The Complexity of Verifying Ground Tree Rewrite Systems

Stefan Göller[1] and Anthony Widjaja To[2]

[1] Universität Bremen, Institut für Informatik, Germany
[2] Oxford University Computing Laboratory, UK

Abstract. Ground tree rewrite systems (GTRS) are an extension of pushdown systems with the ability to spawn new subthreads that are hierarchically structured. In this paper, we study the following problems over GTRS: (1) model checking EF-logic, (2) weak bisimilarity checking against finite systems, and (3) strong bisimilarity checking against finite systems. While they are all known to be decidable, we show that problems (1) and (2) have nonelementary complexity, whereas problem (3) is shown to be in coNEXP by finding a syntactic fragment of EF whose model checking complexity is complete for $\mathsf{P}^{\mathsf{NEXP}}$. The same problems are studied over a more general but decidable extension of GTRS called regular GTRS (RGTRS), where regular rewriting is allowed. Over RGTRS we show that all three problems have nonelementary complexity. We also apply our techniques to problems over PA-processes, a well-known class of infinite systems in Mayr's PRS (Process Rewrite Systems) hierarchy. For example, strong bisimilarity checking of PA-processes against finite systems is shown to be in coNEXP, yielding a first elementary upper bound for this problem.

1 Introduction

Pushdown systems (PDS) are natural abstractions of sequential programs with unbounded recursions. The problems of verifying pushdown systems have hitherto been well-studied (cf. [2, 25, 26]). In addition to recursions, concurrency is indisputably another feature that commonly arises in real-world programs. Multithreading is often introduced by design, e.g., to achieve speedup or to make programming more convenient.

Ground tree rewrite systems (GTRS) (cf. [4, 8, 9, 14]), which were also studied under the name ground term rewrite systems in the rewriting community, are an extension of PDS with the ability to spawn new subthreads that are hierarchically structured, which in turn may terminate and return some values to their parents. While the rules of a PDS rewrite a prefix of a given word, the rules of a GTRS rewrite a *subtree* of a given tree. The decidability status for many standard verification (i.e. model checking and equivalence checking) problems over GTRS is well-known. For example, reachability, recurrent reachability, and fair termination are decidable (cf. [4, 7, 14, 22]). Moreover, model checking first-order logic with reachability predicates is decidable [9], which implies the decidability of model checking the common fragment of Computation Tree Logic (CTL) known as EF-logic. In turn, by a reduction to EF-logic (as shown in [11]), we also obtain the decidability of the problems of weak/strong bisimilarity checking against finite systems over GTRS. In fact, most of these decidability results are known to hold for regular GTRS (RGTRS) [14], which are a well-known extension of GTRS with more general rewrite rules that are given by tree automata[1]. On the negative side, it is known that most linear-time and branching-time logics — such as LTL and CTL — have undecidable model checking problems over GTRS (cf. [14, 24]). This is in stark contrast with pushdown systems, over which model checking monadic second-order logic is still decidable [17].

The precise complexities of some verification problems over GTRS and extensions thereof are also known. For example, reachability and recurrent reachability are polynomial time solvable even for RGTRS (cf. [14]). In contrast, model checking first-order logic with reachability predicates over GTRS has nonelementary complexity [21] since the infinite binary tree with a descendant can be easily generated by a fixed GTRS (in fact, by a one-state pushdown system). The precise complexity of EF-logic model checking over GTRS (and extensions thereof) was stated as an open question in [14]. The best known upper bound is currently nonelementary, while the best known lower bound is only PSPACE (which holds already for pushdown systems [2, 26]). Likewise, for the problems of strong/weak bisimilarity checking against finite

[1] This extension is analogous to how *prefix-recognizable systems* [5] extend PDS.

systems over GTRS (and RGTRS), the best known upper bound is nonelementary, while the best known lower bound is only PSPACE (which holds already for pushdown systems [13, 20]). Interestingly, the same nonelementary gaps are also currently present (cf. [20]) when these three problems are considered over similar infinite-state models like PA and PAD processes, which are well-known classes of infinite systems in Mayr's PRS (Process Rewrite Systems) hierarchy (cf. [16]). Note that these three problems are only PSPACE-complete over pushdown systems (cf. [2, 13, 20, 26]).

Contributions. We investigate the following verification problems over GTRS (and the extension RGTRS): (1) model checking EF-logic, (2) weak bisimilarity checking against finite systems, and (3) strong bisimilarity checking against finite systems. Such problems are arguably the most basic verification problems over infinite-state systems, especially in the concurrent setting (cf. [16]). Our main contribution is to pinpoint the complexity of these problems.

