttt{L}$ internally consists of `mapD`s that compute the force for each particle. The force computations for the individual particles are entirely independent, so we can assume $\texttt{forceOn'}_\texttt{L}$ to have the following structure:

```
forceOn'L ps m l
  = joinD (mapD (mapS (λp. forceOn p m l)) (splitD ps))
```

where `forceOn` is the original, sequential function from the source of our Barnes-Hut implementation and `mapS` is a purely sequential array mapping function.

GHC's inliner will inline the definition of both $\texttt{filterP}_\texttt{S}$ and $\texttt{forceOn'}_\texttt{L}$; i.e., it will perform the following rewriting:

```
forceOn'L (filterPS (isFar len l) ps) m l
```

$\Longrightarrow_{\text{Inlining}}$

```
joinD (mapD (mapS (λp. forceOn p m l)) (
  splitD (joinD (mapD (filterS (isFar len l)) (splitD ps)))))
```

Of special interest here is the function `splitD` which is applied to the immediate result of `joinD` (in the second line of the resulting expression). This turns a distributed array into a global array and distributes it again. In contrast to the original array, the newly distributed one is guaranteed to be distributed evenly; hence, a `splitD/joinD` combination performs load balancing.

However, as we remarked earlier, it is often an advantage to accept some load imbalance in favour of avoiding communication phases in an SPMD computation. In GHC, we easily achieve that by specifying the following *rewrite rule*:

```
"splitD/joinD" forall xs. splitD (joinD xs) = xs
```

GHC has support for specifying such rewrite rules directly in the library source code as compiler pragmas [PTH01]. Applications of the `splitD/joinD` rule frequently produce two adjacent applications of `mapD`, which signal two adjacent purely sequential and thread-local computations. We can combine them, and hence eliminate a synchronisation point, using the well known map fusion law:

```
"mapD/mapD" forall f g xs.
  mapD f (mapD g xs) = mapD (f . g) xs
```

Applying both rules to our example, we get

```
joinD (mapD (mapS (λp. forceOn p m l)) (
  splitD (joinD (mapD (filterS (isFar len l)) (splitD ps)))))
```
$\Longrightarrow$ Apply `splitD/joinD`
```
joinD (mapD (mapS (λp. forceOn p m l)) (
  mapD (filterS (isFar len l)) (splitD ps)))
```
$\Longrightarrow$ Apply `mapD/mapD`
```
joinD (mapD (mapS (λp. forceOn p m l) . filterS (isFar len l))
            (splitD ps)))
```

At this point, the question arises whether we can combine adjacent sequential array combinators, such as `mapS` and `filterS`, to reduce the number of array traversals and intermediate data structures. Indeed, we aggressively remove such inefficiencies using a fusion framework known as *stream fusion* [CSL07, CLS07, CLP+07], but we will refrain from discussing this in detail.

This concludes our brief overview of the post-vectorisation aspects of Data Parallel Haskell. A somewhat more detailed discussion can be found in [CLP+07].

8 Related work

We discussed prior work on the implementation of language support for nested data parallelism in detail in [CLP+07]. In this paper, we will only give a brief overview of existing work, and how they compare to our approach.

The starting point for our work was the nested data parallel programming model of NESL [Ble90, BCH+94], which we extended and implemented in the context of a general-purpose language and GHC, a state-of-the-art compiler. Consequently, we have to deal with a multitude of issues not previously addressed, as for example the combination of user-defined and parallel data structures, selective vectorisation, higher-order functions, separate compilation, and aggressive cross-function optimisation.

Prins et al. worked on various aspects of the vectorisation of nested data parallel programs; see, e.g., [PP93, PPW95]. Most of their work was also in the context of a functional language, but one that like NESL lacks many of Haskell's features. Their work is largely orthogonal to ours.

The Proteus system [MNPR94] promised a combination of data and control parallelism, but Proteus had a particular focus on manual refinement of algorithms, where data parallel components were automatically vectorised, this again was a complete whole-program transformation. Moreover, the system was never fully implemented.

Manticore [FFR+07] supports a range of forms of parallelism including nested data parallelism. Manticore employs some of the same techniques that we use, but does not implement flattening yet [FRRS08]. According to the project web page, a preliminary implementation of the Manticore system should be available around the time when this paper is published.

