Variants of Collapsible Pushdown Systems

Paweł Parys*

University of Warsaw, ul. Banacha 2, 02-097 Warszawa, Poland
parys@mimuw.edu.pl

Abstract

We analyze the relationship between three ways of generating trees using collapsible pushdown systems (CPS's): using deterministic CPS's, nondeterministic CPS's, and deterministic word-accepting CPS's. We prove that (for each level of the CPS and each input alphabet) the three classes of trees are equal. The nontrivial translations increase $n-1$ times exponentially the size of the level-n CPS. The same results stay true if we restrict ourselves to higher-order pushdown systems without collapse. As a second contribution we prove that the hierarchy of word languages recognized by nondeterministic CPS's is infinite. This is a consequence of a lemma which bounds the length of the shortest accepting run. It also implies that the hierarchy of ε-closures of configuration graphs is infinite (which was already known). As a side effect we obtain a new algorithm for the reachability problem for CPS's; it has the same complexity as previously known algorithms.

1998 ACM Subject Classification F.1.1 Models of Computation

Keywords and phrases collapsible pushdown systems, determinization, infinite hierarchy, shrinking lemma, reachability

Digital Object Identifier 10.4230/LIPIcs.CSL.2012.500

1 Introduction

Already in the 70's, Maslov ([14, 15]) generalized the concept of pushdown automata to higher-order pushdown automata and studied such devices as acceptors of string languages. In the last decade, renewed interest in these automata has arisen. They are now studied also as generators of graphs and trees. Knapik et al. [12] showed that the class of trees generated by deterministic level-n pushdown systems coincides with the class of trees generated by *safe* level-n recursion schemes,[1] and Caucal [8] gave another characterization: trees on level $n+1$ are obtained from trees on level n by an MSO-interpretation of a graph, followed by application of unfolding. Carayol and Wöhrle [7] studied the ε-closures of configuration graphs of level-n pushdown systems and proved that these graphs are exactly the graphs in the n-th level of the Caucal hierarchy. Driven by the question whether safety implies a semantical restriction to recursion schemes (which was recently proven [17]), Hague et al. [10] extended the model of level-n pushdown systems to level-n collapsible pushdown systems (n-CPS's) by introducing a new stack operation called collapse. They showed that the trees generated by such systems coincide exactly with the class of trees generated by all higher-order recursion schemes and this correspondence is level-by-level.

We compare three ways of generating trees using CPS's of some level n. We consider here edge-labelled, unranked, and unordered trees. In the classical definition of a tree-generating CPS we take a deterministic CPS. Moreover it is typically assumed that every nonempty

* The author is partially supported by the Polish Ministry of Science grant nr N N206 567840.
[1] Safety is a syntactic restriction on the recursion scheme.

Computer Science Logic 2012 (CSL'12).
Editors: Patrick Cégielski, Arnaud Durand; pp. 500–515

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

run (from some reachable configuration) using only ε-transitions can be extended into a run reading also some letter.[2] We call such CPS strongly deterministic (see Definition 2.11). To generate a tree, we unfold the configuration graph of the strongly deterministic CPS $\mathcal{S}$ into a tree, and we contract all ε-labelled edges. We can also say that we take the ε-closure of the configuration graph, and then we unfold it into a tree. Denote this tree $T(\mathcal{S})$, and the class of all such trees as $Trees^n_D$. Notice that in such a tree each node has, for each letter, at most one outgoing edge labelled by that letter. Trees having this property will be called deterministic.

But we can do the same (i.e. take the ε-closure of the configuration graph, and then unfold it into a tree) for nondeterministic CPS's. In general such trees can be very strange, e.g. can contain a node which has infinitely many successors. We restrict ourselves only to CPS's generating deterministic trees. This class of trees will be called $Trees^n_N$. Notice that whether a tree is deterministic is only a property of the tree, not the system generating it; of course nondeterministic systems can also generate deterministic trees (the configuration graphs can contain big parts consisting of only ε-transitions, containing many loops and nondeterministic choices, but they are whole contracted to one node in the ε-closure). Nevertheless, we prove that every such tree can be also generated by a strongly deterministic system of the same level. In other words, there is an easy syntactic condition saying whether a tree generated by a CPS can be also generated by a strongly deterministic CPS.

▶ **Theorem 1.1.** *Let $\mathcal{S}$ be an n-CPS, such that $T(\mathcal{S})$ is deterministic. Then there exists a strongly deterministic n-CPS $\mathcal{S}'$ such that $T(\mathcal{S}) = T(\mathcal{S}')$. System $\mathcal{S}'$ has size $n-1$ times exponential in the size of $\mathcal{S}$, and can be computed in such time.*

Determinization results are always important, because deterministic automata are, colloquially speaking, simpler and have better properties. For example a strongly deterministic system can be easily simulated: having a configuration one can just run the system to see which letters can be read next. For general (nondeterministic) systems this is much more difficult, as by principle there can be arbitrarily long runs reading just one letter, so we do not know when to stop. Other reason is that it is easier to prove that a tree is not generated by a deterministic system, than that a graph is not generated by any (nondeterministic) system. To be more precise, consider our result [11] showing that the hierarchy of graphs generated by CPS's is strict, i.e. that for each n there is a graph generated by an $(n+1)$-CPS (i.e. which is the ε-closure of its configuration graph) which is not generated by any n-CPS. We perform there a pumping construction which says that in a long enough run there is a fragment which (in some sense) can be repeated arbitrarily many times, and this leads to a contradiction. The problem is when this fragment does not read any letter; then repeating it does not change the path in the ε-closure. To avoid such situation a very sophisticated analysis is performed, which causes a technical complication of the proof. However using the result of this paper we could just show that the unfolding of this graph is not generated by a deterministic system of level n, which would significantly simplify the proof. For deterministic systems it is enough to perform the pumping in a simple form: repeating arbitrarily many times a fragment not reading a letter is impossible in a deterministic system, so automatically by repeating some fragment of a run we obtain longer and longer paths in the ε-closure.

[2] Sometimes this assumption is dropped (and then in the generated tree all branches of ε-labelled edges which do not lead to a non-ε edge are replaced by a $\bot$-labelled edge). Because our definition is more restrictive, our results become stronger.

We also consider CPS's as word acceptors (such CPS's are given together with a set of final states). Let us recall that no determinization is possible for word languages: it is known that nondeterministic systems recognize more languages than deterministic ones (even if we are allowed to increase the level). However we show that every deterministic system can be converted into a strongly deterministic one, in which additionally from each reachable configuration there exists an accepting run.

▶ **Theorem 1.2.** *Let $\mathcal{S}$ be a deterministic n-CPS with final states F, used as word acceptor. Then there exists a strongly deterministic n-CPS $\mathcal{S}'$ with final states F' such that it recognizes the same language as $(\mathcal{S}, F)$, and from every reachable configuration of $\mathcal{S}'$ there is an accepting run (unless the language is empty). System $\mathcal{S}'$ has size $n-1$ times exponential in the size of $\mathcal{S}$, and can be computed in such time.*

This can be equivalently stated for trees. Let $Pref(\mathcal{S}, F)$ be the tree whose nodes are all prefixes of words accepted by the CPS $\mathcal{S}$ with final states F, and let $Trees^n_L$ be the class of all such trees. If from every reachable configuration of $\mathcal{S}$ there is an accepting run, we have $Pref(\mathcal{S}, F) = T(\mathcal{S})$. Thus thanks to the above theorem we have $Trees^n_L \subseteq Trees^n_D$. For the opposite implication it is enough to assume that every state is final (then every path of $T(\mathcal{S})$ is accepted). Thus we obtain equality of the three classes of trees.