The starting point of our paper is a proof that EF-logic model checking over GTRS has a nonelementary complexity, already when considering EF formulas with two occurrences of EF operators that are nested. This shows that the existing automata-based algorithms for the problem (cf. [7, 9, 14]) are in some sense optimal answering Löding's open question [14]. The lower bound proof is achieved by an exponential reduction from the decidable first-order theory over finite words, which is well-known to have a nonelementary complexity [21]. With the same arguments one can also show that Hennesy-Milner logic (i.e. the fragment with no EF operators) suffice to show the nonelementary lower bound over the more general class of RGTRS.

We then proceed to look at the fragment EF_1 of EF-logic consisting of formulas with EF operator nesting depth at most one (i.e. in the parse tree of the formula, every branch has at most one occurence of the operator EF). This fragment is interesting for two reasons. Firstly, as mentioned above, our proof of the nonelementary lower bound for problem (1) over GTRS requires precisely two nested occurrences of EF operators. Secondly, there is a polynomial time reduction from problem (3) to the problem of model checking EF_1 formulas over GTRS if the formulas are represented as DAGs, which are single-exponentially more succinct than the standard tree representation of formulas. Our result is that the problem of model checking EF_1 over GTRS is $\mathsf{P}^{\mathsf{NEXP}}$-complete (i.e. within the second level of the exponential hierarchy). This result cannot be obtained by simply applying the existing automata-based algorithms for EF-logic model checking (cf. [7, 9, 14]). Moreover, a further analysis of our proof shows that problem (3) is solvable in coNEXP. This has substantially closed the nonelementary gap with the best known lower bound for the problem, which is PSPACE. In fact, these proof techniques can be easily applied to derive better upper/lower bounds for verification problems PA-processes: (1) strong bisimilarity checking of PA-processes against finite systems is solvable in coNEXP giving the first elementary upper bound for this problem (cf. [20]), and (2) model checking EF-logic over PA-processes is $\mathsf{P}^{\mathsf{NEXP}}$-hard improving the best known lower bound of PSPACE for the problem (cf. [16]).

We then consider two natural extensions of the problem of checking strong bisimilarity against finite systems over GTRS: (i) checking strong bisimilarity against finite systems over the more general class RGTRS, and (ii) checking weak bisimilarity against finite systems over GTRS. In contrast, it turns out that both of these extensions have a nonelementary complexity. These results are the most technically involved in this paper with the lower bound proof of problem (ii) building upon the lower bound proof of problem (i). Bisimilarity checking has a standard interpretation as a game between two players (cf. [13]) called Attacker (who aims to show non-bisimilarity) and Defender (who aims to show bisimilarity). The difficulty of proving a complexity lower bound for bisimilarity checking problem is due to the asymmetry between the power of Attacker and Defender (Attacker is often more powerful) in such games. Known lower bound techniques for bisimilarity checking, a.k.a. "Defender's forcing", are often implemented by the help of finite control unit, which many infinite-state models have (e.g. PDS and Petri nets). The difficulty of providing Defender's forcing techniques in the absence of finite control unit is witnessed by the plethora of open problems concerning decidability/complexity of equivalence checking over infinite-state models like PA and PAD processes (cf. [20]). The lack of global finite control unit often means that Defender does not have an *immediate* way of punishing Attacker (i.e. forcing him not to do something bad). In the case of GTRS or RGTRS, this means that at any given time Attacker may replace any of (potentially unbounded number of) the subtrees that are present in the current configuration (i.e. a tree). In this paper, we provide

Model checking	GTRS	RGTRS
EF_0	PSPACE-complete	
EF_1	$\mathsf{P}^{\mathsf{NEXP}}$-complete	
$\mathsf{EF}_{k,\ k\geq 2}$	NONELEMENTARY	

Bisimilarity against finite systems	GTRS	RGTRS
$\sim$	PSPACE $\cdots$ coNEXP	
$\approx$	NONELEMENTARY	

Table 1. Overview of our results

the first methods for implementing Defender's forcing technique over infinite-state models that lack finite-control unit resulting in strong (i.e. nonelementary) lower bounds.

Our results for (R)GTRS are summarized in Table 1.

Related work. Other related problems over GTRS have been shown to be decidable: confluence (cf. [8]), model checking fragments of LTL [23], and generalized recurrent reachability [24].

