Accelerating tree-automatic relations

Anthony Widjaja Lin

Oxford University Department of Computer Science

Abstract

We study the problem of computing the transitive closure of tree-automatic (binary) relations, which are represented by tree automata. Such relations include classes of infinite systems generated by pushdown systems (PDS), ground tree rewrite systems (GTRS), PA-processes, and Turing machines, to name a few. Although this problem is unsolvable in general, we provide a semi-algorithm for the problem and prove completeness guarantee for PDS, GTRS, and PA-processes. The semi-algorithm is an extension of a known semi-algorithm for structure-preserving tree-automatic relations, for which completeness is guaranteed for several interesting parameterized systems over tree topology. Hence, there is a single generic procedure that solves reachability for PDS, GTRS, PA-processes, and several parameterized systems in a *uniform* way. As an application, we provide a single generic semi-algorithm for checking repeated reachability over tree-automatic relations, for which completeness is guaranteed for the aforementioned classes.

1998 ACM Subject Classification F.4 Mathematical Logic and Formal Languages

Keywords and phrases Semi-algorithm, Model Checking, Infinite Systems, Automata

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2012.313

1 Introduction

Real-world systems are complex and many sources of infinity naturally arise when modeling them formally, e.g., recursions in function calls, data structures (lists, trees, etc.) of unbounded size, numeric variables of unbounded size, multi-threading (unbounded number of threads spawned). Given the modeling power of infinite-state systems, even checking simple properties (e.g reachability) easily becomes undecidable over simple classes of infinite systems (e.g. those represented by counter machines). Despite this, many interesting properties (e.g. safety, liveness, temporal-logic specifications) have been shown to be decidable over many classes of infinite-state systems (e.g. pushdown systems, timed systems, Petri nets, lossy channel systems, ground tree rewrite systems, and process rewrite systems). We refer the reader to [1, 12, 24, 26, 25, 27, 31] for a glimpse of these decidability results.

Another common approach to infinite-state model checking is to start with expressive ("Turing-powerful") formalisms that can capture many complex real world features and develop semi-algorithms for model checking that can solve many practical instances. A popular framework for reasoning about complex infinite-state systems has been proposed under the rubric of *regular model checking* (e.g. see [5, 7, 9, 28]), in which systems are represented by "regular" symbolic representations including finite-state automata (over words, trees, ω-words, etc.) or logical formulas over a decidable theory (e.g. Presburger formulas). Many regular model checking frameworks (differing in expressive power) have been considered in the literature, including (extended) counter systems (e.g. see [9, 7]), rational graphs (e.g. see [18, 5]), word-automatic graphs with length-preserving relations (e.g. see [5, 28]), tree-automatic graphs with structure-preserving relations (e.g. see [4, 14]), tree transducers (e.g. see [14]), and a natural subclass of ω-word automatic graphs (e.g. see [11]).


© Anthony W. Lin;
licensed under Creative Commons License NC-ND
32nd Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2012).
Editors: D. D'Souza, J. Radhakrishnan, and K. Telikepalli; pp. 313–324

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

Two commonly considered problems in regular model checking are: (1) given a regular symbolic representation of a set X of configurations, compute a regular symbolic representation of the set $post^*(X)$ (or $pre^*(X)$) of reachable configurations, and (2) compute a regular symbolic representation of the transitive closure $\mathcal{R}^*$ of the transition relation $\mathcal{R}$. The second problem is more general than the first (e.g. see [5, 29]). Although these problems are uncomputable in general, semi-algorithms have been developed which can solve these problems for many interesting practical instances (e.g. see [4, 5, 7, 11, 14, 18, 28]), which include *parameterized systems* (i.e. distributed protocols with *any* number of components with some underlying topology, e.g., the dining philosopher problem), which cannot easily be captured by known decidable subclasses. Such semi-algorithms usually employ certain *acceleration* techniques, which compute the effect of arbitrarily long sequences of transitions.

From a theoretical perspective, an important problem when designing semi-algorithms in regular model checking is undoubtedly the question of convergence and completeness. As has been pointed out in [7], many existing semi-algorithms in regular model checking lack general completeness criteria. For example, it was not known if there is a generic semi-algorithm in regular model checking with *completeness guarantee* (i.e. always terminates and gives correct solution) over the *full* class of pushdown systems, let alone strictly more powerful formalisms like ground tree rewrite systems.

Contributions. We investigate regular model checking under the framework of tree-automatic structures [8]. More precisely, we study the problem of computing the transitive closure of a given *tree-automatic (binary) relation* [8]. Such relations are represented by the standard tree-automata over a product alphabet (extended by a padding symbol) allowing configurations (i.e. trees) to grow unboundedly in some paths in the systems. Tree-automatic relations are expressive, i.e., can model the transition graphs of pushdown systems (PDS), PA-processes (PA), ground tree rewrite systems (GTRS), Petri nets, and even Minsky's counter machines and Turing machines.

Our contributions are as follows. We start by showing that the *bounded local depth acceleration* technique given in [4] for structure-preserving tree-automatic relations (i.e. which only relate trees with the same structure) can be extended to the full class of tree-automatic relations. Roughly speaking, for each k and a tree-automatic relation $\mathcal{R}$, the resulting algorithm computes a tree-automatic relation $\lfloor R \rfloor_k$ such that each pair $(T, T') \in \lfloor R \rfloor_k$ has a witnessing path in $\mathcal{R}$, wherein each node is "modified" at most k times. In the case when $\lfloor R \rfloor_k$ is transitive (which can be effectively checked), we have $\mathcal{R}^* = \lfloor R \rfloor_k$. The semi-algorithm simply tries to find a number k such that $\lfloor R \rfloor_k$ is transitive. Abdulla et al. has given some interesting instances of parameterized systems with tree topology for which the semi-algorithm (for structure-preserving case) is guaranteed to terminate [4]. Our main contribution is to show that the extended semi-algorithm is also guaranteed to terminate for several well-studied classes of infinite systems including PDS, PA-processes, and GTRS. As an application, we combine this with the result from [29] to obtain a semi-algorithm for testing repeated reachability, which is guaranteed to terminate for PDS, PA, and GTRS. Hence, we have a *uniform* solution for (repeated) reachability for PDS, PA, and GTRS.

Discussion and Related Work. There are known specialized algorithms for computing the reachability relations (i.e. transitive closure) for PDS [15], PA-processes [26], and GTRS [19, 16] in polynomial time. Although our semi-algorithm has worse upper bounds on the running time, it is more general since it can solve instances of parameterized systems with tree topology [4]. The purpose of this paper has *never* been to provide more efficient algorithms for PA-processes, GTRS, and PDS. Rather, we only intend to show the possibility of devising a *single* generic semi-algorithm that is guaranteed to terminate for the aforementioned classes

of systems, as well as other systems that cannot be easily captured by known decidable subclasses. We leave it for future work to evaluate how the semi-algorithms perform in practice on PA-processes, PDS, and GTRS, and if better acceleration techniques can be devised for them.

Bouajjani & Touili [14] gave a semi-algorithm for computing the reachability sets in the framework of regular tree model checking, where tree transducers (and structure-preserving tree-automatic relations) are adopted. Among others, their semi-algorithms are guaranteed to compute the reachability sets (but not reachability relations) for process rewrite systems (which subsume PA), GTRS (and its extensions), provided that the relations are well-founded (i.e. have no infinite decreasing chain). Process rewrite systems and GTRS are in general not well-founded, but they showed that PA can be transformed into a well-founded PA while preserving reachability. The class of relations generated by tree transducers is more expressive than tree-automatic relations in general, but is less well-behaved than tree-automatic relations (e.g. see [16] and [8]). In particular, it is open whether we can decide repeated reachability, given a tree transducer that generates $\mathcal{R}^*$ (which is the case for tree-automatic relations [29]). There are also other acceleration techniques in regular tree model checking (e.g. see [3, 6, 13]), but they do not have termination guarantee for PA-processes, PDS, and GTRS.