▶ **Corollary 1.3.** *For each $n \in \mathbb{N}$ we have $Trees^n_D = Trees^n_N = Trees^n_L$.*

All our results still hold if we restrict ourselves to systems without collapse (i.e. then the resulting systems also do not use collapse). Let us notice that Theorem 1.2 can be also easily deduced (but probably without the bound on the size of the new automaton) from a theorem [4] saying that collapsible pushdown systems are closed under logical reflection. We however believe that our proof is simpler (and it gives also Theorem 1.1 which cannot be obtained using logical reflection).

As a second contribution we prove that the hierarchy of word languages recognized by (nondeterministic) CPS's is infinite.

▶ **Theorem 1.4.** *For each $n \in \mathbb{N}$ there is a language recognized by a pushdown system (without collapse) of level $2n+1$ which is not recognized by any n-CPS.*

Our example language from Theorem 1.4 can be also used to show that the hierarchies of graphs (i.e. ε-closures of configuration graphs) and of trees are infinite. However we already know [11] that these two hierarchies are not only infinite but in fact strict, i.e. that for each n there is a tree generated by a level-$(n+1)$ pushdown system without collapse which is not generated by any n-CPS, and similarly for ε-closures of configuration graphs. The strictness of the hierarchy of word languages recognized by higher-order pushdown systems without collapse follows from the Damm's paper [9]. Similar results for the hierarchy of word languages recognized by CPS's were missing so far (this is stated as an open problem in [5]).

The strictness results for tree and graph hierarchies are obtained using a pumping lemma. This lemma essentially says that if an automaton has a very long run, then it also has arbitrarily long runs. Such arbitrarily long runs can be then transformed into arbitrarily long paths in the trees or graphs. However for the word languages hierarchy such lemma is useless: the problem is that in a nondeterministic system such longer and longer runs can all read the same word. We instead need a shrinking lemma, which would allow to shorten a run. We prove the following lemma, which is good for this purpose.

▶ **Theorem 1.5.** *Let $\mathcal{S}$ be an n-CPS with set of states Q and stack alphabet Γ, given together with a set of accepting states. Assume that there exists an accepting run of $\mathcal{S}$. Set $\exp_0(i) = i$ and $\exp_{k+1}(i) = 2^{\exp_k(i)}$. Then there exists an accepting run of $\mathcal{S}$ of length at most $\exp_{2n-1}(8|Q|^2|\Gamma|)$.*

As a third contribution we obtain two new algorithms which check whether the language recognized by a CPS is nonempty (one can equivalently talk about reachability of a configuration having the state in a given set). One algorithm is extremely simple: it is enough to check if there exists a run from the initial configuration to a configuration with accepting state, whose length is bounded by the threshold from Theorem 1.5. Its complexity is however $(2n-1)$-NEXPTIME. In Section 4 we present another algorithm for the emptiness problem, whose complexity is $(n-1)$-EXPTIME.

The author is unaware of any result dealing with the emptiness problem directly. Of course the emptiness problem is a special case of the μ-calculus model-checking problem. A direct solution of this problem uses parity games over configuration graphs of the systems [10]. For such games one can (in a nontrivial way) reduce the level of the system by one, increasing its size exponentially. This approach gives n-EXPTIME complexity for μ-calculus model checking (and in fact the problem is n-EXPTIME complete); however it is not difficult to see that if we just consider emptiness, the complexity can be lowered to $(n-1)$-EXPTIME (as the first reduction of level gives then only a polynomial blowup). There are several other approaches to μ-calculus model checking [16, 13, 18], which use equivalent characterizations of collapsible pushdown systems by recursion schemes or Krivine machines. In contrast, our algorithm analyzes directly possible behaviors of the system, works only for the emptiness problem, and is simple.

Let us remark that an algorithm for the emptiness problem allows us also to check whether a given word is accepted by a system, also in $(n-1)$-EXPTIME: it is enough to take the product of the given system with a finite automaton accepting only the one given word.

Organization The strategy for proving our results is as follows. In Section 2 we give all necessary definitions. We begin the proof by Section 3, where we show how Theorem 1.4 follows from Theorem 1.5. Then in Section 4 we define so-called types of stacks, which talk about possible kinds of runs starting in a given configuration. We also present there (in Subsection 4.1) the algorithm for the emptiness problem. Next, in Section 5 we say that the pushdown systems can itself calculate the type of its current configuration. This already gives us Theorem 1.2. Finally in Section 6 we say that the systems can not only calculate the types, but also use them to choose some unique run (from some class of runs). This gives us the determinization described by Theorem 1.1. As a side effect, in Subsection 6.1, we obtain a bound on the length of runs described by types, which gives us Theorem 1.5.

2 Preliminaries

Collapsible pushdown systems of level n (in the following n is always assumed to be a fixed natural number) are an extension of pushdown systems where we replace the stack by an n-fold nested stack structure. For the manipulation of the higher-order stack, we have a push and a pop operation for each stack level $1 \leq i \leq n$ and a collapse operation. When a new symbol is pushed onto the stack, we attach a copy of the stack to this symbol and the collapse operation may replace the current stack with the stack attached to the topmost

symbol again, i.e. in some sense the collapse operation allows to jump back to the stack where the current topmost symbol was created for the first time.

▶ **Definition 2.1.** Given a number n (the level of the system) and stack alphabet Γ, we define the set of stacks as the smallest set satisfying the following.

- If $s_1, s_2, \ldots, s_m$ are $(k-1)$-stacks, where $1 \leq k \leq n$, then the sequence $[s_1, s_2, \ldots, s_m]$ is a k-stack. This includes the empty sequence ($m = 0$).
- If s^k is a k-stack, where $1 \leq k \leq n$, and $\alpha \in \Gamma$, then (α, k, s^k) is a 0-stack.

For a k-stack s^k and a $(k-1)$-stack s^{k-1} we write $s^k : s^{k-1}$ to denote the k-stack obtained by appending s^{k-1} on the end of s^k. We write $s^2 : s^1 : s^0$ for $s^2 : (s^1 : s^0)$.

Let us remark that in the classical definition the stacks are defined differently: they are not nested, the 0-stack does not store the linked k-stack, just stores a natural number. While performing the collapse operation, this number is used to find the stack pointed by the link. Note that this is only a syntactical difference. Let us emphasize however that this modification is essential to obtain a correct definition of "types" below: Our k-stack already contains all stacks in the links, so looking at a k-stack we have the complete information about it, and we can summarize it using a type from a finite set. On the other hand the original k-stack has arbitrarily many numbers pointing to some stacks "outside", and for each of this numbers it is important how the target of the pointer looks like; thus already the interface with the external world is arbitrarily big.

▶ **Definition 2.2.** We define stack operations as follows. We decompose a stack s of level n into its topmost stacks $s^n : s^{n-1} : \cdots : s^0$. We have $\mathsf{pop}^i(s) := s^n : \cdots : s^{i+1} : s^i$, where $1 \leq i \leq n$; the result is undefined if s^i is empty. For $2 \leq i \leq n$ we have $\mathsf{push}^i(s) := s^n : \cdots : s^{i+1} : (s^i : \cdots : s^0) : s^{i-1} : \cdots : s^0$. The level-1 push is $\mathsf{push}^1_{\alpha,k}$ for $\alpha \in \Gamma$, $1 \leq k \leq n$ which is defined by $\mathsf{push}^1_{\alpha,k}(s) := s^n : \cdots : s^2 : (s^1 : s^0) : (\alpha, k, s^k)$. The collapse operation col^i (where $1 \leq i \leq n$) is defined if the topmost 0-stack is (a, i, t^i), and t^i is not empty. Then it is $\mathsf{col}^i(s) := s^n : \cdots : s^{i+1} : t^i$. Otherwise the collapse operation is undefined.

Pushdown systems only use n-stacks that can be created from the initial n-stack using the stack operations. We call those n-stacks *pushdown stores (pds)*.