Several other extensions of PDS with multithreading capabilities have been considered in [3, 12, 16, 19]. Among these extensions, the class of process rewrite systems [16], which generalize both Petri nets and pushdown systems by providing hierarchical structures to threads, seem to have tight connections with GTRS. It is interesting to note that there is also a nonelementary gap between the best known upper/lower bounds for EF-logic model checking and strong/weak bisimilarity checking against finite systems over two proper subclasses of process rewrite systems known as PA-processes and PAD-processes (cf. [15, 16, 20]). PA-processes have also been shown to be a good abstraction of parallel programs for the purpose of interprocedural data-flow analysis (cf. [10]).

Organization. Section 2 contains preliminaries. In Section 3, we analyze the complexity of model checking EF-logic (and its fragments). We prove a nonelementary lower bound for strong bisimilarity checking against finite systems in Section 4. Results concerning weak bisimilarity checking against finite systems for GTRS are discussed in Section 5. Section 5 depends on Section 4 which in turn depends on Section 3.1.

Finally, in Section 6 we show applications of our techniques to problems over PA-processes.

2 Preliminaries

2.1 General notations

Let X be a set and let $R, S \subseteq X \times X$ be a binary relation. By R^+ (resp. R^*) we denote the *transitive (resp. reflexive transitive) closure* of R and by $R \circ S = \{(x, y) \mid \exists z \in X : (x, z) \in R \text{ and } (z, y) \in S\}$ we denote the *composition* of R and S. By $\mathbb{Z}$ we denote the *integers* and by $\mathbb{N}$ we denote the set of *non-negative* integers. For each $i, j \in \mathbb{Z}$, we denote by $[i, j]$ the interval $\{i, i+1, \ldots, j-1, j\}$. Let $f : X \to Y$ be a partial function. Then we denote by $\mathrm{dom}(f) = \{x \in X \mid \exists y \in Y : f(x) = y\}$ the *domain* and by $\mathrm{ran}(f) = \{y \in Y \mid \exists x \in X : f(x) = y\}$ the *range* of f. Throughout the rest of this paper, we fix a countable set of *action labels* Act.

2.2 Complexity theory

We assume familiarity with Turing machines (TMs), alternating Turing machines (ATMs) and the standard complexity classes. In this paper we the following complexity classes are of relevance:

- P: Deterministic polynomial time.
- PSPACE: (Non)deterministic polynomial space.
- AP: Alternating polynomial time. It is well-known that $\mathsf{AP} = \mathsf{PSPACE}$.
- NEXP: Nondeterministic exponential time.
- coNEXP: The complements of all languages in NEXP.
- $\mathsf{P}^{\mathsf{NEXP}}$: The problems solvable in deterministic polynomial time with oracle access to some language in NEXP. This class is contained in the second level of the exponential hierarchy which is in turn contained in $\mathsf{EXPSPACE}$. By a recent result of Allender, Koucky, Ronneburger and Roy [1] it is known that $\mathsf{P}^{\mathsf{NEXP}} = \mathsf{PSPACE}^{\mathsf{NEXP}}$.
- k-EXP: Deterministic k-fold exponential time.
- $\mathsf{ELEMENTARY}$: $\bigcup_k k\text{-}\mathsf{EXP}$

Unless stated otherwise, *reductions* are always polynomial time many-to-one reductions. Let us define the tower function $\mathsf{TOWER} : \mathbb{N} \to \mathbb{N}$ as $\mathsf{TOWER}(0) = 1$ and $\mathsf{TOWER}(n+1) = 2^{\mathsf{TOWER}(n)}$ for each $n \in \mathbb{N}$. For a thorough introduction to complexity theory, we refer the reader to the textbook [18].

2.3 First-order logic over words

A (binary) *word* is a finite sequence $a_1 a_2 \dots a_n$, where $a_i \in \{0,1\}$ for each $i \in [1,n]$. We identify such a word with the logical structure

$$\overline{w} \quad = \quad (U, P_0, P_1, <)$$

with universe $U = [1,n]$, unary predicates $P_a = \{i \in [1,n] \mid a_i = a\}$ for each $a \in \{0,1\}$ and the binary predicate $<$.

Formulas of first-order logic are given by the following grammar, where x and x' range over a countable set of variables:

$$\varphi \qquad ::= \qquad P_0(x) \mid P_1(x) \mid x < x' \mid \neg\varphi \mid \varphi \wedge \varphi \mid \exists x.\varphi$$

We introduce the usual abbreviations $\forall x.\varphi$ and $\varphi_1 \wedge \varphi_2$. A *sentence* is a formula that does not contain any free variables. We assume that formulas of first-order logic are given in prenex normal form. Let us define the following decision problem.