Using *flat acceleration* techniques [7, 17], semi-algorithms for computing the reachability sets (e.g. represented as Presburger formulas) over extended counter systems can be developed that are guaranteed to solve the subcases of reversal-bounded counter systems, and many interesting subclasses of Petri nets (e.g. 2-dim vector addition systems). A similar result of this form can be found in [10], which provides a semi-algorithm which is guaranteed to compute the reachability sets of timed automata (and more general hybrid systems).

2 Preliminaries

General notations For two given natural numbers $i \leq j$, we define $[i, j] = \{i, i+1, \ldots, j\}$. Define $[k] = [0, k]$. Given a set S, we use S^* to denote the set of all finite sequences of elements from S. The set S^* always includes the empty sequence which we denote by ϵ. Given two sets of words S_1, S_2, we use $S_1 \cdot S_2$ to denote the set $\{v \cdot w : v \in S_1, w \in S_2\}$ of words formed by concatenating words from S_1 with words from S_2. Given two relations $R_1, R_2 \subseteq S \times S$, we define their composition as $R_1 \circ R_2 = \{(s_1, s_3) : (\exists s_2)((s_1, s_2) \in R_1 \wedge (s_2, s_3) \in R_2)\}$.

Transition systems Let ACT be a finite set of *action symbols*. A *transition system* over ACT is a tuple $\mathfrak{S} = \langle S, \{\rightarrow_a\}_{a \in \mathsf{ACT}} \rangle$, where S is a set of *configurations*, and $\rightarrow_a \subseteq S \times S$ is a binary relation over S. We use $\rightarrow$ to denote the relation $\left(\bigcup_{a \in \mathsf{ACT}} \rightarrow_a\right)$. The notation $\rightarrow^+$ (resp. $\rightarrow^*$) is used to denote the transitive (resp. transitive-reflexive) closure of $\rightarrow$. We say that a sequence $s_1 \rightarrow \cdots \rightarrow s_n$ is a *path* (or *run*) in $\mathfrak{S}$ (or in $\rightarrow$). Given two paths $\pi_1 : s_1 \rightarrow^* s_2$ and $\pi_2 : s_2 \rightarrow^* s_3$ in $\rightarrow$, we may concatenate them to obtain $\pi_1 \odot \pi_2$ (by gluing together s_2). Given a relation $\rightarrow \subseteq S \times S$ and subsets $S_1, \ldots, S_n \subseteq S$, denote by $\textsc{Rec}_{\rightarrow}(\{S_i\}_{i=1}^n)$ to be the set of elements $s_0 \in S$ for which there exists an infinite path $s_0 \rightarrow s_1 \rightarrow \cdots$ visiting each S_i infinitely often, i.e., such that, for each $i \in [1, n]$, there are infinitely many $j \in \mathbb{N}$ with $s_j \in S_i$.

Trees, automata, and languages A *ranked alphabet* is a nonempty finite set of symbols Σ equipped with an arity function $\mathtt{ar} : \Sigma \rightarrow \mathbb{N}$. A *tree domain* D is a nonempty finite subset of $\mathbb{N}^*$ satisfying (1) *prefix closure*, i.e., if $vi \in D$ with $v \in \mathbb{N}^*$ and $i \in \mathbb{N}$, then $v \in D$, (2) *younger-sibling closure*, i.e., if $vi \in D$ with $v \in \mathbb{N}^*$ and $i \in \mathbb{N}$, then $vj \in D$ for each natural number $j < i$. The elements of D are called *nodes*. Standard terminologies (e.g. parents, children, ancestors, descendants) will be used when referring to elements of a tree domain.

For example, the children of a node $v \in D$ are all nodes in D of the form vi for some $i \in \mathbb{N}$. A *tree* over a ranked alphabet Σ is a pair $T = (D, \lambda)$, where D is a tree domain and the *node-labeling* λ is a function mapping D to Σ such that, for each node $v \in D$, the number of children of v in D equals the arity $\mathtt{ar}(\lambda(v))$ of the node label of v. We use the notation $|T|$ to denote $|D|$. Write $\textsc{Tree}(\Sigma)$ for the set of all trees over Σ. We also use the standard term representations of trees (cf. [16]).

A nondeterministic tree-automaton (NTA) over a ranked alphabet Σ is a tuple $\mathcal{A} = \langle Q, \Delta, F\rangle$, where (i) Q is a finite nonempty set of states, (ii) Δ is a finite set of rules of the form $(q_1, \ldots, q_r) \xrightarrow{a} q$, where $a \in \Sigma$, $r = \mathtt{ar}(a)$, and $q, q_1, \ldots, q_r \in Q$, and (iii) $F \subseteq Q$ is a set of final states. A rule of the form $() \xrightarrow{a} q$ is also written as $\xrightarrow{a} q$. A *run* of $\mathcal{A}$ on a tree $T = (D, \lambda)$ is a mapping ρ from D to Q such that, for each node $v \in D$ (with label $a = \lambda(v)$) with its all children $v_1, \ldots, v_r$, it is the case that $(\rho(v_1), \ldots, \rho(v_r)) \xrightarrow{a} \rho(v)$ is a transition in Δ. For a subset $Q' \subseteq Q$, the run is said to be *accepting at* Q' if $\rho(\epsilon) \in Q'$. It is said to be *accepting* if it is accepting at F. The NTA is said to *accept* T at Q' if it has an run on T that is accepting at Q'. Again, we will omit mention of Q' if $Q' = F$. The language $\mathcal{L}(\mathcal{A})$ of $\mathcal{A}$ is precisely the set of trees which are accepted by $\mathcal{A}$. A language L is said to be *regular* if there exists an NTA accepting L. In the sequel, we use $\|\mathcal{A}\|$ to denote the size of $\mathcal{A}$.

A *context* with *(context) variables* $x_1, \ldots, x_n$ is a tree $T = (D, \lambda)$ over the alphabet $\Sigma \cup \{x_1, \ldots, x_n\}$, where $\Sigma \cap \{x_1, \ldots, x_n\} = \emptyset$ and for each $i = 1, \ldots, n$, it is the case that $\mathtt{ar}(x_i) = 0$ and there exists a unique *context node* u_i with $\lambda(u_i) = x_i$. In the sequel, we will sometimes denote such a context as $T[x_1, \ldots, x_n]$. Intuitively, a context $T[x_1, \ldots, x_n]$ is a tree with n "holes" that can be filled in by trees in $\textsc{Tree}(\Sigma)$. More precisely, given trees $T_1 = (D_1, \lambda_1), \ldots, T_n = (D_n, \lambda_n)$ over Σ, we use the notation $T[T_1, \ldots, T_n]$ to denote the tree (D', λ') obtained by filling each hole x_i by T_i, i.e., $D' = D \cup \bigcup_{i=1}^{n} u_i \cdot D_i$ and $\lambda'(u_i v) = \lambda_i(v)$ for each $i = 1, \ldots, n$ and $v \in D_i$. Given a tree T, if $T = C[t]$ for some context tree $C[x]$ and a tree t, then t is called a *subtree* of T. If u is the context node of C, then we use the notation $T(u)$ to obtain this subtree t. Given an NTA $\mathcal{A} = \langle Q, \Delta, F\rangle$ over Σ and states $\bar{q} = q_1, \ldots, q_n \in Q$, we say that $T[x_1, \ldots, x_n]$ is accepted by $\mathcal{A}$ from $\bar{q}$ (written $T[q_1, \ldots, q_n] \in \mathcal{L}(\mathcal{A})$) if it is *accepted* by the NTA $\mathcal{A}' = \langle Q, \Delta', F\rangle$ over $\Sigma \cup \{x_1, \ldots, x_n\}$, where Δ' is the union of Δ and the set containing each rule of the form $\xrightarrow{x_i} q_i$.