So et al. [SGW06] developed a parallel library of immutable arrays for C/C++ supporting what they call *sub-primitive fusion*. Their choice of immutable arrays, despite working with imperative languages, is to enable aggressive program transformations, much like in

our approach. However, where we apply transformations statically during compile time, their library builds a representation of the to be executed computation at runtime. Consequently, they require less compiler support and do not have to worry about inlining and similar optimisations. However, they incur a runtime penalty by performing optimisations at runtime and need to amortise that penalty by further optimisations. Like us, they also strive for a seamless integration of data parallelism and explicit concurrency within a single program.

9 Conclusion

We are excited about Data Parallel Haskell because it gives us some chance of writing parallel programs that can in principle efficiently exploit very large parallel machines working on large data sets.

In this paper we have outlined solutions to the challenges of polymorphism, higher order functions, and user-defined data types. There is much to do, however, before we can declare victory. The very generality of Data Parallel Haskell makes it an ambitious undertaking. Many components have to work together smoothly to generate efficient code — and that is before we start to consider matters such as using SSE vector instructions or GPUs, or mapping to a distributed memory architecture. Nevertheless, we regard nested data parallelism general, and Data Parallel Haskell in particular, as a very promising and exciting approach to harnessing the multicores.

Acknowledgements

We gratefully acknowledge the help of Max Bolingbroke, Ryan Ingram and John Reppy, whose comments led to real improvements in the paper. Thank you.

References

[AJ89] A. W. Appel and T. Jim. Continuation-passing, closure-passing style. In *POPL '89: Proceedings of the 16th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 293–302, New York, NY, USA, 1989. ACM Press.

[BCH+94] Guy E. Blelloch, Siddhartha Chatterjee, Jonathan C. Hardwick, Jay Sipelstein, and Marco Zagha. Implementation of a portable nested data-parallel language. *Journal of Parallel and Distributed Computing*, 21(1):4–14, April 1994.

[BH86] J. Barnes and P. Hut. A hierarchical $O(n \log n)$ force calculation algorithm. *Nature*, 324, December 1986.

[Ble90] Guy E. Blelloch. *Vector Models for Data-Parallel Computing*. The MIT Press, 1990.

[Ble96] Guy E. Blelloch. Programming parallel algorithms. *Communications of the ACM*, 39(3):85–97, 1996.

[BS90] Guy E. Blelloch and Gary W. Sabot. Compiling collection-oriented languages onto massively parallel computers. *Journal of Parallel and Distributed Computing*, 8:119–134, 1990.

[CK00] Manuel M. T. Chakravarty and Gabriele Keller. More types for nested data parallel programming. In Philip Wadler, editor, *Proceedings of the Fifth ACM*

SIGPLAN International Conference on Functional Programming (ICFP'00), pages 94–105. ACM Press, 2000.

[CK03] Manuel M. T. Chakravarty and Gabriele Keller. An approach to fast arrays in haskell. In Johan Jeuring and Simon Peyton Jones, editors, *Lecture notes for The Summer School and Workshop on Advanced Functional Programming 2002*, number 2638 in Lecture Notes in Computer Science, 2003.

[CKPM05] Manuel Chakravarty, Gabriele Keller, Simon Peyton Jones, and Simon Marlow. Associated types with class. In *ACM Symposium on Principles of Programming Languages (POPL'05)*. ACM Press, 2005.

[CLJK08] Manuel MT Chakravarty, Roman Leshchinskiy, Simon Peyton Jones, and Gabriele Keller. Partial vectorisation of Haskell programs. In *Proc ACM Workshop on Declarative Aspects of Multicore Programming*, San Francisco, January 2008. ACM Press.

[CLP+07] Manuel M. T. Chakravarty, Roman Leshchinskiy, Simon Peyton Jones, Gabriele Keller, and Simon Marlow. Data Parallel Haskell: a status report. In *DAMP 2007: Workshop on Declarative Aspects of Multicore Programming*. ACM Press, 2007.

[CLS07] Duncan Coutts, Roman Leshchinskiy, and Don Stewart. Stream fusion: From lists to streams to nothing at all. In *Proceedings of the ACM SIGPLAN International Conference on Functional Programming, ICFP 2007*, April 2007.