FIRST-ORDER THEORY OVER WORDS
INPUT: A first-order sentence $\exists x_1 \forall x_2 \ \dots \ \exists x_{2n-1} \forall x_{2n} \ \varphi(x_1, \dots, x_{2n})$ in prenex normal-form.
QUESTION: Does $\overline{w} \models \varphi$ hold for all $w \in \{0,1\}^*$?

The following result is a well-known result due to Stockmeyer.

Theorem 1 ([21]). *The first-order theory over words is nonelementary.*

2.4 Logic and strong/weak bisimulation equivalence

A *transition system* is a tuple $\mathfrak{S} = (C, \mathbb{A}, \{\xrightarrow{a} \mid a \in \mathbb{A}\})$, where

- C is a set of *configurations*,
- $\mathbb{A} \subseteq \mathsf{Act}$ is some finite set of *actions*, and
- $\xrightarrow{a} \subseteq C \times C$ is a set of transitions for each action $a \in \mathbb{A}$.

Let us fix a subset $\Sigma \subseteq \mathbb{A}$ of $\mathfrak{S}$'s actions. By $\xrightarrow{\Sigma}$ we abbreviate $\bigcup_{a \in \Sigma} \xrightarrow{a}$. We prefer to use the infix notation $c \xrightarrow{a} d$ instead of $(c,d) \in \xrightarrow{a}$. Similar remarks apply to $\xrightarrow{\Sigma}$. We say that $\mathfrak{S}$ is *finite* in case C is finite – we often use the identifier $\mathfrak{F}$ to refer to finite transition systems.

Logic Let us fix a finite set of actions $\mathbb{A} \subseteq \mathsf{Act}$. Formulas of EF-logic are given by the following grammar, where $\Sigma \subseteq \mathbb{A}$:

$$\varphi \quad ::= \quad \mathtt{true} \mid \neg\varphi \mid \varphi \wedge \varphi \mid \langle\Sigma\rangle\varphi \mid \langle\Sigma^*\rangle\varphi$$

We introduce the usual abbreviations $\mathtt{false} = \neg\mathtt{true}$, $\varphi_1 \vee \varphi_2 = \neg(\neg\varphi_1 \wedge \neg\varphi_2)$, $[\Sigma]\varphi = \neg\langle\Sigma\rangle\neg\varphi$, and $[\Sigma^*]\varphi = \neg\langle\Sigma^*\rangle\neg\varphi$. Sometimes we avoid the usage of negation and assume that these abbreviations are used (however, we say this explicitly when abbreviations are used). All of our lower bound proofs can easily be adapted for the more restricted version of EF logic that does not allow to parametrize over subsets Σ (rather only $\Sigma = \mathbb{A}$ is allowed).

For each transition system $\mathfrak{S} = (C, \mathbb{A}, \{\xrightarrow{a} \mid a \in \mathbb{A}\})$ and each EF-formula φ (over $\mathbb{A}$) define the set of all configurations $[\![\varphi]\!]_{\mathfrak{S}} \subseteq C$ that satisfy φ by induction on the structure of φ as follows:

$$\begin{array}{rcl}
[\![\mathtt{true}]\!]_{\mathfrak{S}} & = & C \\
[\![\neg\varphi]\!]_{\mathfrak{S}} & = & C \setminus [\![\varphi]\!]_{\mathfrak{S}} \\
[\![\varphi_1 \wedge \varphi_2]\!]_{\mathfrak{S}} & = & [\![\varphi_1]\!]_{\mathfrak{S}} \cap [\![\varphi_2]\!]_{\mathfrak{S}} \\
[\![\langle\Sigma\rangle\varphi]\!]_{\mathfrak{S}} & = & \{c \in C \mid \exists d \in [\![\varphi]\!]_{\mathfrak{S}} : c \xrightarrow{\Sigma} d\} \\
[\![\langle\Sigma^*\rangle\varphi]\!]_{\mathfrak{S}} & = & \{c \in C \mid \exists d \in [\![\varphi]\!]_{\mathfrak{S}} : c \xrightarrow{\Sigma}{}^{*} d\}
\end{array}$$