▶ **Definition 2.3.** The *initial n-stack* $\bot_n$ is the n-stack which contains only one 0-stack, which is $(\bot, 1, s^1)$, where $\bot \in \Gamma$ is a special symbol, and s^1 is the empty 1-stack.[3] Some n-stack s is a *pushdown store* (or *pds*), if there is a finite sequence of stack operations that creates s from $\bot_n$.

▶ **Definition 2.4.** A *collapsible pushdown system of level n* (an n-CPS) is a tuple $\mathcal{S} = (\Gamma, A, Q, q_I, \bot, \Delta)$, where Γ is a finite stack alphabet containing the initial stack symbol $\bot$, A is a finite input alphabet, Q is a finite set of states, $q_I \in Q$ is an initial state, and $\Delta \subseteq Q \times \Gamma \times (A \cup \{\varepsilon\}) \times Q \times OP^n_\Gamma$ is a transition relation where OP^n_Γ denotes the set of stack operations. When saying that a transition goes from $(q, \alpha) \in Q \times \Gamma$ we mean that it has q and α on the first two coordinates. A *configuration* is a pair (q, s) for a state q and an pds s. The *initial configuration* is $(q_I, \bot_n)$.

Below our convention is that, whenever we have a system $\mathcal{S}$, we assume that Q is its set of states, A its input alphabet, and Γ its stack alphabet. The *size* of an n-CPS is the number of its transitions. Let us emphasize that, according to our definition, a configuration of a pushdown system contains a pds, not an arbitrary n-stack.

[3] The choice of 1 is arbitrary, it can be any number from 1 to n.

▶ **Definition 2.5.** A *run* R of length m, from c_0 to c_m, is a sequence $c_0 \vdash^{a_1} c_1 \vdash^{a_2} \cdots \vdash^{a_m} c_m$ where $c_i = (q_i, s_i)$ are configurations and $a_i \in A \cup \{\varepsilon\}$ are such that, for $1 \leq i \leq m$, there exists a transition $(q_{i-1}, \alpha_{i-1}, a_i, q_i, op)$ such that the topmost stack symbol of s_{i-1} is α_{i-1} and $s_i = op(s_{i-1})$.

▶ **Definition 2.6.** A *labelled transition system* (LTS) over alphabet A is an edge-labelled directed graph with a distinguished initial node. More formally, it is $(G, init, (E_a)_{a\in A})$, where G is a set of nodes, $init \in G$, and $E_a \subseteq G \times G$ for each $a \in A$. A *configuration graph* of a CPS $\mathcal{S}$ is an LTS over alphabet $A \cup \{\varepsilon\}$, which contains all reachable configurations of $\mathcal{S}$ as nodes, and where $(c, d) \in E_a$ when there is a run $c \vdash^a d$ of length 1.

By a *tree* we always mean an LTS which is a tree, i.e. in which every node is reachable from the initial node by exactly one path.

▶ **Definition 2.7.** The *ε-closure* of an LTS $\mathcal{G} = (G, init, (E_a)_{a\in A\cup\{\varepsilon\}})$ over alphabet $A\cup\{\varepsilon\}$ is the LTS $(G', init, (E'_a)_{a\in A})$ where

$$G' := \{init\} \cup \{d \in G : \exists c \in G\ (c, d) \in E_a \text{ for some } a \in A\}$$

and two nodes c, d are connected by E'_a if there is a path in $\mathcal{G}$ from c to d whose last edge is labelled by a, and all earlier edges are labelled by ε. The *unfolding* of an LTS $\mathcal{G} = (G, init, (E_a)_{a\in A})$ is a tree $(G', init, (E'_a)_{a\in A})$ where G' contains all paths of $\mathcal{G}$ starting in its initial node, and paths $p_1 = c_1c_2 \ldots c_k$ and $p_2 = c_1c_2 \ldots c_kc_{k+1}$ for $(c_k, c_{k+1}) \in E_a$ are connected by E'_a.

We denote the unfolding of the ε-closure of the configuration graph of an CPS $\mathcal{S}$ by $T(\mathcal{S})$.

▶ **Definition 2.8.** The word read by a run $c_0 \vdash^{a_1} c_1 \vdash^{a_2} \cdots \vdash^{a_m} c_m$ is obtained from $a_1a_2 \ldots a_m$ by dropping all appearances of ε. We say that a word w is accepted by a CPS $\mathcal{S}$ given together with a set of final states $F \subseteq Q$, if there exists a run of $\mathcal{S}$ reading w from the initial configuration to a configuration whose state is final. The language recognized by $(\mathcal{S}, F)$, denoted $L(\mathcal{S}, F)$, is the set of all words accepted by $(\mathcal{S}, F)$. The prefix tree of $L(\mathcal{S}, F)$, denoted $Pref(\mathcal{S}, F)$, is $(G, \varepsilon, (E_a)_{a\in A})$ where G contains all prefixes of words from $L(\mathcal{S}, F)$, and words w and wa (for $a \in A$) are connected by E_a.

▶ **Definition 2.9.** A system $\mathcal{S} = (\Gamma, A, Q, q_I, \bot, \Delta)$ is called *deterministic* if Δ is a partial function $\Delta\colon Q \times \Gamma \times (A \cup \{\varepsilon\}) \to Q \times OP^n_\Gamma$, and additionally from each $(q, \alpha) \in Q \times \Gamma$ we either have only an ε-transition, or only letter-labelled transitions.

▶ **Definition 2.10.** Let $(G, init, (E_a)_{a\in A})$ be an LTS. We say that it is *deterministic* if for each node c and each $a \in A$ there is at most one d such that $(c, d) \in E_a$.

Notice that if a system $\mathcal{S}$ is deterministic, then also the tree $T(\mathcal{S})$ is deterministic; however the opposite implication does not hold.

▶ **Definition 2.11.** We say that a deterministic CPS $\mathcal{S}$ is *strongly deterministic* if for some set of *dead states* $Q_{die} \subseteq Q$ it holds

- in each reachable configuration with state q and topmost stack symbol α each transition from (q, α) can be applied (i.e. we do not try to perform pop or col when it is impossible), and
- there are no transitions from (q, α) for $q \in Q_{die}$ and any α, and

- if c is a reachable configuration whose state is not a dead state, there is a run from c having some non-ε-transitions.[4]

3 The hierarchy is infinite

In this section we prove that the hierarchy of word languages is infinite (Theorem 1.4) basing on Theorem 1.5. We keep the notation $\exp_k$ from the statement of Theorem 1.5. For each $n \in \mathbb{N}$ consider the language

$$L_n = \{1^k 0^{\exp_n(k)} : k \in \mathbb{N}\}.$$

It is known that L_n can be recognized by a level-$(n+1)$ pushdown system (without collapse); see e.g. [2], example on pages 6-7, where a very similar pushdown system is constructed. Thus L_{2n} can be recognized by a level-$(2n+1)$ pushdown system.

Assume now that there exists an n-CPS $\mathcal{S}$, which recognizes L_{2n}. Let Q be its set of states, and Γ its stack alphabet. Choose k such that $\exp_{2n}(k) > \exp_{2n-1}(8(k+1)^2|Q|^2|\Gamma|)$. We construct a system $\mathcal{R}$ which accepts only one word $w = 1^k 0^{\exp_{2n}(k)}$, i.e. the only word from L_{2n} which has k letters 1. Such system can have the same stack alphabet as $\mathcal{S}$, and $(k+1)|Q|$ states. Indeed, we make $k+1$ copies of the set of states of $\mathcal{S}$. System $\mathcal{R}$ works like $\mathcal{S}$, but whenever it reads the 1 letter, it passes to the next copy of the set of states. If it is in the last copy, no more 1 letters can be read. And only the states from the last copy (those which were accepting in $\mathcal{S}$) are accepting. This way $\mathcal{R}$ accepts all words from the language of $\mathcal{S}$ which contain k letters 1, which is what we want.