[CSL07] Duncan Coutts, Don Stewart, and Roman Leshchinskiy. Rewriting haskell strings. In *Practical Aspects of Declarative Languages 8th International Symposium, PADL 2007*, pages 50–64. Springer-Verlag, January 2007.

[Dar01] Frederica Darema. The spmd model: Past, present and future. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, volume 2131 of *Lecture Notes in Computer Science*. Springer-Verlag, 2001.

[DG04] Jeffrey Dean and Sanjay Ghemawat. MapReduce: simplified data processing on large clusters. In *Sixth Symposium on Operating System Design and Implementation (OSDI'04)*, San Francisco, December 2004.

[FFR+07] M. Fluet, N. Ford, M. Rainey, J. Reppy, A. Shaw, and Y. Xiao. Status report: The manticore project. In *2007 ACM SIGPLAN Workshop on ML*. ACM Press, 2007.

[For97] High Performance Fortran Forum. High performance fortran language specification version 2.0. Technical report, Rice University, 1997.

[FRRS08] M. Fluet, M. Rainey, J. Reppy, and A. Shaw. Implicitly-threaded parallelism in manticore. In *Proceedings of the 13th ACM SIGPLAN International Conference on Functional Programming (ICFP'08)*. ACM Press, 2008.

[GHLL+98] William Gropp, Steven Huss-Lederman, Andrew Lumsdaine, Ewing Lusk, Bill Nitzberg, William Saphir, and Marc Snir. *MPI: The Complete Reference (Vol. 2)*. The MIT Press, 1998.

[JW07] Simon Peyton Jones and Philip Wadler. Comprehensive comprehensions: comprehensions with order by and group by. In *Haskell Workshop 2007*, pages 61–72, Frieburg, Germany, September 2007.

[KC98] Gabriele Keller and Manuel M. T. Chakravarty. Flattening trees. In David Pritchard and Jeff Reeve, editors, *Euro-Par'98, Parallel Processing*, number 1470

in Lecture Notes in Computer Science, pages 709–719, Berlin, 1998. Springer-Verlag.

[KC99] Gabriele Keller and Manuel M. T. Chakravarty. On the distributed implementation of aggregate data structures by program transformation. In José Rolim et al., editors, *Parallel and Distributed Processing, Fourth International Workshop on High-Level Parallel Programming Models and Supportive Environments (HIPS'99)*, number 1586 in LNCS, pages 108–122, Berlin, Germany, 1999. Springer-Verlag.

[LCK06] Roman Leshchinskiy, Manuel M. T. Chakravarty, and Gabriele Keller. Higher order flattening. In *Third International Workshop on Practical Aspects of High-level Parallel Programming (PAPP 2006)*, number 3992 in LNCS. Springer-Verlag, 2006.

[MNPR94] P. Mills, L. Nyland, J. Prins, and J. Reif. Software issues in high-performance computing and a framework for the development of hpc applications. In *Computer Science Agendas for High Perfromance Computing*. ACM Press, 1994.

[NVI07] NVIDIA. *NVIDIA CUDA Compute Unified Device Architecture, Programming Guide, Version 1.1*, 2007. `http://developer.download.nvidia.com/compute/cuda/1_1/NVIDIA_CUDA_Programming_Guide_1.1.pdf`.

[PCS99] Jan F. Prins, S. Chatterjee, and M. Simons. Irregular computations in Fortran — expression and implementation strategies. *Scientific Programming*, 7:313–326, 1999.

[Pey96] SL Peyton Jones. Compilation by transformation: a report from the trenches. In *European Symposium on Programming*, volume 1058 of *LNCS*, pages 18–44. Springer Verlag, 1996.

[PL91] SL Peyton Jones and J Launchbury. Unboxed values as first class citizens. In RJM Hughes, editor, *ACM Conference on Functional Programming and Computer Architecture (FPCA'91)*, volume 523 of *Lecture Notes in Computer Science*, pages 636–666, Boston, 1991. Springer.

[PM02] SL Peyton Jones and S Marlow. Secrets of the Glasgow Haskell Compiler inliner. *Journal of Functional Programming*, 12:393–434, 2002. First published at Workshop on Implementing Declarative Languages, Paris, Sept 1999.