We write $(\mathfrak{S}, c) \models \varphi$ whenever $c \in [\![\varphi]\!]_{\mathfrak{S}}$. The EF *nesting depth* $\mathsf{nd}(\varphi)$ of an EF-formula φ is inductively defined as follows:

$$\begin{array}{rcl}
\mathsf{nd}(\mathtt{true}) & = & 0 \\
\mathsf{nd}(\neg\varphi) & = & \mathsf{nd}(\varphi) \\
\mathsf{nd}(\varphi_1 \wedge \varphi_2) & = & \max\{\mathsf{nd}(\varphi_1), \mathsf{nd}(\varphi_2)\} \\
\mathsf{nd}(\langle\Sigma\rangle\varphi) & = & \mathsf{nd}(\varphi) \\
\mathsf{nd}\left(\langle\Sigma^*\rangle\,\varphi\right) & = & \mathsf{nd}(\varphi) + 1
\end{array}$$

For each $i \geq 0$, we denote by EF_i the syntactic fragment of EF restricted to formulas of EF nesting depth at most i. We remark that EF_0 is Henessy-Milner logic (HM).

Strong bisimulation equivalence Let $\mathfrak{S}_1 = (C_1, \mathbb{A}, \{\xrightarrow{a}_1 \mid a \in \mathbb{A}\})$ and $\mathfrak{S}_2 = (C_2, \mathbb{A}, \{\xrightarrow{a}_2 \mid a \in \mathbb{A}\})$ be two transition systems over a common set of actions $\mathbb{A}$. A relation $R \subseteq C_1 \times C_2$ is a *strong bisimulation* if for each $(c_1, c_2) \in R$ the following two conditions hold for each $a \in \mathbb{A}$:

- for every $c_1 \xrightarrow{a}_1 c_1'$ there is some $c_2 \xrightarrow{a}_2 c_2'$ with $(c_1', c_2') \in R$.
- for every $c_2 \xrightarrow{a}_2 c_2'$ there is some $c_1 \xrightarrow{a}_1 c_1'$ with $(c_1', c_2') \in R$.

We say that c_1 is *strongly bisimilar* to c_2 (abbreviated by $c_1 \sim c_2$) whenever there is a strong bisimulation R such that $(c_1, c_2) \in R$.

Weak bisimulation equivalence Let us fix a *silent action* $\tau \notin \mathbb{A}$ and let $\mathbb{A}_\tau = \mathbb{A} \cup \{\tau\}$. Moreover let $\mathfrak{S}_1 = (C_1, \mathbb{A}_\tau, \{\xrightarrow{a}_1 \mid a \in \mathbb{A}_\tau\})$ and $\mathfrak{S}_2 = (C_2, \mathbb{A}_\tau, \{\xrightarrow{a}_2 \mid a \in \mathbb{A}_\tau\})$ be two transition systems over the common set of actions $\mathbb{A}_\tau$. We define the binary relations $\overset{\tau}{\Longrightarrow}_i = (\xrightarrow{\tau}_i)^*$ and $\overset{a}{\Longrightarrow}_i = (\xrightarrow{\tau}_i)^* \circ \xrightarrow{a}_i \circ (\xrightarrow{\tau}_i)^*$ for each $a \in \mathbb{A}$ and for each $i \in \{1, 2\}$. A binary relation $R \subseteq C_1 \times C_2$ is a *weak bisimulation* if for each $(c_1, c_2) \in R$ the following two conditions hold for each $a \in \mathbb{A}_\tau$:

- for every $c_1 \xrightarrow{a}_1 c_1'$ there is some $c_2 \overset{a}{\Longrightarrow}_2 c_2'$ with $(c_1', c_2') \in R$.
- for every $c_2 \xrightarrow{a}_2 c_2'$ there is some $c_1 \overset{a}{\Longrightarrow}_1 c_1'$ with $(c_1', c_2') \in R$.

We say that c_1 is *weakly bisimilar* to c_2 (abbreviated by $c_1 \approx c_2$) whenever there is a weak bisimulation R such that $(c_1, c_2) \in R$.

Game characterization of strong/weak bisimulation equivalence Strong (resp. weak) bisimilarity can also be described by simple pebble games between two players: Attacker and Defender. Attacker's goal is to prove that two given processes are *not* strongly (resp. *not* weakly) bisimilar, while Defender tries to prove otherwise. We will refer to Attacker as *him* and to Defender as *her*. In every round of the game, there is a pebble placed on a unique state in each transition system. Attacker then chooses one transition system and moves the pebble from the pebbled state to one of its successors by an action $\xrightarrow{a}$, where $a \in \mathbb{A}$ (resp. $a \in \mathbb{A}_\tau$). Defender must imitate this by moving the pebbled state from the other system to one of its successors by the same action $\xrightarrow{a}$ (resp. $\overset{a}{\Longrightarrow}$). If one player cannot move, then the other player wins. Defender wins every infinite game. Two configurations c_1 and c_2 are strongly/wealy bisimilar (resp. not strongly/weakly-bisimilar) if and only if Defender (resp. Attacker) has a winning strategy on the game with initial pebble configuration (c_1, c_2).