By Theorem 1.5 $\mathcal{R}$ has an accepting run of length at most $\exp_{2n-1}(8(k+1)^2|Q|^2|\Gamma|)$. However this run reads $|w|$ letters, so it has length at least $|w| \geq \exp_{2n}(k)$. This is a contradiction, as $\exp_{2n}(k) > \exp_{2n-1}(8(k+1)^2|Q|^2|\Gamma|)$. This finishes the proof of Theorem 1.4.

As a side remark we observe that the same proof works for the hierarchy of ε-closures of configuration graphs. Let G_n be the graph whose nodes are

$$\{1^k 0^m : m \leq \exp_n(k),\ k \in \mathbb{N}\} \cup \{1^k 0^{\exp_n(k)} \sharp : k \in \mathbb{N}\},$$

and a node w is connected with node wa by an edge labelled a (where $a \in \{0, 1, \sharp\}$). We know that graph G_{2n} is the ε-closure of the configuration graph of a level-$(2n+1)$ pushdown system (similar to the one recognizing L_{2n}). On the other hand, G_{2n} cannot be the ε-closure of the configuration graph of a level-n pushdown system. If such system exists, it can be easily transformed into a word-accepting n-CPS recognizing L_{2n}; it is enough to stop in an accepting state instead of reading $\sharp$. The same can be done for trees.

4 Types of stacks

For the rest of this section we fix some n-CPS $\mathcal{S}$. The aim of this section is to assign to any k-stack s^k a type $\mathsf{type}(s^k)$ that determines existence of some runs starting in a stack with the topmost k-stack s^k. In order to obtain Theorem 1.1 we are interested in runs in which every transition except the last is labelled by ε, and the last is labelled by a letter from A. Additionally we want to know whether it can be further extended to read some next letter

[4] Thus there are no infinite runs reading no letters, and only configurations just after reading a letter can have no successors; whether this is the case is determined by the state.

(as otherwise we have to hold the system immediately to ensure strong determinism). For this reason we fix a morphism $\varphi\colon (A\cup\{\varepsilon\})^* \to M$ into a finite monoid M, which[5] applied to a word w tells us for each $a \in A$ whether $w \in \varepsilon^* a$ and whether $w \in \varepsilon^* a \varepsilon^* A$. The morphism φ will be applied to the word created from labels of a run, i.e. to the word read by a run, but including all epsilons; for simplicity of notation we just say that we apply φ to a run.

The idea of defining types is present also in [11] (and in [17] for automata without collapse). The novelty is that we bound here (see Subsection 6.1) the length of runs implied by the type (the previous proof instead of giving such bound was using a nontrivial induction). Moreover we define the types more carefully than in [11], so that their number is exponentially smaller (and hence our algorithm for the emptiness problem is $(n-1)$-EXPTIME, not n-EXPTIME). On the other hand the types in [11] are much more general, as we characterize here only accepting runs, while there much more sophisticated kinds of runs were characterized; however the new proof can be easily generalized to those other kinds of runs.

The type of s^k will be a set of *run descriptors* which come from a set $\mathcal{T}^k$ that will be defined inductively from $k=n$ to $k=0$. A typical element of $\mathcal{T}^k$ has the form

$$\sigma = (q, \Psi^n, \Psi^{n-1}, \dots, \Psi^{k+1}, m, q')$$

where $q, q' \in Q$ are states of $\mathcal{S}$, Ψ^i are some types of i-stacks, and $m \in M$. Let us explain the intended meaning of such a tuple. We want to have $\sigma \in \mathsf{type}(s^k)$ if and only if for all stacks $t^n, t^{n-1}, \dots, t^{k+1}$ where $\Psi^i \subseteq \mathsf{type}(t^i)$ there is a run evaluating to m under φ, from configuration $(q, t^n : t^{n-1} : \dots : t^{k+1} : s^k)$ to a configuration with state q'. In other words, if we put σ into $\mathsf{type}(s^k)$ we make the following claim. If for each $k+1 \le i \le n$ we take an i-stack t^i that satisfies the claims made by Ψ^i, then we will find a run evaluating to m that starts in state q and the stack obtained by putting s^k on top of the sequence of $t^n : \dots : t^{k+1}$, and ends in state q'. Let us mention that in order to obtain Theorem 1.1 we need not to keep the state q'.

We first introduce the set $\mathcal{T}^k$ of possible run descriptors of level k (the possible types of k-stacks are elements of $\mathcal{P}(\mathcal{T}^k)$). We write $\mathcal{P}(X)$ for the power set of X, and $\mathcal{P}_{\le 1}(X)$ for $\{Y \in \mathcal{P}(X) : |Y| \le 1\}$.

▶ **Definition 4.1.** We define $\mathcal{T}^n = Q \times M \times Q$, and inductively for $0 \le k \le n-1$:

$$\mathcal{T}^k = Q \times \mathcal{P}_{\le 1}(\mathcal{T}^n) \times \big(\mathcal{P}(\mathcal{T}^{n-1}) \times \mathcal{P}(\mathcal{T}^{n-2}) \times \dots \times \mathcal{P}(\mathcal{T}^{k+1})\big) \times M \times Q.$$

Notice that the n-th level is treated differently: we take $\mathcal{P}_{\le 1}(\mathcal{T}^n)$ instead of $\mathcal{P}(\mathcal{T}^n)$. This is possible because an n-stack can be used only once (cannot be copied), so we need just one run descriptor. The purpose for restricting to $\mathcal{P}_{\le 1}(\mathcal{T}^n)$ is to decrease exponentially the number of all run descriptors; for all other reasons a definition with $\mathcal{P}(\mathcal{T}^n)$ would be also good.

Next, we state the intended meaning of run descriptors and types.

▶ **Lemma 4.2.** *Let $0 \le l \le n$, and let $c = (q, s^n : s^{n-1} : \dots : s^l)$ be a configuration. The following two conditions are equivalent:*

1. *there exists a run from c which evaluates to m under φ and ends in state q',*
2. *$\mathsf{type}(s^l)$ contains a run descriptor $(q, \Psi^n, \Psi^{n-1}, \dots, \Psi^{l+1}, m, q')$ such that $\Psi^i \subseteq \mathsf{type}(s^i)$ for $l+1 \le i \le n$.*

[5] The results about types hold for any morphism; this morphism is just what we need for Theorem 1.1.

Let us remark that in Section 6 we strengthen the "2⇒1" implication of the above lemma: we not only say that the run exists, but we present a deterministic CPS which reconstructs such run (arbitrarily choosing one of them), and we also bound the length of this run.

Now we come to the definition of types. We first define how types can be composed. The intention of the next definition is that when Ψ^k is the type of a k-stack s^k, and Ψ^{k-1} is the type of a $(k-1)$-stack s^{k-1}, then $\Psi^k : \Psi^{k-1}$ is the type of $s^k : s^{k-1}$.

▶ **Definition 4.3.** Let $1 \leq k \leq n$, let Ψ^k be a subset of $\mathcal{T}^k$, and Ψ^{k-1} a subset of $\mathcal{T}^{k-1}$. Their *composition*, $\Psi^k : \Psi^{k-1}$, is a subset of $\mathcal{T}^k$ containing all run descriptors $(q, \Sigma^n, \Sigma^{n-1}, \ldots, \Sigma^{k+1}, m, q')$ such that in Ψ^{k-1} there is a run descriptor $(q, \Sigma^n, \Sigma^{n-1}, \ldots, \Sigma^{k+1}, \Sigma^k, m, q')$ for which $\Sigma^k \subseteq \Psi^k$.

Like for stacks, we write $\Psi^2 : \Psi^1 : \Psi^0$ for $\Psi^2 : (\Psi^1 : \Psi^0)$. Notice that the composition of types is monotone: if $\Psi^k \subseteq \Phi^k$ and $\Psi^{k-1} \subseteq \Phi^{k-1}$, then also $\Psi^k : \Psi^{k-1} \subseteq \Phi^k : \Phi^{k-1}$.