[PP93] Jan Prins and Daniel Palmer. Transforming high-level data-parallel programs into vector operations. In *Proceedings of the Fourth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 119–128, San Diego, CA., May 19-22, 1993. ACM Press.

[PPW95] Daniel Palmer, Jan Prins, and Stephan Westfold. Work-efficient nested data-parallelism. In *Proceedings of the Fifth Symposium on the Frontiers of Massively Parallel Processing (Frontiers 95)*. IEEE Press, 1995.

[PTH01] Simon L. Peyton Jones, Andrew Tolmach, and Tony Hoare. Playing by the rules: rewriting as a practical optimisation technique in GHC. In Ralf Hinze, editor, *2001 Haskell Workshop*. ACM SIGPLAN, September 2001.

[SCPD07] Martin Sulzmann, Manuel Chakravarty, Simon Peyton Jones, and Kevin Donnelly. System F with type equality coercions. In *ACM SIGPLAN International Workshop on Types in Language Design and Implementation (TLDI'07)*. ACM, 2007.

[SGW06] Byoungro So, Anwar Ghuloum, and Youfeng Wu. Optimizing data parallel

operations on many-core platforms. In *First Workshop on Software Tools for Multi-Core Systems (STMCS)*, 2006. `http://www.isi.edu/~kintali/stmcs06/prog.html`.

[SJ08] Satnam Singh and Simon Peyton Jones. *A Tutorial on Parallel and Concurrent Programming in Haskell*. Lecture Notes in Computer Science. Springer Verlag, Nijmegen, Holland, May 2008.

Foundations of Software Technology and Theoretical Computer Science (Bangalore) 2008.
Editors: R. Hariharan, M. Mukund, V. Vinay; pp 415-422

Knowledge Infusion: In Pursuit of Robustness in Artificial Intelligence*

Leslie G. Valiant
Harvard University
valiant@seas.harvard.edu

ABSTRACT. Endowing computers with the ability to apply commonsense knowledge with human-level performance is a primary challenge for computer science, comparable in importance to past great challenges in other fields of science such as the sequencing of the human genome. The right approach to this problem is still under debate. Here we shall discuss and attempt to justify one approach, that of *knowledge infusion*. This approach is based on the view that the fundamental objective that needs to be achieved is *robustness* in the following sense: a framework is needed in which a computer system can represent pieces of knowledge about the world, each piece having some uncertainty, and the interactions among the pieces having even more uncertainty, such that the system can nevertheless reason from these pieces so that the uncertainties in its conclusions are at least controlled. In knowledge infusion rules are learned from the world in a principled way so that subsequent reasoning using these rules will also be principled, and subject only to errors that can be bounded in terms of the inverse of the effort invested in the learning process.

1 Introduction

One of the most important challenges for computer science is that of understanding how systems that acquire and manipulate commonsense knowledge can be created. By commonsense knowledge we mean knowledge of the kind that humans can successfully manipulate but for which no systematic theory is known. For example, conducting appropriate everyday conversations among humans requires such commonsense knowledge, while the prediction of the trajectory of a projectile can be accomplished using the systematic theory offered by physics.

We argue that to face this challenge one first needs a framework in which inductive learning and logical reasoning can be both expressed and their different natures reconciled. The learning provides the necessary robustness to the uncertainties of the world. It enables a system to go to the world for as much data as needed to resolve uncertainties. The reasoning is needed to provide a principled basis for manipulating and reaching conclusions from the uncertain knowledge that has been learned. The process by which we can infuse a system with commonsense knowledge, in a form suitable for such reasoning, we call *knowledge infusion* or *KI* [13, 15].

Robust logic [14] is a concrete proposal for realizing KI. It offers a formalism for learning rules that are suitable for later chaining together for the purpose of reasoning. In this system both learning and reasoning can be performed in polynomial time, and, further, the reasoning has certain soundness and completeness properties, and the errors in learning

*This work was supported in part by grant NSF-CCF-04-27129.

FSTTCS 2008
IARCS Annual Conference on
Foundations of Software Technology and Theoretical Computer Science
http://drops.dagstuhl.de/opus/volltexte/2008/1770

and reasoning can be upper bounded in terms of the inverse of a polynomial function of the effort expended in the learning.