3 Tree-automatic relations

Fix a nonempty ranked alphabet Σ. We reserve a special symbol $\perp$ such that $\perp \notin \Sigma$ with $\mathtt{ar}(\perp) := 0$. Let $\Sigma_\perp = \Sigma \cup \{\perp\}$, and $\overline{\perp} := (\perp, \perp)$. In order to define the notion of tree-automatic relations, we will need to first define the convolution operator $\otimes$, which maps a pair of trees over Σ into a tree over the "product alphabet" $\boldsymbol{\Sigma}_\perp := (\Sigma_\perp \times \Sigma_\perp) \setminus \overline{\perp}$ containing labels of the form (a, b) with arity $\mathtt{ar}((a, b)) := \max\{\mathtt{ar}(a), \mathtt{ar}(b)\}$. Given two trees $T_1 = (D_1, \lambda_1)$ and $T_2 = (D_2, \lambda_2)$ over Σ, their convolution is the tree $T_1 \otimes T_2 := (D_1 \cup D_2, \lambda)$ over $\boldsymbol{\Sigma}_\perp$ such that $\lambda(v) = (\lambda_1'(v), \lambda_2'(v))$, where λ_i' is the extension of the function λ_i to the domain $D_1 \cup D_2$ such that $\lambda_i'(w) = \perp$ whenever $w \notin D_i$.

Consider a binary relation $\mathcal{R} \subseteq \textsc{Tree}(\Sigma) \times \textsc{Tree}(\Sigma)$. We say that the relation is *tree-automatic* [8] (also see [16, Chapter 3]) if the language $\{T \otimes T' : (T, T') \in \mathcal{R}\}$ over $\boldsymbol{\Sigma}_\perp$ is regular. In this case, a *presentation* of $\mathcal{R}$ is any NTA recognizing the language. In the sequel, a presentation of tree-automatic relations is also referred to as a *synchronous (tree)-automaton*. The relation $\mathcal{R} \subseteq \textsc{Tree}(\Sigma) \times \textsc{Tree}(\Sigma)$ is said to be *structure-preserving* if it contains only pairs (T, T') of trees with the same tree domains. Therefore, a synchronous automaton presenting a structure-preserving binary relation runs over the alphabet $\boldsymbol{\Sigma} := \Sigma \times \Sigma$ (i.e.

may not take tuples involving $\perp$).

A *tree-automatic transition system* [8] is a transition system of the form $\mathfrak{S} = \langle S, \{\rightarrow_a\}_{a\in\mathsf{ACT}}\rangle$, where for some ranked alphabet Σ: (1) the domain S is the language of some NTA $\mathcal{A}_{Dom}$ over Σ, and (2) for each action $a \in \mathsf{ACT}$, $\rightarrow_a \subseteq \textsc{Tree}(\Sigma) \times \textsc{Tree}(\Sigma)$ is a tree-automatic relation presented by some NTA $\mathcal{A}_a$. The tuple $\upsilon = \langle \mathcal{A}_{Dom}; \{\mathcal{A}_a\}_{a\in\mathsf{ACT}}\rangle$ is said to be a *presentation* for the transition system $\mathfrak{S}$. Given a first-order formula $\varphi(x_1, \ldots, x_n)$ over the vocabulary $\{\rightarrow_a\}_{a\in\mathsf{ACT}}$, we write $[\![\varphi]\!]_{\mathfrak{S}}$ for the set of interpretations $(T_1, \ldots, T_n) \in \textsc{Tree}(\Sigma)^n$ such that $\mathfrak{S}$ satisfies the formula φ, i.e., $\mathfrak{S} \models \varphi(T_1, \ldots, T_n)$. When $\mathfrak{S}$ is understood, we simply write $[\![\varphi]\!]$. The following are basic results from the theory of (tree)-automatic structures.

▶ Proposition 1 ([8]). Given a tree-automatic transition system $\mathfrak{S}$ presented by the presentation υ and a first-order sentence φ over the vocabulary of $\mathfrak{S}$, checking whether $\mathfrak{S} \models \varphi$ is decidable. In fact, if φ has free variables x_1, x_2, then a synchronous automaton for $[\![\varphi]\!]_{\mathfrak{S}}$ is computable.

When the input formula is existential, model checking is solvable in exponential time [8]. This implies that given an NTA $\mathcal{A}$ presenting the relation $\mathcal{R}$, checking whether $\mathcal{R}$ is transitive can be done in exponential time. This is because non-transitivity can be expressed as $\exists x, y, z(\mathcal{R}(x, y) \wedge \mathcal{R}(y, z) \wedge \neg\mathcal{R}(x, z))$. A simple analysis of the proof of Proposition 1 (e.g. see [29]) also shows that given an automatic transition system $\mathfrak{S}$ presented by the presentation υ and a *fixed* existential positive (i.e. negation-free) first-order formula $\varphi(x_1, x_2)$, we may compute a synchronous automaton for $[\![\varphi]\!]$ in time polynomial in the size $\|\upsilon\|$ of υ.

4 Synchronized automata of finite local depth

In this section, we consider an arbitrary tree-automatic binary relation $\mathcal{R} \subseteq \textsc{Tree}(\Sigma) \times \textsc{Tree}(\Sigma)$ presented by any given synchronous automaton $\mathcal{A} = \langle Q, \Delta, F\rangle$ over the alphabet $\boldsymbol{\Sigma}_\perp$. We shall keep our terminologies as close to [4] as possible.

The *suffix* of a state $q \in Q$ is the set of contexts $T[x]$ accepted by $\mathcal{A}$ from q, i.e., $\mathit{suff}(q) := \{T[x] : T[q] \in \mathcal{L}(\mathcal{A})\}$. For a subset $Q' \subseteq Q$, define $\mathit{suff}(Q') := \bigcup_{q\in Q'} \mathit{suff}(q)$. A context $T[x] = (D, \tau)$ over the alphabet $\boldsymbol{\Sigma}_\perp$, with the unique node u such that $\tau(u) = x$, is said to be *copying* if for each $v \in D \setminus \{u\}$ it is the case that $\tau(v) = (a, a)$ for some $a \in \Sigma$. The state q is said to be *idempotent* if $\mathit{suff}(q)$ contains only copying contexts, and that, for each transition $(q_1, \ldots, q_m) \xrightarrow{(a,b)} q$, we have $a, b \neq \perp$. [The latter restriction is not imposed in the definition of copying contexts in [4], but is a necessary technical restriction.]

The *prefix* $\mathit{pref}(q)$ of a state $q \in Q$ is defined to be the set of trees accepted by $\mathcal{A}$ at $\{q\}$. A tree $T = (D, \tau)$ over $\boldsymbol{\Sigma}_\perp$ is said to be *copying* if each $v \in D$ satisfies $\tau(v) = (a, a)$ for some $a \in \Sigma$. The state q is said to be a *copying (prefix) state* if $\mathit{pref}(q)$ contains only copying trees.

Local depth Let $Q' \subseteq Q$. Consider a run $\pi := T_0 = (D_0, \tau_0), \ldots, T_m = (D_m, \tau_m)$ through $\mathcal{R}$ such that $T_i \otimes T_{i+1} \in \mathcal{L}(\mathcal{A})$ with a witnessing run ρ_i of $\mathcal{A}$, for each $i \in [m-1]$. Let D denote the set $\bigcup_{i=0}^{m} D_i$, and let $\bar\rho$ denote the sequence $\rho_0, \ldots, \rho_{m-1}$. The *$Q'$-local depth of a node $v \in D$ in π with respect to the runs $\bar\rho$* is defined to be the number of indices $i \in [m-1]$ such that $\rho_i(v) \in Q'$. Intuitively, it is the number of times v is "touched" by Q' in the runs $\bar\rho$. The *Q'-local depth of π with respect to $\bar\rho$* is the maximum of Q'-local depths of nodes $v \in D$ in π wrt $\bar\rho$. The *Q'-local depth of the run π* is the minimum of Q'-local depths of π with respect to *some* witnessing runs of $\mathcal{A}$. The *Q'-local depth of a pair $(T, T') \in \mathcal{R}^*$ of trees (with respect to $\mathcal{A}$)* is the minimum of Q'-local depths of runs $\pi = T_0, \ldots, T_m$ through $\mathcal{R}$,

where $T_0 = T$ and $T_m = T'$. The *Q'-local depth of the relation $\mathcal{R}$ (with respect to $\mathcal{A}$)* is the *supremum* over all Q'-local depths of pairs $(T, T') \in \mathcal{R}^*$. [To understand this last concept, an analogy to the notion of "diameter" of a graph G (supremum of lengths of paths in G) can be drawn.] Whenever Q' is clear, we shall omit mention of Q' and simply say "local depth".