2.5 (Regular) ground tree rewriting systems

Ranked trees Let $\preceq$ denote the prefix order on $\mathbb{N}^*$, i.e. $x \preceq y$ for $x, y \in \mathbb{N}^*$ if there is some $z \in \mathbb{N}^*$ such that $y = xz$, and $x \prec y$ if $x \preceq y$ and $x \neq y$. A *ranked alphabet* is a collection of finite and pairwise disjoint alphabets $A = (A_i)_{i\in[1,k]}$ for some $k \geq 0$. For simplicity we identify A with $\bigcup_{i\in[1,k]} A_i$. A *ranked tree* (over the ranked alphabet A) is a mapping $T : D_T \to A$, where $D_T \subseteq [1,k]^*$ satisfies the following:

- D_T is non-empty, finite and prefix-closed,
- for each $x \in D_T$ with $T(x) \in A_i$ we have $x1, \ldots, xi \in D_T$ and $xj \notin D_T$ for each $j > i$.

We say that D_T is the *domain* of T – we call these elements *nodes*. In case $T(x) \in A_2$ for some node x, then $x1$ is the *left child* and $x2$ the *right child* of x. A *leaf* is a node x with $T(x) \in A_0$. We also refer to $\varepsilon \in D_T$ as the *root* of T. By Trees_A we denote the set of all ranked trees over the ranked alphabet A.

Let T be a ranked tree and let x be a node of T. We define $xD_T = \{xy \in [1,k]^* \mid y \in D_T\}$ and $x^{-1}D_T = \{y \in [1,k]^* \mid xy \in D_T\}$. By $T^{\downarrow x}$ we denote the *subtree of* T with root x, i.e. the tree with domain $D_{T^{\downarrow x}} = x^{-1}D_T$ defined as $T^{\downarrow x}(y) = T(xy)$. Let $S, T \in \mathsf{Trees}_A$ and let x be a node of T. We define $T[x/S]$ to be the tree that is obtained by replacing $T^{\downarrow x}$ in T by S, more formally $D_{T[x/S]} = (D_T \setminus xD_{T^{\downarrow x}}) \cup xD_S$ with

$$T[x/S](y) = \begin{cases} T(y) & \text{if } y \in D_T \setminus xD_{T^{\downarrow x}} \\ S(z) & \text{if } y = xz \text{ with } z \in D_S. \end{cases}$$

Define $|T|$ as the number of nodes in a tree T. We will also use the identifier t to refer to ranked trees, in particular to refer to subtrees.

Regular tree languages A *nondeterministic tree automaton (NTA)* is a tuple $\mathcal{A} = (Q, F, A, \Delta)$, where Q is a finite set of *states*, $F \subseteq Q$ is a set of *final states*, $A = (A_i)_{i\in[1,k]}$ is a ranked alphabet, and $\Delta \subseteq \bigcup_{i\in[1,k]} Q^i \times A_i \times Q$ is the *transition relation*. A *run* of $\mathcal{A}$ on some tree $T \in \mathsf{Trees}_A$ is a mapping $\rho : D_T \to Q$ such that for each $x \in D_T$ with $T(x) \in A_i$ we have $(\rho(x1), \ldots, \rho(xi), t(x), \rho(x)) \in \Delta$. We say ρ is *accepting* if $\rho(\varepsilon) \in F$. By $L(\mathcal{A}) = \{T \in \mathsf{Trees}_A \mid \text{there is an accepting run of } \mathcal{A} \text{ on } T\}$. A set of trees $S \subseteq \mathsf{Trees}_A$ is *regular* if $S = L(\mathcal{A})$ for some NTA $\mathcal{A}$. The *size* of an NTA $\mathcal{A}$ is defined as $|\mathcal{A}| = |Q| + |A| + |\Delta|$. By $\mathsf{Trees}_A^{\leq n} = \{T \in \mathsf{Trees}_A : |T| \leq n\}$ we denote the set of all trees over A with at most n nodes.