Next, we are going to define a function cons. In fact, cons is defined as a fixpoint of a sequence $(\mathsf{cons}_z)_{z\in\mathbb{N}}$. For each $z \in \mathbb{N}$, each stack symbol $\alpha \in \Gamma$, each number $1 \leq K \leq n$, and each $\Sigma^K \subseteq \mathcal{T}^K$ we define a set $\mathsf{cons}_z(\alpha, K, \Sigma^K) \subseteq \mathcal{T}^0$. The intention is that if a run descriptor $(q, \Psi^n, \Psi^{n-1}, \ldots, \Psi^1, m, q')$ is in the set $\mathsf{cons}_z(\alpha, K, \Sigma^K)$, then there exists a run evaluating to m, ending in state q, and starting in a configuration $(q, s^n : s^{n-1} : \cdots : s^0)$ such that Ψ^i is contained in the type of s^i and s^0 has symbol α and carries a link of level K to a stack of type Σ^K. When we enter the fixpoint $\mathsf{cons}(a, K, \Sigma^K)$ we will be able to replace the "if-then" by an "if and only if". For the "and only if" part, we need to know the complete type; when we reach the fixpoint cons of the functions cons_z, it will compute the complete type.

▶ **Definition 4.4.** Let $z \in \mathbb{N}$, let $\alpha \in \Gamma$, let $1 \leq K \leq n$, and let $\Sigma^K \subseteq \mathcal{T}^K$. For $z = 0$ we define $\mathsf{cons}_z(\alpha, K, \Sigma^K) = \emptyset$. For $z > 0$ assume that cons_{z-1} is already defined. We define $\mathsf{cons}_z(\alpha, K, \Sigma^K)$ as the set containing all run descriptors $(q_0, \Psi^n, \Psi^{n-1}, \ldots, \Psi^1, m', q')$ such that

1. $q' = q_0$ and $m' = \mathbf{1}_M$ is the identity element of M, or
2. $\mathcal{S}$ has a transition $(q_0, \alpha, a, q_1, \mathsf{pop}^k)$, $m' = \varphi(a)m$, and in Ψ^k there is a run descriptor $(q_1, \Phi^n, \Phi^{n-1}, \ldots, \Phi^{k+1}, m, q')$ such that $\Phi^i \subseteq \Psi^i$ for $k+1 \leq i \leq n$, or
3. $\mathcal{S}$ has a transition $(q_0, \alpha, a, q_1, \mathsf{col}^K)$, $m' = \varphi(a)m$, and in Σ^K there is a run descriptor $(q_1, \Phi^n, \Phi^{n-1}, \ldots, \Phi^{K+1}, m, q')$ such that $\Phi^i \subseteq \Psi^i$ for $K+1 \leq i \leq n$, or
4. $\mathcal{S}$ has a transition $(q_0, \alpha, a, q_1, \mathsf{push}^1_{\beta,k})$, $m' = \varphi(a)m$, and in $\mathsf{cons}_{z-1}(\beta, k, \Psi^k)$ there is a run descriptor $(q_1, \Phi^n, \Phi^{n-1}, \ldots, \Phi^1, m, q')$ such that $\Phi^i \subseteq \Psi^i$ for $2 \leq i \leq n$, and $\Phi^1 \subseteq \Psi^1 : \mathsf{cons}_{z-1}(\alpha, K, \Sigma^K)$, or
5. $\mathcal{S}$ has a transition $(q_0, \alpha, a, q_1, \mathsf{push}^k)$ where $k \geq 2$, $m' = \varphi(a)m$, and in $\mathsf{cons}_{z-1}(\alpha, K, \Sigma^K)$ there is a run descriptor $(q_1, \Phi^n, \Phi^{n-1}, \ldots, \Phi^1, m, q')$ such that $\Phi^i \subseteq \Psi^i$ for $1 \leq i \leq k-1$ and for $k+1 \leq i \leq n$, and $\Phi^k \subseteq \Psi^k : \Psi^{k-1} : \cdots : \Psi^1 : \mathsf{cons}_{z-1}(\alpha, K, \Sigma^K)$.

Notice that the sequence cons_z is monotone with respect to both z and Σ^K: for $\Sigma^K \subseteq \Sigma'^K$ and each $z \in \mathbb{N}$ we have $\mathsf{cons}_z(\alpha, K, \Sigma^K) \subseteq \mathsf{cons}_{z+1}(\alpha, K, \Sigma'^K)$. Independent of $z \in \mathbb{N}$, the domain and range of cons_z are fixed finite sets whence there is some $z_\infty \in \mathbb{N}$ such that $\mathsf{cons}_{z_\infty} = \mathsf{cons}_{z_\infty - 1}$. This fixpoint is denoted as cons. Next, we define types of stacks of arbitrary level.

▶ **Definition 4.5.** We define $\mathsf{type}(s^k)$ for each k-stack s^k (for $0 \leq k \leq n$) by induction on the structure of s^k. Assume that $k = 0$ and $s^k = (\alpha, K, t^K)$ where $\alpha \in \Gamma$, $1 \leq K \leq n$, and t^K is a K-stack such that $\mathsf{type}(t^K)$ is already defined. In this case we set $\mathsf{type}(s^k) = \mathsf{cons}\big(\alpha, K, \mathsf{type}(t^K)\big)$. Otherwise $k \geq 1$; if s^k is empty, we set $\mathsf{type}(s^k) = \emptyset$. Finally, assume

that $k \geq 1$ and $s^k = t^k : t^{k-1}$ such that $\mathsf{type}(t^k)$ and $\mathsf{type}(t^{k-1})$ are already defined. In this case we set $\mathsf{type}(s^k) = \mathsf{type}(t^k) : \mathsf{type}(t^{k-1})$.

Observe that $\mathsf{type}(s^k : s^{k-1} : \cdots : s^l) = \mathsf{type}(s^k) : \mathsf{type}(s^{k-1}) : \cdots : \mathsf{type}(s^l)$. From Definition 4.3 it follows that we have $(q, \Sigma^n, \Sigma^{n-1}, \ldots, \Sigma^{k+1}, m, q') \in \Psi^k : \Psi^{k-1} : \cdots : \Psi^l$ if and only if in Ψ^l there is a run descriptor $(q, \Sigma^n, \Sigma^{n-1}, \ldots, \Sigma^{l+1}, m, q')$ such that $\Sigma^i \subseteq \Psi^i$ for $l+1 \leq i \leq k$. It follows that the second condition of Lemma 4.2 for one value of l immediately implies this condition for all other values of l.

The proof of the "1⇒2" implication of Lemma 4.2 is almost a straightforward induction on the length of the run (we prove it for $l = 0$); the proof is in fact slightly complicated because we are using $\mathcal{P}_{\leq 1}(\mathcal{T}^n)$ instead of $\mathcal{P}(\mathcal{T}^n)$ in the definition of type, but these are just technical complications. The opposite implication follows from [11], and also follows from the facts proved later in this paper: we not only prove that the run (from item 1 of the lemma) exists, but we construct a deterministic CPS which simulates some (arbitrarily chosen) such run.

Next, let us calculate the number of run descriptors.

▶ Proposition 4.6.

$$\sum_{k=1}^{n} |\mathcal{T}^n| \leq |\mathcal{T}^0| \leq \frac{1}{2} \exp_{n-1}(4|Q|^2).$$

4.1 The algorithm

Let us now describe the algorithm for the emptiness problem. By Lemma 4.2 there is an accepting run from the initial configuration if and only if $(q_I, \emptyset, \emptyset, \ldots, \emptyset, m, q') \in \mathsf{cons}(\bot, 1, \emptyset)$ for any m and any accepting state q', where q_I is the initial state and $\bot$ the initial stack symbol (recall that the type of an empty stack is empty). Thus it is enough to calculate the sets $\mathsf{cons}(\alpha, K, \Sigma^K)$ for all values of α, K, Σ^K. We do that directly from Definition 4.4. Notice that it can be done in time polynomial in the size of $\mathcal{T}^0$ and in the number of triples (α, K, Σ^K), which by Proposition 4.6 are $n-1$ times exponential in the number of states, and polynomial in the size of the stack alphabet.