▶ **Example 1.** Let $\Sigma = \{0, 1, \hat{0}, \hat{1}\}$ with $\mathtt{ar}(0) = \mathtt{ar}(1) = 0$ and $\mathtt{ar}(\hat{0}) = \mathtt{ar}(\hat{1}) = 2$. Define the relation $\mathcal{R} \subseteq \textsc{Tree}(\Sigma) \times \textsc{Tree}(\Sigma)$ containing tuples (T, T'), where $T = C[i]$ $(i = 0, 1)$ and $T' = C[\hat{i}(j, k)]$ for some context tree $C[x]$, and some $j, k \in \{0, 1\}$. It is easy to give an NTA $\mathcal{N} = \langle Q, \Delta, F\rangle$ for this relation, where $Q = \{q, q_{cpy}, q_{idm}\}$ with a copying (resp. idempotent) state q_{cpy} (resp. q_{idm}), with $\{q\}$-local depth 1. For example, for each $i = 0, 1$, add to Δ the following transitions: $\xrightarrow{(i,i)} q_{cpy}$, $\xrightarrow{(\bot,i)} q$, $(q_{cpy}, q_{cpy}) \xrightarrow{(\hat{i},\hat{i})} q_{cpy}$, $(q, q) \xrightarrow{(i,\hat{i})} q_{idm}$, $(q_{cpy}, q_{idm}) \xrightarrow{(\hat{i},\hat{i})} q_{idm}$, and $(q_{idm}, q_{cpy}) \xrightarrow{(\hat{i},\hat{i})} q_{idm}$. The unique final state is q_{idm}.

Bounded local depth accelerations Given a positive integer k, the *k-local depth acceleration* $\lfloor\mathcal{R}\rfloor_{k,Q'}$ of the relation $\mathcal{R}$ with respect to the state-set $Q' \subseteq Q$ is the relation containing all pairs $(T, T') \in \mathcal{R}^*$ of Q'-local depth at most k. Define the identity relation $\mathcal{R}_{id} = \{(T, T) : T \in \textsc{Tree}(\Sigma)\}$. Observe that $\mathcal{R} \cup \mathcal{R}_{id} \subseteq \lfloor\mathcal{R}\rfloor_{k,Q'} \subseteq \mathcal{R}^*$ for each positive integer k. In the case when $\mathcal{R}$ has finite Q'-local depth, we have $\lfloor\mathcal{R}\rfloor_{k,Q'} = \mathcal{R}^*$ for some k. Again, whenever Q' is clear from the context, we will simply write $\lfloor\mathcal{R}\rfloor_k$.

▶ **Theorem 2.** *Given a copying state q_{cpy} and an idempotent state q_{idm} of $\mathcal{A}$, the k-local depth acceleration $\lfloor\mathcal{R}\rfloor_k := \lfloor\mathcal{R}\rfloor_{k,Q\setminus\{q_{cpy},q_{idm}\}}$ of $\mathcal{R}$ is tree-automatic. Furthermore, an NTA presenting $\lfloor\mathcal{R}\rfloor_k$ can be computed in time polynomial in $\|\mathcal{A}\|$ and exponential in k.*

The above theorem is proven by a reduction to the computation of k-local depth acceleration of a structure-preserving relation with *two* special copying states; the latter can be proven by a simple adaptation of the proof for the subcase shown in [4] with *one* special copying state q_{cpy}. [In the technical report of [4], an extension is given with several special copying states but has a different condition from the above theorem.] This reduction itself is achieved by "reserving" enough space in the resulting structure-preserving relation (by means of padding) to simulate computation paths in the original system. Details for the reduction and adaptation of the proof of [4] are given in the full version.

Using Theorem 2, we can design a simple semi-algorithm (call it *bounded local depth acceleration semi-algorithm)* for computing an NTA $\mathcal{A}^*$ for the reachability relation $\mathcal{R}^*$ of the tree-automatic relation $\mathcal{R}$ presented by $\mathcal{A}$:

1. Let $k := 1$;
2. **Repeat**
3. Construct $\mathcal{A}^*$ presenting $\lfloor\mathcal{R}\rfloor_k$; let $k := k + 1$;
4. **Until** the relation presented by $\mathcal{A}^*$ is transitive

Checking transitivity of a tree-automatic relation is done using Proposition 1 (see the related remark). Since $\mathcal{R} \cup \mathcal{R}_{id} \subseteq \lfloor\mathcal{R}\rfloor_k \subseteq \mathcal{R}^*$ for each positive integer k, termination implies that the output is an NTA presenting $\mathcal{R}^*$.

We will see other examples of (presentations of) tree-automatic relations with finite local depth in the subsequent sections. The reader is also referred to [4] for nice examples of parameterized systems modeled by structure-preserving tree-automatic relations.

Checking (generalized) repeated reachability The problem of generalized repeated reachability for tree-automatic relations is defined as follows: given an NTA presenting a tree-automatic relation $\rightarrow \subseteq \textsc{Tree}(\Sigma) \times \textsc{Tree}(\Sigma)$ and a set $\{\mathcal{N}_i\}_{i=1}^m$ of m NTAs over Σ, compute the set $\textsc{Rec}_{\rightarrow}(\{\mathcal{L}(\mathcal{N}_i)\}_{i=1}^m)$ of trees from which there exists a path visiting each $\mathcal{L}(\mathcal{N}_i)$ infinitely often. As for checking safety, this problem is also undecidable (in fact,

Σ^1_1-complete); see [29]. However, it is known that whenever $\rightarrow^*$ is given as an NTA $\mathcal{A}^*$ as part of the input, the problem becomes decidable:

▶ Proposition 2 ([29, 30]). Given two NTAs $\mathcal{A}$ and $\mathcal{A}^*$ presenting, respectively, a relation $\rightarrow$ and its closure $\rightarrow^*$, and given the set $\{\mathcal{N}\}_{i=1}^m$ of m NTAs over the alphabet Σ, we may compute the set $\textsc{Rec}_{\rightarrow}(\{\mathcal{L}(\mathcal{N}_i)\}_{i=1}^m)$ in time polynomial in $\|\mathcal{A}\| \times \|\mathcal{A}^*\| \times \prod_{i=1}^m \|\mathcal{N}_i\|$.

This proposition (proven in [29]) is a corollary of the result on repeated reachability with $m = 1$, which initially appeared in [30] and was proven using Ramsey theory on infinite graphs. Thus, we may combine this proposition with the bounded local depth acceleration semi-algorithm to obtain a semi-algorithm for generalized repeated reachability, which is guaranteed to terminate for relations $\rightarrow$ of finite local depth. In the sequel, we refer to this semi-algorithm as *bounded local depth repeated reachability semi-algorithm*.

5 PA-processes

PA-processes [27] are a well-known generalization of basic parallel processes (BPP) and pushdown systems with one-state (a.k.a. BPA). They are known to be incomparable to both pushdown systems and Petri nets, and are an important class in the well-known process rewrite systems hierarchy [27].