For brevity we shall refer to a system that can successfully reason with commonsense knowledge as an *intelligent system*. Recent headlines in the New York Times - with one word omitted in each case - included "Can Weeds Help Solve the ..... Crisis?" and "Oil Hits New High as Dow Flirts With ..... Territory." It would be reasonable to expect intelligent systems to be able to make reasonable guesses of the missing words. To achieve such capabilities there is a need both for access to commonsense knowledge, as well as for an ability to apply such knowledge to situations not previously experienced.

We suggest that for such a word completion, or any other, task to be a valid test for intelligent systems, it will need to have two properties in common with the Turing Test [11]. First, there should be no *a priori* restrictions, to any limited subdomain or microworld, on the domain of knowledge treated. Second, there needs to be some numerical evaluation of performance relative to some baseline. We regard these two properties as the most fundamental prerequisites for tests of progress in this area.

Recently we have reported on the results of experiments that test whether KI is effective for such an unrestricted word completion task [8]. These experiments, performed on a data set of a half a million natural language sentences, showed that this task of predicting a deleted word from a sentence could be performed to a higher accuracy by this method than by a baseline learning method that did not use reasoning. In this experiment the learned rules contained commonsense knowledge about the world, while the baseline method could be regarded as a more syntactic learning method, in the sense of n-gram methods in natural language processing but using more powerful Winnow based learning methods as developed by Roth and his coworkers [4]. The experiments highlight the technical challenges of learning from noisy data reliably enough that the learned rules could be chained together fruitfully. In particular there is a need for algorithms that have good run times and good generalization properties, and for methods of chaining rules that preserve the generalization guarantees.

Technical descriptions of the approach described can be found in references [8, 14, 15] and we shall not detail any of that here. In this note we shall attempt to summarize informally the general justification of our approach in comparison with some alternatives. Since the effort needed to endow computer systems with usable commonsense knowledge can be expected to be very considerable, it seems worthwhile to invest effort into evaluating carefully the various available approaches.

2 Achieving Robustness

As soon as the feasibility of large scale computations became evident in the middle of the twentieth century an immediate concern was whether the execution of millions of instructions, each one highly accurate in itself, would inevitably lead to errors accumulating and giving totally incorrect final answers. For processing commonsense knowledge this robustness problem would appear to be an especially important concern, since significant uncertainties may appear here even in individual steps. This paper is predicated on the proposition that any theory of commonsense reasoning that fails to include robustness in its subject

matter will also fail as a basis for intelligent systems as these scale up.

That *some* theoretical basis is required for intelligent systems to be successfully realized is widely acknowledged. The system is expected to make determinations for circumstances that may not be foreseen by the designer, and these determinations will need therefore to be derived using some principled basis. In this pursuit the most widely advocated theories have been the equivalents of the predicate calculus [6], on the one hand, and Bayesian reasoning [10], on the other, or some combination of these. These theories do have much to offer, being mathematically consistent theories that attempt to address directly the representation of knowledge. Their main drawback from our perspective is that they do not address directly the issue of robustness. Indeed, as a broad generalization, it has proved in practice that systems based on these theories are brittle, in the sense that, as the knowledge bases in such systems grow, the predictions made by them degrade significantly. This phenomenon is not difficult to explain. These theories guarantee accuracy of predictions only if the model created in terms of them is consistent and accurate. Such guarantees of accuracy and consistency are not available in areas, such as commonsense knowledge, which we define here to be just those for which no exact axiomatization is known.

We regard the Bayesian framework as an elaboration of the classical logical one. It is appropriate in cases where the knowledge being axiomatized contains probabilistic processes, and there is some hope of an axiomatization. Since it is just an elaboration of logic, and in that sense at least as difficult to apply to model complex knowledge, we do not regard it as helpful in cases where even the deterministic aspects of the knowledge being modeled is so ill understood that there has been no success in modeling even that part. Putting it another way, the Bayesian framework would be a panacea if the only obstacle to modeling commonsense knowledge wete that it was some probabilistic version of something that could be successfully modeled otherwise. However, we believe that the obstacles are of a different and more severe nature: The basic concepts in this knowledge, as represented typically by natural language words, do not generally have unambiguous meanings. They may number tens or hundreds of thousands, as they do in the experiments reported in [8]. Typically, an observed situation contains much incomplete information - the truth value of most concepts in any one situation is unstated and unknown. Finally, there is no reason to believe that an accurate model of the totality of the possible relationships among these multitudinous concepts exists.