Let us remark that in the same way we can check whether from any configuration (q, s) there is an accepting run: it is enough to calculate $\mathsf{type}(s)$ (the algorithm is $n-1$ times exponential in the number of states, and polynomial in the size of the stack alphabet and in the size of s).

5 Computing the types

In this section we show that a CPS can at each moment maintain the type of its current stack. In fact the same can be done for any homomorphism, defined as follows. A *stack algebra of level* n $\mathcal{A} = (A^0, A^1, \ldots, A^n)$ over a stack alphabet Γ is an algebra which has $n+1$ sorts (of level $0, 1, \ldots, n$) and for each $1 \leq k \leq n$ operations $\mathsf{empty}^k : A^k$, $\mathsf{comp}^k : A^k \times A^{k-1} \to A^k$ (which we denote using the colon symbol), and $\mathsf{cons}(\alpha, k, \cdot) : A^k \to A^0$ for each $\alpha \in \Gamma$. A typical stack algebra of level n is the algebra of all stacks: in the sort of level k we have stacks of level k; empty^k returns an empty stack of level k, $\mathsf{comp}^k(s^k, s^{k-1}) = s^k : s^{k-1}$ puts stack s^{k-1} on top of stack s^k, and $\mathsf{cons}(\alpha, k, s^k)$ constructs the 0-stack with label α and a link to s^k. Notice that it is in fact the free algebra (without generators). But the set of all run descriptors is also a stack algebra of level n: $\mathcal{P}(\mathcal{T}^k)$ is the sort of level k, and the empty^k operation returns the empty set. Moreover type is the unique homomorphism between these two algebras.

Let $\mathcal{A} = (A^0, A^1, \ldots, A^n)$ be a finite stack algebra of level n (letter A is also used to denote the input alphabet), and f the unique homomorphism from the algebra of stacks to $\mathcal{A}$. We define f-driven n-CPS's as an extension of n-CPS's which work as follows. The transitions of an *f-driven n-CPS* are elements of

$$Q \times A^n \times A^{n-1} \times \cdots \times A^1 \times \Gamma \times \Big(\bigcup_{k=1}^{n} \{k\} \times A^k \Big) \times A \times Q \times OP_\Gamma^n.$$

From a configuration $(q, s^n : s^{n-1} : \cdots : s^1 : (\alpha, k, t^k))$ we can perform a transition

$$(q, f(s^n), f(s^{n-1}), \ldots, f(s^1), \alpha, k, f(t^k), a, q', op),$$

which reads letter a, changes state to q', and performs operation op on the stack. In other words, the set of transitions which can be applied to a configuration depend not only on the state and the topmost stack symbol, but also on the values of f on all stacks s^i (the topmost i-stack with removed its topmost $(i-1)$-stack) and on the stack t^k (the stack to which we have a link in the topmost 0-stack). The *size* of an f-driven n-CPS is the number of its transitions.

We have the following lemma (which is very similar to Theorem 3 in [4]). It says that a CPS can maintain the values of f applied to its current stack, so its transitions may depend on these values (like it is for an f-driven CPS). The proof is not difficult: we just have to keep in the topmost symbol of each substack (of any level) the value of f applied to this substack.

▶ **Lemma 5.1.** *Let $\mathcal{S}$ be a strongly deterministic f-driven n-CPS. Then there exists a strongly deterministic n-CPS $\mathcal{S}'$ such that $T(\mathcal{S}) = T(\mathcal{S}')$. For every $F \subseteq Q$ we have $L(\mathcal{S}, F) = L(\mathcal{S}, F)$ (in particular $\mathcal{S}'$ contains the states of $\mathcal{S}$); if from every reachable configuration of $(\mathcal{S}, F)$ there is an accepting run, the same holds for $(\mathcal{S}', F)$. The size of $\mathcal{S}'$ is linear in the size of $\mathcal{S}$, and $\mathcal{S}'$ can be computed in such time.*

The proof of Theorem 1.2 becomes now straightforward. Notice that, for an n-stack $s = s^n : s^{n-1} : \cdots : s^1 : (\alpha, k, t^k)$, it is enough to know α and k and $\mathsf{type}(t^k)$ and $\mathsf{type}(s^i)$ for all $1 \leq i \leq n$ in order to determine $\mathsf{type}(op(s))$ for any stack operation op (for example $\mathsf{type}(\mathsf{pop}^j(s)) = \mathsf{type}(s^n) : \mathsf{type}(s^{n-1}) : \cdots : \mathsf{type}(s^j)$, etc.). Moreover $\mathsf{type}(op(s))$ determines if there is an accepting run from configuration $(q, op(s))$ (by Lemma 4.2 taken for $l = n$ such run exists if and only if $(q, m, q') \in \mathsf{type}(op(s))$ for some m and some $q' \in F$).[6] Denote by $good(q, op)$ the set of those tuples

$$(\mathsf{type}(s^n), \mathsf{type}(s^{n-1}), \ldots, \mathsf{type}(s^1), \alpha, k, \mathsf{type}(t^k))$$

for which there is an accepting run from configuration $(q, op(s))$. Then from $(\mathcal{S}, F)$ we construct a type-driven n-CPS $(\mathcal{S}', F)$, in which from every reachable configuration there is an accepting run, as follows. For each transition (q, α, a, q', op) of $\mathcal{S}$, to $\mathcal{S}'$ we add those transitions

$$(q, \Psi^n, \Psi^{n-1}, \ldots, \Psi^1, \alpha, k, \Sigma^k, a, q', op)$$

for which $(\Psi^n, \Psi^{n-1}, \ldots, \Psi^1, \alpha, k, \Sigma^k) \in good(q', op)$. So in $\mathcal{S}'$ a transition will be performed only if it leads to a configuration from which there is an accepting run. Moreover $(\mathcal{S}', F)$

[6] This is particular means that op can be applied to s: if $op = \mathsf{pop}^k$ (or col^k) would result in an empty k-stack, $\mathsf{type}(op(s))$ would be empty.

accepts the same words as $(\mathcal{S}, F)$ since we do not remove transitions of accepting runs. By applying Lemma 5.1 we obtain from $(\mathcal{S}', F)$ a deterministic n-CPS having the same property.[7] Let us calculate the size (the number of transitions) of $\mathcal{S}'$ (hence of the resulting n-CPS). Notice that $|\mathcal{T}^n|$ and $|\mathcal{T}^{n-1}|$ are polynomial in the size of $\mathcal{S}$, and $|\mathcal{T}^i| = |\mathcal{T}^{i+1}| \cdot 2^{|\mathcal{T}^{i+1}|}$ for $i < n-1$ is $n-1-i$ times exponential in the size of $\mathcal{S}$. In particular $|\mathcal{T}^1|$ is the greatest and is $n-2$ times exponential in the size of $\mathcal{S}$. The transitions of $\mathcal{S}'$ are defined for subsets of $\mathcal{T}^i$, hence their number is at most $n-1$ times exponential in the size of $\mathcal{S}$.

6 Reconstructing a run

In this section we sketch how Theorem 1.1 can be proved. Fix some n-CPS $\mathcal{S}$ (with states Q), for which $T(\mathcal{S})$ is deterministic. As a first step we would like to construct a deterministic type-driven n-CPS $\mathcal{R}$ which would uniquely choose some run of $\mathcal{S}$ in the following sense. System $\mathcal{R}$ has the same stack alphabet as $\mathcal{S}$, and for each state $q \in Q$ of $\mathcal{S}$, system $\mathcal{R}$ also has the state q, as well a state $\mathsf{start}_{q,a}$ for each $a \in A$. Assume that $\mathcal{S}$ has a run whose labels form a word from $\varepsilon^* a$ (for some $a \in A$), starting in a reachable configuration $c = (q, s)$, and ending in a configuration $d = (q', s')$ (by determinism of $T(\mathcal{S})$, there is at most one such d for given a and c). Then in $\mathcal{R}$ there is a run from $(\mathsf{start}_{q,a}, s)$ to d using only ε-transitions, and not using states from Q (except its last configuration, which is d).