We follow the presentation of [25] to define PA-processes. Let $\mathcal{V} = \{X, Y, \ldots\}$ be a given finite set whose elements are called *process constants*. A *Process term* over $\mathcal{V}$ is simply a tree in $\textsc{Tree}(\Gamma)$, where $\Gamma = \mathcal{V} \cup \{0, \|, \circ\}$ and $\mathtt{ar}(0) = \mathtt{ar}(X) = 0$, for each $X \in \mathcal{V}$, and $\mathtt{ar}(\|) = \mathtt{ar}(\circ) = 2$. Let $\mathcal{F}_{\mathcal{V}} = \textsc{Tree}(\Gamma)$. Here, 0 denotes a "nil" process, and $\circ$ (resp. $\|$) is the sequential (resp. parallel) composition. Process terms are usually represented using standard term representation of trees where $\|$ and $\circ$ are written in infix notations. A *PA declaration* $\mathcal{P}$ over $\mathcal{V}$ is a set of *PA transition rules* of the form $X \rightarrow t$, where $X \in \mathcal{V}$ and $t \in \mathcal{F}_{\mathcal{V}}$. The PA declaration $\mathcal{P}$ defines a relation $\mathcal{R}_{\mathcal{P}} \subseteq \mathcal{F}_{\mathcal{V}} \times \mathcal{F}_{\mathcal{V}}$, which we will also write as $\rightarrow_{\mathcal{P}}$ (in infix notation), as given by the following inference rules:

$$\frac{t_1 \rightarrow_{\mathcal{P}} t_1'}{t_1 \| t_2 \rightarrow_{\mathcal{P}} t_1' \| t_2} \qquad \frac{t_1 \rightarrow_{\mathcal{P}} t_1'}{t_1 \circ t_2 \rightarrow_{\mathcal{P}} t_1' \circ t_2} \qquad \frac{}{X \rightarrow_{\mathcal{P}} t}\ (X \rightarrow t) \in \mathcal{P}$$

$$\frac{t_2 \rightarrow_{\mathcal{P}} t_2'}{t_1 \| t_2 \rightarrow_{\mathcal{P}} t_1 \| t_2'} \qquad \frac{t_2 \rightarrow_{\mathcal{P}} t_2'}{t_1 \circ t_2 \rightarrow_{\mathcal{P}} t_1 \circ t_2'}\ t_1 \in \text{IsNil}$$

Here $\text{IsNil} = \textsc{Tree}(\{0, \circ, \|\})$ is the set of "terminated" process terms.

Due to the condition $t_1 \in \text{IsNil}$ in the inference rule $\frac{t_2 \rightarrow_{\mathcal{P}} t_2'}{t_1 \circ t_2 \rightarrow_{\mathcal{P}} t_1 \circ t_2'}\ t_1 \in \text{IsNil}$, an NTA presenting $\mathcal{R}_{\mathcal{P}}$ will need two special copying states (one to cater for copying trees in IsNil, and another for copying trees that are not in IsNil), which is prohibited in our framework[1]. In order to capture PA within our framework, we will have to modify the above semantics slightly. Extend the ranked alphabet Σ with a new binary symbol $\circ_R$, i.e., $\Sigma = \Gamma \cup \{\circ_R\}$ with $\mathtt{ar}(\circ_R) = 2$. The PA declaration $\mathcal{P}$ now gives rise to another relation $\mathcal{R}_{\mathcal{P},1} \subseteq \textsc{Tree}(\Sigma) \times \textsc{Tree}(\Sigma)$ (also written $\rightarrow_{\mathcal{P},1}$) obtained from the above inference rules, but replacing the inference rule $\frac{t_2 \rightarrow_{\mathcal{P},1} t_2'}{t_1 \circ t_2 \rightarrow_{\mathcal{P},1} t_1 \circ t_2'}\ t_1 \in \text{IsNil}$ by the following two inference rules: **(R1)** $t_1 \rightarrow_{\mathcal{P},1} t_2$, where $t_1 = (D, \tau_1)$ and $t_2 = (D, \tau_2)$ such that there exists a unique $v \in D$ with $\tau_1(v) = \circ$, $\tau_2(v) = \circ_R$, and $\tau_1(u) = \tau_2(u)$ for each $u \in D \setminus \{v\}$ **(R2)** $\frac{t_2 \rightarrow_{\mathcal{P},1} t_2'}{t_1 \circ_R t_2 \rightarrow_{\mathcal{P},1} t_1 \circ_R t_2'}$.

[1] It is possible to extend our framework to allow several special copying states in a way that subsumes PA, which we refrain from doing for space reasons

Intuitively, trees rooted at a node with label $\circ$ (resp. $\circ_R$) can only modify its left (resp. right) subtrees; the only exception to this is an application of Rule (R1). Note that, once $\circ$ is relabeled to $\circ_R$ by Rule (R1), then it cannot be further relabeled.

We now show that the two semantics are *equivalent* in some precise sense. Let $\text{IsNil}' := \textsc{Tree}(\{0, \circ, \circ_R, \|\})$, i.e., the set IsNil where each $\circ$-labeled node may possibly be relabeled by $\circ_R$. Let $\mathcal{F}'_{\mathcal{V}}$ be the set of trees $T \in \textsc{Tree}(\Sigma)$ such that (i) the left subtree of each $\circ_R$-labeled node is in IsNil', and (ii) the left subtree of each $\circ$-labeled node is *not* in IsNil' (i.e. some process constant must be found at a leaf). Observe that we can easily construct an NTA of size $O(|\Sigma|)$ recognizing $\mathcal{F}'_{\mathcal{V}}$. Let $\mathcal{R}_{\leftarrow} \subseteq \mathcal{F}'_{\mathcal{V}} \times \mathcal{F}_{\mathcal{V}}$ consists of tuples (T, T') where T' is obtained by relabeling each $\circ_R$-labeled node in T by $\circ$. Notice that this is also a bijective function (by virtue of the conditions (i) and (ii) above), and additionally a tree-automatic relation which can be presented by an NTA $\mathcal{A}_{\leftarrow}$ of size $O(|\Sigma|)$.

▶ Proposition 3. The following statements are equivalent for all trees T_1, T'_1, T_2, T'_2 with $(T'_1, T_1), (T'_2, T_2) \in \mathcal{R}_{\leftarrow}$:

1. $(T_1, T_2) \in \mathcal{R}^*_{\mathcal{P}}$
2. $(T'_1, T'_2) \in \mathcal{R}^*_{\mathcal{P},1}$

Hence, given a synchronous automaton $\mathcal{A}^*_{\mathcal{P},1}$ presenting $\mathcal{R}^*_{\mathcal{P},1}$, we may compute a synchronous automaton $\mathcal{A}^*_{\mathcal{P}}$ presenting $\mathcal{R}^*_{\mathcal{P}}$ in time polynomial in $\|\mathcal{A}^*_{\mathcal{P},1}\|$.

The proof can be found in the full version.