Ground tree rewriting systems and regular ground tree rewriting systems A *ground-tree rewriting system (GTRS)* is tuple $\mathcal{R} = (A, \mathbb{A}, R)$, where A is a ranked alphabet, $\mathbb{A} \subseteq \mathsf{Act}$ is finite set of actions, and R is finite set of rewriting rules of the form $S \overset{a}{\hookrightarrow} S'$, where $S, S' \in \mathsf{Trees}_A$ and $a \in \mathbb{A}$. The transition system defined by $\mathcal{R}$ is $\mathfrak{S}(\mathcal{R}) = (\mathsf{Trees}_A, \mathbb{A}, \{\xrightarrow{a} \mid a \in \mathbb{A}\})$, where for each $a \in \mathbb{A}$, we have $T \xrightarrow{a} T'$ if and only if there is some $x \in D_T$ and some rule $S \overset{a}{\hookrightarrow} S'$ in R such that $T^{\downarrow x} = S$ and $T' = T[x/S']$. A *regular ground tree rewriting system (RGTRS)* is a tuple $\mathcal{R} = (A, \mathbb{A}, R)$, where A and $\mathbb{A}$ is the same as for GTRS but where R is finite set of rewriting rules $L \overset{a}{\hookrightarrow} L'$, where L and L' are regular tree languages given

as NTA. The transition system defined by a RGTRS $\mathcal{R}$ is $\mathfrak{S}(\mathcal{R}) = (\mathsf{Trees}_A, \mathbb{A}, \{\xrightarrow{a} | \ a \in \mathbb{A}\})$, where for each $a \in \mathbb{A}$, we have $T \xrightarrow{a} T'$ if and only if there is some $x \in D_T$ and some rule $L \overset{a}{\hookrightarrow} L' \in R$ such that $T^{\downarrow x} = S$ and $T' = T[x/S']$ for some $S \in L$ and some $S' \in L'$.

2.6 Decision problems

In this paper we will be interested in the following decision problems.

EF MODEL CHECKING
INPUT: A (R)GTRS $\mathcal{R} = (A, \mathbb{A}, R)$, $T \in \mathsf{Trees}_A$ and an EF-formula formula φ.
QUESTION: Does $(\mathfrak{S}(\mathcal{R}), T) \models \varphi$ hold?

The analogous question can be asked for the syntactic fragments EF_i of EF. The following upper bound on EF model checking is proven in the PhD thesis of Löding[14].

Theorem 2 ([14]). EF *model checking of (R)GTRS is decidable in time* $\mathsf{TOWER}(O(n))$.

STRONG BISIMILARITY CHECKING AGAINST FINITE SYSTEMS
INPUT: A (R)GTRS $\mathcal{R} = (A, \mathbb{A}, R)$, $T \in \mathsf{Trees}_A$, a finite transition system $\mathfrak{F} = (C, \mathbb{A}, \{\xrightarrow{a} | \ a \in \mathbb{A}\})$ and a configuration $c \in C$.
QUESTION: Does $T \sim c$ hold in the pair of transition systems $(\mathfrak{S}(\mathcal{R}), \mathfrak{F})$?

WEAK BISIMILARITY CHECKING AGAINST FINITE SYSTEMS
INPUT: A (R)GTRS $\mathcal{R} = (A, \mathbb{A}_\tau, R)$, $T \in \mathsf{Trees}_A$, a finite transition system $\mathfrak{F} = (C, \mathbb{A}_\tau, \{\xrightarrow{a} | \ a \in \mathbb{A}_\tau\})$ and a configuration $c \in C$.
QUESTION: Does $T \approx c$ hold in the pair of transition systems $(\mathfrak{S}(\mathcal{R}), \mathfrak{F})$?

The following two propositons state that strong and weak bisimilarity checking can be reduced to model checking (syntactic fragments of) EF logic: Both reductions were proven in [11], but we also refer the reader to [13].