While the predicate calculus and Bayesian reasoning may be useful intellectual aids in designing systems, by themselves they do not offer the guarantee of robustness that is needed: significant aspects of the world are difficult enough to describe accurately, and when conclusions are to be drawn from conjoining a series of these aspects then the errors are likely to grow out of control.

Our proposal is that the only guarantee of robustness that is viable for complex manipulations on uncertain unaxiomatized pieces of knowledge is that offered by learning processes that have access to instances of the world to which the knowledge refers. The knowledge in the system can then be constantly tested and updated against real world examples. The behavior of the system will then be guaranteed in a statistical sense to be correct with high probability on examples drawn from the same probability distribution from which the learning experience was drawn. Thus the semantics we advocate for systems that

manipulate commonsense knowledge is *PAC semantics* [12], which we shall discuss below. We shall require not just the learning aspects but also the *outcomes of the reasoning processes* to be predictably accurate in that sense. An argument for why an *a priori* guarantee of accuracy, as guaranteed by PAC semantics, is needed for all aspects of the system can be illustrated by distinguishing three situations:

In a first kind of situation, which we call (A), we have a candidate intelligent system at hand. To test whether its behavior is effective we can run it on live examples. We will for sure get a reliable statistical assessment of the system's accuracy on the distribution of examples on which it is tested.

In another situation, which we shall label (C), we do not have a candidate system at hand, but are asking whether one can be built at all, and wondering on what principles it might be built so as to be able to pass a live test as described above. The PAC-model of learning is designed exactly for this situation. It promises that a system, based on certain algorithms and trained on enough examples of a fixed but arbitrary function that is within the capabilities of the learning algorithm, will with high probability be able to pass the live test described in situation (A). The PAC model guarantees that the processes are *computationally feasible*, needing only a polynomial amount of computation and data. Equally importantly, the model acknowledges the possibility that errors will be made in the predictions, but these *errors will be controlled* in the sense that they can be made arbitrarily small by increasing, in a polynomially bounded manner, the amount of data and computation that is being invested. The PAC model, which captures and quantifies both the computational and statistical aspects of learning, is designed to capture exactly the desiderata of any system that draws its knowledge from, and needs to perform well in, a world that is too complex to be modeled exactly.

It is conceivable, of course, that systems based on principles, such as Bayesian inference or the predicate calculus, that do not guarantee robustness *a priori* in this way will by chance offer such robustness. This has not happened to date. We would argue that if such robustness is found then that too will be a phenomenon of PAC semantics, and therefore most fruitfully described in those terms. Whatever thought aids may have been used in the design, the only sense in which the result can be declared a success is in the PAC sense that the system is accurate in its ultimate task on natural examples, and requires only efficiently computable processes.

Returning to our enumeration of the different situations, we note that there is also an intermediate situation (B). There we have a candidate system at hand, as in (A), but instead of testing it against live data we are given a set of examples on which the system has performed well, with the promise that the examples were once chosen live from a distribution, but no promise that the system was designed independently of these examples. We can validate the system against the examples, as in (A), but we have reason to be suspicious that the system was tailor made to fit the data. However, ignoring the computational aspects of the PAC model and retaining only the statistical ones, we can obtain confidence in the system if the system is simple enough in terms of the amount of corroborating data, whether this simplicity is measured in terms of the number of bits [1] or the VC-dimension [2] of the system description. This situation (B) is also interesting because it, like situation (A), provides a principled reason for having confidence in a system even if the design methodology of the

system did not guarantee such confidence.

We conclude that what we need ideally is a design methodology that guarantees robustness in the PAC sense, as in situation (C). We may be lucky and derive systems with similar performance in the PAC sense, as verified in situations (A) or (B), without having used a methodology that is guided by a PAC guarantee. However, based on the past history of such attempts, we estimate the likelihood of this succeeding as being small.