The relation $\mathcal{R}_{\mathcal{P},1}$ is tree-automatic and can be presented by the NTA $\mathcal{A}_{\mathcal{P},1} = \langle Q, \Delta, F\rangle$ over $\mathbf{\Sigma}_{\perp}$ defined as follows. Suppose that there are n rules in $\mathcal{P}$, where the ith rule is $X_i \to t_i$ where $t_i = (D_i, \tau_i)$. The set Q includes the states q_{cpy}, q_{idm}, $q_{\circ}$, and a state $q_{i,v}$ for each $i \in [1, n]$ and $v \in D_i$. The set F of final states is $\{q_{idm}, q_{\circ}\} \cup \{q_{i,\epsilon} : i \in [1, n]\}$. For each $(a, a) \in \mathbf{\Sigma}$ with $\mathtt{ar}((a, a)) = k$, we add the transition $(q_1, \ldots, q_k) \xrightarrow{(a,a)} q_{cpy}$ to Δ, where $q_1 = \cdots = q_k = q_{cpy}$. Add the transitions $(q_{cpy}, q_F) \xrightarrow{(\|,\|)} q_{idm}$, $(q_F, q_{cpy}) \xrightarrow{(\|,\|)} q_{idm}$, $(q_{cpy}, q_F) \xrightarrow{(\circ_R,\circ_R)} q_{idm}$, and $(q_F, q_{cpy}) \xrightarrow{(\circ,\circ)} q_{idm}$, for each final state $q_F \in F$, to Δ. Add the transition $(q_{cpy}, q_{cpy}) \xrightarrow{(\circ,\circ_R)} q_{\circ}$ to Δ. For each $i \in [1, n]$ and $\epsilon \neq v \in D_i$ with $\mathtt{ar}(\tau_i(v)) = k$, add the transition $(q_{i,v0}, \ldots, q_{i,v(k-1)}) \xrightarrow{(\perp,\tau_i(v))} q_{i,v}$ to Δ. For each $i \in [1, n]$ with $\mathtt{ar}(\tau_i(\epsilon)) = k$, add the transition $(q_{i,0}, \ldots, q_{i,k-1}) \xrightarrow{(X_i,\tau_i(\epsilon))} q_{i,\epsilon}$ to Δ. It is easy to see that $\mathcal{A}_{\mathcal{P},1}$ presents the relation $\mathcal{R}_{\mathcal{P},1}$ and has size $O(\|\mathcal{P}\|)$. Furthermore, it is easy to see that q_{cpy} (resp. q_{idm}) is a copying (resp. idempotent) state. In the sequel, we shall not distinguish $\mathcal{P}$ from $\mathcal{A}_{\mathcal{P},1}$.

▶ **Theorem 3.** *The local depth of a PA $\mathcal{A}_{\mathcal{P},1}$ is at most $O(\|\mathcal{P}\|)$. Therefore, bounded local depth acceleration and repeated reachability semi-algorithms terminate on the class of PA.*

The proof idea is as follows. Suppose $T_1 \to^*_{\mathcal{P},1} T_2$. If a rule $X \to t$ with $|t| > 1$ is applied to an X-labeled node v along a witnessing path, the label of v will next be in $\{\|, \circ, \circ_R\}$. So, the only rule that may modify v in the rest of the path is (R1), which can only be applied at most once at any node. The only problematic case is when we apply rule of the form $X \to Y$, where $X, Y \in \mathcal{V}$. To account for this case, observe that once such rules are applied for more than $|\mathcal{V}|$ times at node v, we have detected redundant applications of PA rules and may remove them to ensure that such rules are applied at most $|\mathcal{V}|$ times at node v. The full proof is given in the full version.

6 Ground tree rewrite systems

Ground-tree rewrite systems (e.g. [24]) are an extension of prefix-rewrite systems (equivalently, pushdown systems), where rewrite rules are given as pairs of trees over some ranked alphabet

and they rewrite subtrees (instead of word prefixes). They are incomparable to PA-processes up to strong bisimulations, but subsume PA-processes up to branching bisimulations [20]. Although local model checking (e.g. fragments of LTL) of PA-processes can be reduced to the same problem over GTRS in polynomial time [20], the same is not known for global model checking (e.g. computing $pre^*(S)$ or reachability relations).

A *ground-tree rewrite systems (GTRS)* [24] over the ranked alphabet Σ is a set $\mathcal{P}$ of rules of the form $t_1 \to t_2$, where $t_1, t_2 \in \textsc{Tree}(\Sigma)$. A *pushdown system (PDS)* is a GTRS where Σ contains no label a with $\mathtt{ar}(a) > 1$. We write $\mathrm{Dom}(\mathcal{P})$ for the set of trees t on the l.h.s. or r.h.s. of some rule in $\mathcal{P}$. The GTRS $\mathcal{P}$ defines a binary relation $\mathcal{R}_\mathcal{P} \subseteq \textsc{Tree}(\Sigma) \times \textsc{Tree}(\Sigma)$ as follows: given two trees $T_1, T_2 \in \textsc{Tree}(\Sigma)$, we have $(T_1, T_2) \in \mathcal{R}_\mathcal{P}$ iff there exists a context $C[x]$ such that $T_1 = C[t_1]$ and $T_2 = C[t_2]$ for some rule $(t_1 \to t_2) \in \mathcal{P}$. For convenience, we will also denote $(T_1, T_2) \in \mathcal{R}_\mathcal{P}$ by $T_1 \to_\mathcal{P} T_2$.

We now give an obvious presentation of the relation $\mathcal{R}_\mathcal{P}$ as a synchronous automaton $\mathcal{A}_\mathcal{P} = \langle Q, \Delta, F\rangle$ over the alphabet $\mathbf{\Sigma}_\perp$. In the sequel, when understood from the context, we will confuse GTRS and their presentations. Suppose that there are n rules in $\mathcal{P}$, where the ith rule is $r_i : t_i \to t_i'$ with $t_i = (D_i, \tau_i)$ and $t_i' = (D_i', \tau_i')$. We extend the function τ_i (resp. τ_i') to $\mathbb{N}^*$ so that whenever $v \notin D_i$ (resp. $v \notin D_i'$) we have $\tau_i(v) = \perp$ (resp. $\tau_i'(v) = \perp$). Then, for each $i \in [1, n]$ and each node $v \in D_i \cup D_i'$, we add a state $q_{i,v}$ to Q. We also add two states q_{idm} and q_{cpy} to Q. We now define the transition relation Δ. For each $a \in \Sigma$ with $\mathtt{ar}(a) = k$, we add the transition $(q_1, \ldots, q_k) \xrightarrow{(a,a)} q_{cpy}$, where $q_1 = \cdots q_k = q_{cpy}$. Such a transition will be used in the subtrees that are not rewritten by $\mathcal{P}$. Likewise, for each $i \in [1, n]$ and node $v \in D_i \cup D_i'$ with children $v0, \ldots, v(k-1)$ (so $vk \notin D_i \cup D_i'$), add the transition $(q_{i,v0}, \ldots, q_{i,v(k-1)}) \xrightarrow{(\tau_i(v), \tau_i'(v))} q_{i,v}$ to Δ. Such a transition will occur in the subtree that is rewritten by $\mathcal{P}$. Finally, for each $a \in \Sigma$ with arity $\mathtt{ar}(a) = k$, we add each possible transition of the form $(q_1, \ldots, q_k) \xrightarrow{(a,a)} q_{idm}$, where: (1) for a unique index $j \in [1, k]$ we have $q_j = q_{idm}$ or $q_j = q_{i,\epsilon}$ for some $i \in [1, n]$, and (2) for each other index $j' \neq j$, we have $q_{j'} = q_{cpy}$. The set F of final states consists of q_{idm} and $q_{i,\epsilon}$ for each $i \in [1, n]$. Note that $\|\mathcal{A}_\mathcal{P}\| = O(\|\mathcal{P}\|)$. Observe that q_{cpy} is a copying state, and q_{idm} an idempotent state. Moreover, it is easy to see that $\mathcal{A}_\mathcal{P}$ presents the relation $\mathcal{R}_\mathcal{P}$.

▶ **Theorem 4.** *The local depth of a GTRS $\mathcal{A}_\mathcal{P}$ is bounded exponentially by $\|\mathcal{A}_\mathcal{P}\|$. So, bounded local depth acceleration and repeated reachability semi-algorithms terminate on GTRS.*

The proof of this theorem is much more involved than the proof for the case of PA-processes. This is because in GTRS (and PDS) a subtree may be rewritten (expanded and shrunk) ad infinitum, unlike for PA-processes where applying any rewrite rule of the form $X \to t$ (with t has more than one node) at a node v guarantees that v can only be touched once more with rule (R1). The proof idea for the theorem is as follows. We will first prove a path factorization lemma for GTRS, which allows us to consider "simpler" paths. Intuitively, simpler paths are paths that visit $\mathrm{Dom}(\mathcal{P})$, i.e., of the form $\pi : T \to^* t \to^* T'$, where $t \in \mathrm{Dom}(\mathcal{P})$. We then prove upper bounds on the local depths of such paths, which will transfer to local depths for $\mathcal{P}$. We will spend the rest of this section to sketch the proof of Theorem 4. We first need the following path factorization lemma for GTRS.