Assume we have such $\mathcal{R}$.[8] Then we can construct a strongly deterministic type-driven n-CPS $\mathcal{S}'$ such that $T(\mathcal{S}) = T(\mathcal{S}')$ as follows. Let $\mathcal{G}$ be the ε-closure of the configuration graph of $\mathcal{S}$, and G its set of nodes; in other words G contains the initial configuration and all configurations reachable by a run which ends by a non-ε-transition. Let also $H \subseteq G$ be its subsets containing only those of them, from which there exists a run containing some non-ε-transition (i.e. a run to another configuration from G). The set of configurations of $\mathcal{S}'$ which have state from Q are exactly those from H. In particular the initial configuration of $\mathcal{S}$ and $\mathcal{S}'$ is the same. All non-ε-transitions will be going from elements of H (i.e. configurations with state from Q). For each such configuration $c = (q, s)$, type determines the labels of edges outgoing from c in $\mathcal{G}$. Indeed, there is an edge from c to some $d \in G$ labelled by some $a \in A$ if and only if there is a run from c whose labels form a word from $\varepsilon^* a$; thus if and only if $(q, m, q') \in \mathsf{type}(s)$ for any element m which is the image of a word from $\varepsilon^* a$ and for any q'. Notice also that configuration d is unique for given c and a, by determinism of $T(\mathcal{S})$. Moreover, type determines whether $d \in H$: this is the case when there is a run from c whose labels form a word from $\varepsilon^* a \varepsilon^* A$; thus when $(q, m, q') \in \mathsf{type}(s)$ for any element m which the image of a word from $\varepsilon^* a \varepsilon^* A$ and for any q'. In such situation we add an a-labelled transition to the state $\mathsf{start}_{q,a}$ which does not change the stack (e.g. we make a push and then a pop). Then the system will simulate deterministically some run to d, making only ε-transitions (this is already realized by $\mathcal{R}$). Otherwise (for $d \in G \setminus H$), we just make an a-labelled transition to a dead state q_{die}, from which there are no more transitions. This way $T(\mathcal{S}) = T(\mathcal{S}')$, and $\mathcal{S}'$ is strongly deterministic.

It remains to construct $\mathcal{R}$. Let $c = (q, s^n : s^{n-1} : \cdots : s^0)$ be a configuration such that in $\mathsf{type}(s^0)$ we have a run descriptor $\sigma = (q, \Psi^n, \Psi^{n-1}, \ldots, \Psi^1, m, q')$ such that $\Psi^i \subseteq \mathsf{type}(s^i)$ for $1 \leq i \leq n$. Lemma 4.2 says that then there exists a run from c which evaluates to m under φ and ends in state q'. But how to simulate it using a type-driven CPS? It is easy to

[7] In order to obtain strong determinism, an additional step has to be performed.
[8] In the actual construction $\mathcal{R}$ is slightly more complicated (we keep some additional information on the stack).

perform some first step of such run; we just need to follow a transition used in Definition 4.4 to add σ to $\mathsf{type}(s^0)$. Moreover this leads to a configuration which again satisfies the same condition, so we can repeat the same. When we reach a configuration in which the first point of Definition 4.4 is used, we are done: we have finished the run.

The problem is that by such construction we can obtain an infinite run (e.g. a loop). The reason why the "2⇒1" implication of Lemma 4.2 holds is that cons is defined as the smallest fixpoint, not any fixpoint. According to the definition of cons_z, after a push operation it is enough to use cons_{z-1} both for the 0-stack which was topmost till now, and for the new topmost 0-stack. Thus we should keep the maximal allowed value of z with each 0-stack, and decrease this value for each push operation. Then for calculating the type of a bigger stack we should only use this cons_{z-1}, not whole cons.

There is also a technical difficulty that the values of z stored with each symbol have to be reset everywhere to z_∞ (recall that z_∞ is a number for which $\mathsf{cons} = \mathsf{cons}_{z_\infty}$) when we finally reach a configuration from G, and we want to run the simulation again. Of course we cannot do this explicitly, but we can just put some marker on the stack, which denotes that the value of z is z_∞ everywhere below. Then above the marker we work as previously, and we assume that bellow the marker all symbols have z_∞ on its second coordinate. More precisely, we need a separate kind of marker for each level: in each k-stack we will mark the bottommost $(k-1)$-stack in which the z values are actual; we also need another kind of marker to denote the 0-stacks in which the z values in the link are not actual.

▶ **Example 6.1.** Consider the 1-CPS having the following transitions.

$$(\bot, q_1, a, q_2, \mathsf{push}^1_{x,1}) \quad (x, q_3, \varepsilon, q_1, \mathsf{push}^1_{x,1}) \quad (x, q_1, \varepsilon, q_1, \mathsf{pop}^1)$$
$$(x, q_2, \varepsilon, q_3, \mathsf{push}^1_{x,1}) \quad (x, q_3, \varepsilon, q_2, \mathsf{pop}^1)$$

Then the types contain the following run descriptors (not all are shown).

$$(q_1, \emptyset, \varepsilon^* a, q_2) \in \mathsf{cons}(\bot, 1, \emptyset) \qquad (q_2, \{\tau\}, \varepsilon^* a, q_2) \in \mathsf{cons}_3(x, 1, \emptyset)$$
$$\tau = (q_1, \varepsilon^* a, q_2) \in \mathsf{type}([\bot]) \qquad \sigma = (q_2, \varepsilon^* a, q_2) \in \mathsf{type}([\bot x])$$
$$(q_1, \{\tau\}, \varepsilon^* a, q_2) \in \mathsf{cons}_1(x, 1, \emptyset) \quad (q_3, \{\tau, \sigma\}, \varepsilon^* a, q_2) \in \mathsf{cons}_1(x, 1, \emptyset)$$
$$(q_3, \{\tau\}, \varepsilon^* a, q_2) \in \mathsf{cons}_2(x, 1, \emptyset)$$

The key decision to take is in configuration $(q_3, [\bot x x])$. By just choosing the run descriptor $(q_3, \{\tau, \sigma\}, \varepsilon^* a, q_2)$ from cons_z for the smallest z ($z = 1$) we make pop^1 leading to $(q_2, [\bot x])$. Now the only possibility is to use $(q_2, \{\tau\}, \varepsilon^* a, q_2) \in \mathsf{cons}_3(x, 1, \emptyset)$, and to perform $\mathsf{push}^1_{x,1}$. If we just went to $(q_3, [\bot x x])$, we would fall into a loop. Also restricting z used in this configuration (for the new topmost x) does not help. We should instead go to a configuration $(q_3, [\bot x' x'])$, where x' is a version of x which implies that only $z \leq 2$ can be used. Then $\sigma \notin \mathsf{type}([\bot x'])$, so we cannot perform the pop^1 transition again. We have to use $(q_3, \{\tau\}, \varepsilon^* a, q_2) \in \mathsf{cons}_2(x, 1, \emptyset)$, thus we perform $\mathsf{push}^1_{x,1}$ and we leave the loop.

6.1 Bound on the run length

Next we notice that the method described above not only guarantees that the obtained run is finite, but also gives a concrete bound on its length. Let us repeat that in the system $\mathcal{R}$, which deterministically simulates one run of $\mathcal{S}$, we keep with each stack symbol a natural number z (which is between 0 and z_∞). Whenever we perform some push^k operation, this value is decreased in the topmost 0-stack, both in the original $(k-1)$-stack and in its copy. We want to argue that every run working like that will terminate. In order to obtain that we define potential of a stack.