We are not suggesting that heuristics, or algorithms whose success is not well understood, be avoided altogether. In robust logic we first learn rules that are accurate in the PAC sense, and then we chain these together in a way that gives predictions that are also accurate in the PAC sense if the learned rules were. It may be valid to use heuristics in each of the two halves if sight is not lost of the overall goal that the final predictions have to be accurate in the PAC sense. For example, the first half is a standard machine learning task. There is ample evidence for the existence of algorithms, such as various decision tree algorithms, that appear to be effective PAC learning algorithms for some useful set of functions and distributions that have yet to be characterized. There is no reason for not using these if these are shown to be effective in practice. What we are saying, however, is that if we do not *plan* for PAC accuracy at every stage, in the manner of robust logic, for example, then we are unlikely to get PAC accuracy in the final predictions.

3 Teaching Materials

The problem of creating systems that realize KI has two parts. The first is the design of the specific learning and reasoning algorithms that are to be used, as discussed for example in [14]. The second is the manner in which the real world knowledge is presented to the system. It may be possible to arrive at reasonable proposals for the former algorithmic questions once and for all. However, the second aspect, which we call the preparation of *teaching materials*, may be an endless task reflective of the endless effort humans put into the analogous process in the education of the young.

While we emphasize that the main characteristic of commonsense knowledge is that no axiomatization is known, we welcome the use of any attempted axiomatizations of parts of the knowledge. For example, when processing natural language texts dictionaries of synonyms and antonyms, as provided, for example, by WordNet [9], are extremely useful, and are used, in fact, in the experiments reported in [8]. Similarly, hand-designed ontologies of knowledge, as developed for example in [5], may have an important role in providing information that is difficult to acquire elsewhere. We shall regard such hand-designed attempted axiomatizations also as teaching materials. When these are used in a KI system they should be regarded as having PAC semantics also, and subject to modification in the light of experience. For example, if a dictionary contains some inconsistencies then this will be discovered in the course of applying this knowledge to examples. Of course, equally welcome as teaching materials to hand-crafted methods, are automatic methods of obtaining reliable knowledge, even when these are of restricted forms (e.g. [3]).

The teaching materials can be expected to have some hand-designed architecture. For example, the knowledge may be layered, so that the most fundamental knowledge is infused first, and subsequent layers that depend on that first layer are infused later. Of course,

the creation of teaching materials for even one layer may be expected to be challenging. Naturally occurring sources, such as books or the web, may omit essential knowledge that humans acquire by other means. We hope that progress in building useful systems will be made, nevertheless, once the problem of constructing teaching materials is raised to the status of a first-class intellectual activity.

A fundamental difficulty may arise in bootstrapping this process. For example, if the lowest layer of concepts on which the multi-layered learning is performed consists of visual primitives, which are at least partially available at birth in biological systems, or of knowledge of three dimensional space at a level not explicitly taught to children, then there remains the problem of providing these primitives to the system. It is conceivable that this can be done by programming. However, there remains the possibility that, just as with higher level concepts, the only practical way of putting these into a machine in a robust enough manner is by learning. Now evolution can also be regarded as a learning process, and recently a theory of evolvability has been formulated in the PAC framework [16]. Hence one can envisage constructing teaching materials for intelligent systems to correspond not only to knowledge learned by individual humans, but also to knowledge acquired by them from their ancestors through evolution. We believe that biology provides an existence proof that cognitive systems based on pure learning and appropriate teaching materials are feasible. It remains, however, a significant research endeavor to find pragmatic ways of constructing useful systems by means of these methods, with or without programmed components.

4 Further Issues

What we have attempted to argue here is that there is no hope of creating intelligent systems if one fails to incorporate mechanisms, in the manner of KI, that guarantee robustness of the decisions made by the system. Over the decades researchers have identified many other difficulties in the pursuit of intelligent systems. The question arises as to whether KI makes some of these difficulties even less tractable, or contributes to alleviating these.

The first general point we make is that, at least from a cognitive perspective, the PAC semantics of KI should be viewed as substantially assumption-free and not as imposing substantive constraints. The definition does presuppose that the function being learned is within the capabilities of the learning algorithm. However, as long as we are learning concepts that are learnable at all, for example by a biological system, then we have an existence proof that such a learning algorithm exists. We note that an actual system will attempt to learn many concepts simultaneously. It will succeed for those for which it has enough data, and that are simple enough when expressed in terms of the previously reliably learned concepts that they lie in the learnable class. The system can recognize which concepts it has learned reliably and which not, and will only use the former for reasoning. In this way a system will have a principled way of discovering which fragments of the knowledge offer useful predictive power, without having to embark on the hopeless task of modeling all of it.