▶ **Lemma 5.** *For two arbitrary trees $T_1, T_2 \in \textsc{Tree}(\Sigma)$, it is the case that $T_1 \to_\mathcal{P}^* T_2$ iff there exists a context tree $C[x_1, \ldots, x_n]$ for some $n \in \mathbb{N}$ such that*

1. *$T_1 = C[t_1, \ldots, t_n]$ for some trees $t_1, \ldots, t_n \in \textsc{Tree}(\Sigma)$.*
2. *$T_2 = C[t_1', \ldots, t_n']$ for some trees $t_1', \ldots, t_n' \in \textsc{Tree}(\Sigma)$.*
3. *for each $i \in [1, n]$, there exists a tree $t_i'' \in \mathrm{Dom}(\mathcal{P})$ such that $t_i \to_\mathcal{P}^* t_i'' \to_\mathcal{P}^* t_i'$.*

Notice that the condition on the r.h.s. of Lemma 5 allows context trees with no variables (i.e. an element of $\textsc{Tree}(\Sigma)$), which accounts for the case when $T_1 \to_{\mathcal{P}}^* T_2$ within zero step (i.e. $T_1 = T_2$). We give the proof of Lemma 5 in the full version.

Let us now prove Theorem 4. Consider any path $\sigma : T_1 \to^* T_2$ in $\mathcal{R}_{\mathcal{P}}$. Let $\mathcal{A}_{cpy}$ denote the NTA obtained from $\mathcal{A}_{\mathcal{P}}$ by removing all transitions in $\mathcal{A}_{\mathcal{P}}$ of the form $(q_1, \ldots, q_k) \xrightarrow{(a,b)} q$ with $q, q_1, q_2, \ldots, q_k \in Q \setminus \{q_{cpy}, q_{idm}\}$. That is, we remove the transitions which rewrite subtrees. Suppose now that $T_1 \to_{\mathcal{P}}^* T_2$. By Lemma 5, there exists a context tree $C[x_1, \ldots, x_n] = (D_C, \tau_C)$, along with trees $t_i, t_i', t_i'' \in \textsc{Tree}(\Sigma)$ (for each $i \in [1, n]$), satisfying the three stated conditions. Let $u_1, \ldots, u_n$ denote the context nodes of C. Let $\mathcal{C}$ denote the context tree (D_C, τ_C') obtained from C by replacing each label a in non-context nodes by (a, a) and leave variables in context nodes as they are, i.e., $\mathcal{C}$ is over the alphabet $\Sigma_{id} := \{(a, a) : a \in \Sigma\} \cup \{x_1, \ldots, x_n\}$ with $\mathtt{ar}((a, a)) = \mathtt{ar}(a)$ such that, if v is not a context node in C, then $\tau_C'(v) = (\tau_C(v), \tau_C(v))$; otherwise, $\tau_C'(v) = \tau_C(v)$. Then, for each $i \in [1, n]$, we see that $\mathcal{C}[q_1, \ldots, q_n] \in \mathcal{L}(\mathcal{A}_{cpy})$, for every $\bar{q}$ satisfying $q_i \in F$ and $q_j = q_{cpy}$ $(j \neq i)$. We now need the following lemma.

▶ **Lemma 6.** *Given two trees $t \in Dom(\mathcal{P})$ and $T \in \textsc{Tree}(\Sigma)$, if $t \to_{\mathcal{P}}^* T$, then there exists a witnessing run $\pi : t \to_{\mathcal{P}}^* T$ of bounded local depth that is exponentially bounded by $\|\mathcal{P}\|$.*

By considering the new GTRS $\mathcal{P}^{-1}$ obtained by swapping the l.h.s. with the r.h.s. of each rule in $\mathcal{P}$, this lemma also implies that $T \to_{\mathcal{P}}^* t$ has a witnessing run of local depth that is exponentially bounded by $\|\mathcal{P}\|$. By condition (3) in Lemma 5, this lemma gives a path $\pi : T_1 \to_{\mathcal{P}}^* T_2$ in $\mathcal{R}_{\mathcal{P}}$ of local depth that is exponentially bounded by $\|\mathcal{P}\|$, which implies Theorem 4.

It remains to prove Lemma 6. To this end, we will need two ingredients. The first ingredient (Lemma 7) is an upper bound on local depth of a *given* tuple (T_1, T_2) with $T_1 \to_{\mathcal{P}}^* T_2$, which will be given by bounds on the length of the shortest witnessing paths. The second ingredient (Lemma 8) is a lemma for using the first ingredient to derive the maximum of local depths of *all* tuples (T_1, T_2) with $T_1 \to_{\mathcal{P}}^* T_2$.

▶ **Lemma 7.** *Given two trees $T_1, T_2 \in \textsc{Tree}(\Sigma)$, if $T_1 \to_{\mathcal{P}}^* T_2$, then T_1 can reach T_2 within at most k steps, where k is exponentially bounded by $\|\mathcal{P}\| + \|T_1\| + \|T_2\|$.*

▶ **Lemma 8.** *Let $T_1 = (D_1, \tau_1), T_2 = (D_2, \tau_2) \in \textsc{Tree}(\Sigma)$. Then, $T_1 \to_{\mathcal{P}}^* T_2$ implies that there exists a context tree $C[x_1, \ldots, x_n] = (D', \tau')$, for some $n \in \mathbb{N}$, with context nodes $u_1, \ldots, u_n$ such that*

1. *$\{u_1, \ldots, u_n\} \subseteq D_2$, $\{u_1, \ldots, u_n\} \cap D_1 = \emptyset$ and $D' \setminus \{u_1, \ldots, u_n\} = D_1 \cap D_2$, i.e., each u_i is a node of D_2 of the form vj for some leaf node v in D_1 and some $j \in \mathbb{N}$.*
2. *for some trees $t_1, \ldots, t_n \in \textsc{Tree}(\Sigma)$ with $|t_i| \leq \|\mathcal{P}\|$ for each $i \in [1, n]$, we have $T_1 \to_{\mathcal{P}}^* C[t_1, \ldots, t_n]$,*
3. *$T_2 = C[t_1', \ldots, t_n']$ for some trees $t_1', \ldots, t_n' \in \textsc{Tree}(\Sigma)$ with $t_i \to_{\mathcal{P}}^* t_i'$ for each $i \in [1, n]$,*

Lemma 7 is an immediate corollary of [23, Theorem 1 and Lemma 2]. Intuitively, Lemma 8 means that $T_1 \to_{\mathcal{P}}^* T_2$ can go via a tree $C[t_1, \ldots, t_n]$, which is not much bigger than T_1, such that $T_2 = C[t_1', \ldots, t_n']$ and $t_i \to_{\mathcal{P}}^* t_i'$ for each $i = 1, \ldots, n$. That is, if we assume that T_1 is small, the intermediate configuration $C[t_1, \ldots, t_n]$ is also small, and we can use Lemma 7 on the path from T_1 to $C[t_1, \ldots, t_n]$ and apply the same reasoning on the path from t_i to t_i', for each $i \in [1, n]$, since each t_i is small.