▶ **Definition 6.2.** Let $0 \leq k \leq n$, and let s^k be a k-stack over alphabet $\Gamma \times \{0, 1, \ldots, z_\infty\}$. We define a natural number $\mathsf{pt}(s^k)$ (the *potential* of s^k) by induction on the structure of s^k.

- When $k = 0$ and $s^k = ((\alpha, z), K, t^K)$, we take $\mathsf{pt}(s^k) = 2z$.
- When $k \geq 1$ and s^k is empty, we take $\mathsf{pt}(s^k) = 0$.
- When $k \geq 1$ and $s^k = t^k : t^{k-1}$, we take

$$\mathsf{pt}(s^k) = \mathsf{pt}(t^k) + 2^{\mathsf{pt}(t^{k-1})}.$$

It is not difficult to see that the potential is decreased by each operation (assuming that the push operations behave as described above). Thus the length of the run reconstructed by $\mathcal{R}$ is bounded by the potential of the starting configuration. If we forget about the system $\mathcal{R}$, we obtain the following strengthened version of the '2⇒1" implication of Lemma 4.2 (for $l = n$).

▶ **Lemma 6.3.** *Let $c = (q, s)$ be a configuration of $\mathcal{S}$, and let $\widetilde{s}$ be the stack over alphabet $\Gamma \times \{0, 1, \ldots, z_\infty\}$ obtained from s by appending z_∞ to each stack symbol. Assume that $(q, m, q') \in \mathsf{type}(s)$. Then there exists a run from c which evaluates to m under φ, ends in state q', and has length at most $\mathsf{pt}(\widetilde{s})$.*

Let us conclude by a proof of Theorem 1.5. By assumption some accepting run exists from the initial configuration (q_I, s_I). By Lemma 4.2 this means that $(q_I, m, q') \in \mathsf{type}(s_I)$ for some $m \in M$ and some accepting state q'. Using the above lemma we obtain back an accepting run, but now its length is bounded by the potential of the initial configuration. Notice that the potential is n times exponential in z_∞. Moreover z_∞ is $n-1$ times exponential in the number of states of the system, which follows from Proposition 4.6. Thus we obtain an accepting run of length $2n - 1$ times exponential in the size of the system; a precise calculation gives the value written in Theorem 1.5.

7 Conclusions

Let us mention that the methods presented in this paper can be used in a much more general context. In [11] we define types of stacks which characterize a lot of interesting kinds of runs, called top^k-non-erasing runs, pumping runs, k-returns, k-colreturns; additionally a general definition is given, which allows to consider also other classes of runs having some properties. To be concrete: we can consider for example a class of runs such that the topmost k-stack at the end is equal to the topmost k-stack at the beginning, and is indeed obtained as its copy. It is almost immediate to generalize the results of this paper to all these classes of runs:

- The number of run descriptors for these more general types is $n - 1$ times exponential.
- The type function is still a homomorphism of stack algebras, so it can be computed by the CPS, as in Section 5.
- We can deterministically reconstruct runs of that kind, like in Section 6.
- The length of such runs can be bounded like in Lemma 6.3.

Observe that most of these classes of runs cannot (at least in any natural way) be defined using μ-calculus in the configuration graph.

Techniques similar to ours were used independently in a recent paper [3]. Moreover some of our results can be deduced from another independent work [6].

Future work. One open question is whether the hierarchy of word languages recognized by collapsible pushdown systems is strict (if every two its levels are different). Notice that we only show that the hierarchy is infinite (that there are infinitely many different levels). One possible way to obtain the strictness would be to show that if two levels coincide, then all higher levels have to coincide (however we do not see any easy reason for that). A second way would be to improve the bounds given in our proof.

It is also an open problem whether these languages are context-sensitive.

A second direction would be to extend our approach of testing emptiness to the μ-calculus model checking problem (and obtain a simple algorithm for that problem, which uses collapsible pushdown systems directly).

Another interesting question is about the relation between the classes of word languages recognized by collapsible and non-collapsible pushdown systems. It is known that on level 2 these classes coincide [1] (unlike for trees and graphs); nevertheless the conjecture is that for higher levels these classes are different.

References

1 K. Aehlig, J. G. de Miranda, and C.-H. L. Ong. Safety is not a restriction at level 2 for string languages. In V. Sassone, editor, *FoSSaCS*, volume 3441 of *Lecture Notes in Computer Science*, pages 490–504. Springer, 2005.

2 A. Blumensath. On the structure of graphs in the Caucal hierarchy. *Theor. Comput. Sci.*, 400(1-3):19–45, 2008.

3 C. Broadbent, A. Carayol, M. Hague, and O. Serre. A saturation method for collapsible pushdown systems. To appear in ICALP, 2012.

4 C. H. Broadbent, A. Carayol, C.-H. L. Ong, and O. Serre. Recursion schemes and logical reflection. In *LICS*, pages 120–129. IEEE Computer Society, 2010.

5 C. H. Broadbent and C.-H. L. Ong. On global model checking trees generated by higher-order recursion schemes. In L. de Alfaro, editor, *FOSSACS*, volume 5504 of *Lecture Notes in Computer Science*, pages 107–121. Springer, 2009.

6 A. Carayol and O. Serre. Collapsible pushdown automata and labeled recursion schemes. Equivalence, safety and effective selection. To appear in LICS, 2012.

7 A. Carayol and S. Wöhrle. The Caucal hierarchy of infinite graphs in terms of logic and higher-order pushdown automata. In P. K. Pandya and J. Radhakrishnan, editors, *FSTTCS*, volume 2914 of *Lecture Notes in Computer Science*, pages 112–123. Springer, 2003.

8 D. Caucal. On infinite terms having a decidable monadic theory. In K. Diks and W. Rytter, editors, *MFCS*, volume 2420 of *Lecture Notes in Computer Science*, pages 165–176. Springer, 2002.

9 W. Damm. The IO- and OI-hierarchies. *Theor. Comput. Sci.*, 20:95–207, 1982.

10 M. Hague, A. S. Murawski, C.-H. L. Ong, and O. Serre. Collapsible pushdown automata and recursion schemes. In *LICS*, pages 452–461. IEEE Computer Society, 2008.

11 A. Kartzow and P. Parys. Strictness of the collapsible pushdown hierarchy. *CoRR*, abs/1201.3250, 2012.

12 T. Knapik, D. Niwinski, and P. Urzyczyn. Higher-order pushdown trees are easy. In M. Nielsen and U. Engberg, editors, *FoSSaCS*, volume 2303 of *Lecture Notes in Computer Science*, pages 205–222. Springer, 2002.

13 N. Kobayashi and C.-H. L. Ong. Complexity of model checking recursion schemes for fragments of the modal mu-calculus. In S. Albers, A. Marchetti-Spaccamela, Y. Matias, S. E. Nikoletseas, and W. Thomas, editors, *ICALP (2)*, volume 5556 of *Lecture Notes in Computer Science*, pages 223–234. Springer, 2009.

14 A. N. Maslov. The hierarchy of indexed languages of an arbitrary level. *Soviet Math. Dokl.*, 15:1170–1174, 1974.

15 A. N. Maslov. Multilevel stack automata. *Problems of Information Transmission*, 12:38–43, 1976.

16 C.-H. L. Ong. On model-checking trees generated by higher-order recursion schemes. In *LICS*, pages 81–90. IEEE Computer Society, 2006.

17 P. Parys. On the significance of the collapse operation. Accepted to LICS 2012, 2012.

18 S. Salvati and I. Walukiewicz. Krivine machines and higher-order schemes. In L. Aceto, M. Henzinger, and J. Sgall, editors, *ICALP (2)*, volume 6756 of *Lecture Notes in Computer Science*, pages 162–173. Springer, 2011.