Second, we argue that the statistical notion of correctness against a real world distribution of examples in the PAC sense is the best we can hope for. Of course, in many areas of science, particularly physics, strong predictive models of the world whether deterministic

or probabilistic do hold. This is because these models are based on an axiomatization of a restricted aspect of the world. Clearly, for any aspect of the world that can be similarly axiomatized (i.e. for which an accurate generative model can be designed) whether in terms of differential equations, mathematical logic, or explicit probabilistic models, such models can lead to predictions that work in all cases with quantifiable error and are superior. However, commonsense reasoning addresses areas where such axiomatizations and generative models have met with limited success. In particular systems based on them have not scaled. The considerable success of machine learning as compared with programmed systems, in speech recognition, computer vision and natural language processing, we interpret as deriving from the fact that the robustness that learning offers outweighs the possible benefits of partially correct axiomatizations. For the general commonsense reasoning problem we expect this tradeoff to tilt considerably further towards machine learning.

Finally, we ask whether PAC semantics offers solutions to the difficulties that have been identified for other approaches? This issue has been discussed in [13]. There it is argued that such issues as conflict resolution, context, incomplete information, and nonmonotonic phenomena, which are problematic to various degrees for classical logic, are not inherently problematic in PAC semantics. In fact, interesting new possibilities arise, for example, in the treatment of incomplete information [7].

5 Acknowledgement

I am grateful to Loizos Michael for his helpful comments on this paper.

References

[1] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth. Occam's razor. *Information Processing Letters*, 24 (1987) 377-380.

[2] A. Blumer, A. Ehrenfeucht, D. Haussler and M. Warmuth. Learnability and the Vapnik -Chervonenkis dimension. *Journal of the ACM*, 36 (1989) 929965.

[3] O. Etzioni, M. Cafarella, D. Downey, A. Popescu, T. Shaked, S. Soderland, D. S. Weld, A. Yates. Unsupervised named-entity extraction from the web: an experimental study *Artif. Intell.*, Vol. 165, No. 1. (June 2005), pp. 91-134.

[4] Y. Even-Zohar and D. Roth. A classification approach to wod prediction. In *Proc. First Annual Meeting of the North American Chapter of the Association for Computational Linguistics (NAACL'00)*, (2000) 124-131.

[5] D. B. Lenat. CYC: A large-scale investment in knowledge infrastructure, *Comm. Assoc. Computing Machinery* 38:11 (1995) 33-38.

[6] J. McCarthy. Programs with common sense. In *Proc. Teddington Conference on the mechanization of Thought Processes*, (1958), 75-79.

[7] L. Michael. Learning from partial observations. *IJCAI* (2007), 968-974.

[8] L. Michael and L. G. Valiant. A first experimental demonstration of massive knowledge infusion, *Proc. 11th International Conference on Principles of Knowledge Representation and Reasoning*, Sept. 16-20, 2008, Sydney, Australia, 378-389.

[9] G. A. Miller. WordNet: A lexical database for English. *Comm. Assoc. Computing Machinery*, 38:11(1995) 39-41.
[10] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, San Mateo, CA (1988).
[11] A. M. Turing. Computing machinery and intelligence. *Mind*, LIX (236):433-460, (1950).
[12] L. G. Valiant. A theory of the learnable. *Comm. Assoc. Computing Machinery* 27:11 (1984) 1134-1142.
[13] L. G. Valiant. A neuroidal architecture for cognitive computation, *J. Assoc. Computing Machinery*, 47:5 (2000) 854-882.
[14] L. G. Valiant. Robust logics, *Artificial Intelligence Journal*, 117 (2000) 231-253.
[15] L. G. Valiant. Knowledge infusion, *Proc. 21st National Conference on Artificial Intelligence, AAAI06*, Jul 16-20, 2006, Boston, MA, AAAI Press, 1546-1551.
[16] L. G. Valiant. Evolvability, *J. Assoc. Computing Machinery*, to appear. (Earlier version: *Proc. 32nd International Symposium on Mathematical Foundations of Computer Science*, Aug. 26-31, Český Krumlov, Czech Republic, *LNCS*, Vol 4708, (2007) Springer-Verlag, 22-43.)