Proof of Lemma 8. Let us take the unique context tree $C[x_1, \ldots, x_n] = (D', \tau')$ as defined by Condition 1 of the statement of the lemma (note that this context tree depends only on

T_1 and T_2). Let $t'_1, \ldots, t'_n \in \textsc{Tree}(\Sigma)$ be the unique trees such that $T_2 = C[t'_1, \ldots, t'_n]$. It suffices to show that Condition 2 and 3 are satisfied. Let $\pi : G_0 \to_{\mathcal{P}} \cdots \to_{\mathcal{P}} G_m$ be a path such that $G_0 = T_1$ and $G_m = T_2$. Then, for each $i \in [1, m]$ there exists a context tree $C_i[x]$ with context node $v_i \in \mathbb{N}^*$ and a rule $\alpha_i \to \alpha'_i$ in $\mathcal{P}$ such that $G_{i-1} = C_i[\alpha]$ and $G_i = C_i[\alpha'_i]$. For each $i \in [1, n]$, let n_i be the maximum index $j \in [1, m]$ such that $v_j \prec u_i$ (i.e. v_j is an ancestor of u_i excluding u_i), which must exist since $u_i \notin D_1$. Intuitively, n_i denotes the last point in the path π which modifies ancestors of u_i (excluding itself). This means also that u_i is not a node of G_{n_i-1}, but is a node of G_{n_i} (introduced by the rule $\alpha_{n_i} \to \alpha'_{n_i}$ of $\mathcal{P}$). Let t_i denote the subtree of G_{n_i+1} rooted at the node u_i. We have $|t_i| \leq |\alpha'_i| \leq \|\mathcal{P}\|$. It is also easy to see that $t_i \to^*_{\mathcal{P}} t'_i$. A witnessing path is the sequence $G_{n_i+1}(u_i), \ldots, G_m(u_i)$ (removing duplicates). [Recall that $T(u)$ denotes the subtree of T rooted at u.] Furthermore, the path witnessing $T_1 \to^*_{\mathcal{P}} C[t_1, \ldots, t_n]$ can be obtained from π by removing each application of rules that operate on descendants of u_i after G_{n_i+1} (for each $i \in [1, n]$). ◀

To prove Lemma 6, we use Lemma 8 starting with $T_1 = t \in \text{Dom}(\mathcal{P})$. Reaching $C[t_1, \ldots, t_n]$ requires a path of length exponential in $\|\mathcal{P}\|$ by Lemma 7. We now apply the same reasoning again to $t_i \to^*_{\mathcal{P}} t'_i$, for each $i \in [1, n]$, and so build the path from $C[t_1, \ldots, t_n]$ to $C[t'_1, \ldots, t'_n]$ in this fashion. Since each application of this reasoning can only modify descendants of the root of T_1 of at most exponential distance, the constructed path witnessing $t \to^*_{\mathcal{P}} T$ has local depth that is exponential in $\|\mathcal{P}\|$, which completes the proof of Lemma 6. A more detailed argument is given in the full version.

▶ Remark. Even for PDS, an exponential upper bound on the local depth of $\mathcal{P}$ is tight. It is well-known that the shortest path from a configuration C to another configuration C' in PDS can be exponential in the size of the PDS $\mathcal{P}$ (more precisely, in the number of states). A witnessing PDS exhibiting this lower bound (e.g. see [21]) simply represents a number with k bits in the stack in binary, and counts from 0 to $2^k - 1$. Since the least significant bit has to be toggled $2^k - 1$ times, the local depth of the PDS also has $2^k - 1$ as the lower bound.

7 Future work

We mention two possible future avenues: (1) Can other interesting classes of infinite systems (e.g. PAD-processes [27] and order-2 collapsible pushdown automata [22]) be captured within bounded local depth acceleration framework? (2) Improve the framework of bounded local depth acceleration techniques so that the semi-algorithm has faster termination guarantee for PA-processes, PDS, and GTRS. One technique that might also help the latter is the simulation-based antichain technique for language inclusion proposed in [2].

Acknowledgements Many thanks to P. Abdulla for answering questions about [4], M. Hague and anonymous reviewers for their comments, and EPSRC (H026878) for the support.

References

1 P. A. Abdulla, K. Cerans, B. Jonsson, and Y. K. Tsay. General decidability theorems for infinite-state systems. In *LICS*, pages 313–321, 1996.
2 P. A. Abdulla, Y.-F. Chen, L. Holík, R. Mayr, and T. Vojnar. When simulation meets antichains. In *TACAS*, pages 158–174, 2010.
3 P. A. Abdulla, N. B. Henda, G. Delzanno, F. Haziza, and A. Rezine. Parameterized tree systems. In *FORTE*, pages 69–83, 2008.
4 P. A. Abdulla, B. Jonsson, P. Mahata, and J. d'Orso. Regular tree model checking. In *CAV*, pages 555–568, 2002.

5 P. A. Abdulla, B. Jonsson, M. Nilsson, and M. Saksena. A survey of regular model checking. In *CONCUR*, pages 35–48, 2004.

6 P. A. Abdulla, A. Legay, J. d'Orso, and A. Rezine. Tree regular model checking: A simulation-based approach. *J. Log. Algebr. Program.*, 69(1-2):93–121, 2006.

7 S. Bardin, A. Finkel, J. Leroux, and Ph. Schnoebelen. Flat acceleration in symbolic model checking. In *ATVA*, pages 474–488, 2005.

8 A. Blumensath and E. Grädel. Finite presentations of infinite structures: Automata and interpretations. *Theory Comput. Syst.*, 37(6):641–674, 2004.

9 B. Boigelot. *Symbolic Methods for Exploring Infinite State Spaces.* PhD thesis, Université de Liège, 1999.

10 B. Boigelot and F. Herbreteau. The power of hybrid acceleration. In *CAV*, pages 438–451, 2006.

11 B. Boigelot, A. Legay, and P. Wolper. Iterating transducers in the large (extended abstract). In *CAV*, pages 223–235, 2003.

12 A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: Application to model-checking. In *CONCUR*, pages 135–150, 1997.

13 A. Bouajjani, P. Habermehl, A. Rogalewicz, and T. Vojnar. Abstract regular (tree) model checking. *STTT*, 14(2):167–191, 2012.

14 A. Bouajjani and T. Touili. Widening techniques for regular tree model checking. To appear in *Int J Softw Tools Technol Transfer*, (Published online: Aug. 2011).

15 D. Caucal. On the regular structure of prefix rewriting. In *CAAP*, pages 87–102, 1990.

16 H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications, 2007.

17 H. Comon and Y. Jurski. Multiple counters automata, safety analysis and Presburger arithmetic. In *CAV*, pages 268–279, 1998.

18 D. Dams, Y. Lakhnech, and M. Steffen. Iterating transducers. *J. Log. Algebr. Program.*, 52-53:109–127, 2002.

19 M. Dauchet and S. Tison. The theory of ground rewrite systems is decidable. In *LICS*, pages 242–248, 1990.

20 S. Göller and A. W. Lin. Refining the process rewrite systems hierarchy via ground tree rewrite systems. In *CONCUR*, pages 543–558, 2011.

21 M. Hague and A. W. Lin. Model checking recursive programs with numeric data types. In *CAV*, pages 743–759, 2011.

22 A. Kartzow. Collapsible pushdown graphs of level 2 are tree-automatic. In *STACS*, pages 501–512, 2010.

23 A. W. Lin. Weakly synchronized ground tree rewriting. In *MFCS*, pages 630–642, 2012.

24 C. Löding. *Infinite Graphs Generated by Tree Rewriting.* PhD thesis, RWTH Aachen, 2003.

25 D. Lugiez and Ph. Schnoebelen. The regular viewpoint on PA-processes. *Theor. Comput. Sci.*, 274(1-2):89–115, 2002.

26 D. Lugiez and Ph. Schnoebelen. Decidable first-order transition logics for PA-processes. *Inf. Comput.*, 203(1):75–113, 2005.

27 R. Mayr. *Decidability and Complexity of Model Checking Problems for Infinite-State Systems.* PhD thesis, TU-Munich, 1998.

28 M. Nilsson. *Regular Model Checking.* PhD thesis, Uppsala Universitet, 2005.

29 A. W. To. *Model Checking Infinite-State Systems: Generic and Specific Approaches.* PhD thesis, LFCS, School of Informatics, University of Edinburgh, 2010.

30 A. W. To and Leonid Libkin. Recurrent reachability analysis in regular model checking. In *LPAR*, pages 198–213, 2008.

31 I. Walukiewicz. Pushdown processes: Games and model-checking. *Inf. Comput.*, 164(2):234–263, 2001.