Proposition 3 ([11], [13] Corollary 1). *The following is computable in polynomial time:*
INPUT: An instance $\mathcal{R}, T, \mathfrak{F}, c$ *of strong bisimilarity checking against finite systems.*
OUTPUT: EF_0 *formulas* φ_1, φ_2 *in DAG-representation such that*

$$T \sim c \quad \textit{if and only if} \quad (\mathfrak{S}(\mathcal{R}), T) \models \varphi_1 \wedge [\mathbb{A}^*]\varphi_2.$$

Proposition 4 ([11], [13] Theorem 1). *The following is computable in polynomial time:*
INPUT: An instance $\mathcal{R}, T, \mathfrak{F}, c$ *of weak bisimilarity checking against finite systems.*
OUTPUT: An EF *formula* φ *in DAG-representation such that*

$$T \approx c \quad \textit{if and only if} \quad (\mathfrak{S}(\mathcal{R}), T) \models \varphi.$$

3 Model Checking

3.1 EF_2 (resp. EF_0) is nonelementary over GTRS (resp. over RGTRS)

In this section we prove that model checking EF_2 over GTRS has nonelementary complexity. Very similarly one can prove that EF_0 model checking over RGTRS is nonelementary. For this, let us fix a first-order sentence interpreted over binary words

$$\psi \quad = \quad \exists x_1 \forall x_2 \ \ldots \ \exists x_{2n-1} \forall x_{2n} \ \varphi(x_1, \ldots, \varphi_{2n}).$$

For technical reasons, we will assume without loss of generality that $\overline{w} \not\models \psi$ for each binary word w with $|w| < 2$. Our goal is to compute in exponential time a GTRS $\mathcal{R} = (A, \mathbb{A}, R)$, some initial tree $\mathsf{start} \in \mathsf{Trees}_A \cap A_0$ and an EF_2-formula θ such that

$$\exists w \in \{0,1\}^* : \overline{w} \models \varphi \qquad \text{if and only if} \qquad (\mathfrak{S}(\mathcal{R}), \mathsf{start}) \models \theta.$$

Intuitively speaking, the word w that witnesses $\overline{w} \models \varphi$ will be represented by the yield string of some tree (i.e. the string that one obtains by ordering the leaves in a standard preorder traversal) that is 'downward' reachable from the singleton tree start of the GTRS. After this tree has been guessed, it will be checked via a bottom-up simulation of some NTA whether the yield string of this reached tree indeed satisfies ψ.

We define as action labels $\mathbb{A} = \{a_i \mid i \in [1, 2n]\} \cup \{\mathsf{down}, \mathsf{up}_0, \mathsf{up}_1, \mathsf{up}_2\}$. Let us define the set of proper leaf labels P.

Definition 5 (Proper leaf labels). $P = \left(\left(2^{[1,2n]} \cup \{\bot\}\right) \times \{0,1\}\right)$.

We define the ranked alphabet $A = (A_i)_{i \in \{0,1,2\}}$ of $\mathcal{R}$ as follows, where the set Q will be defined later:

$$\begin{aligned} A_0 &= \{\mathsf{start}\} \cup P \cup Q \\ A_1 &= \{\mathsf{root}\} \\ A_2 &= \{\bigstar\} \end{aligned}$$

The first-component label $\bot$ will not be relevant in this proof, however we will need this label in subsequent proofs.

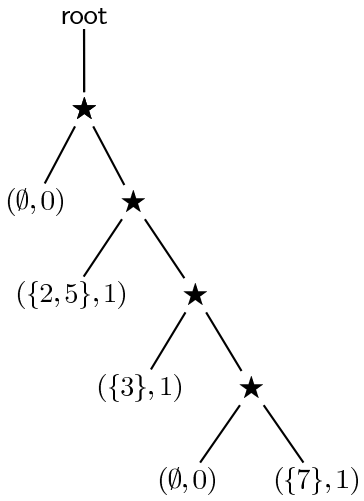
The regular tree language Combs consists of precisely those trees $T \in \mathsf{Trees}_A$ that satisfiy the following conditions:

1. $T^{-1}(\mathsf{root}) = \{\varepsilon\}$, i.e. the (one and) only node of T that is labeled with root is the root of T.
2. For every leaf x of T we have $T(x) \in P$.
3. For each inner node $x \neq \varepsilon$ of T we have
   - $T(x) = \bigstar$.
   - x has a left child that is a leaf.
4. There is at least one inner node x (in particular $T(x) = \bigstar$ holds by Point 3.) that is a child of the root ε.

For each $I \subseteq [1, 2n]$ define the regular tree language Combs_I to consist of precisely those combs $T \in \mathsf{Combs}$ that satisfy the following conditions:

- For every leaf x of T we have $T(x) \in 2^I \times \{0,1\}$.
- For each two distinct leaves x, x' of T with $T(x) = (J, \alpha)$ and $T(x') = (J', \alpha')$ we have $J \cap J' = \emptyset$.
- $I = \bigcup\{J \mid x \text{ is a leaf of } T \text{ and } T(x) = (J, \alpha)\}$

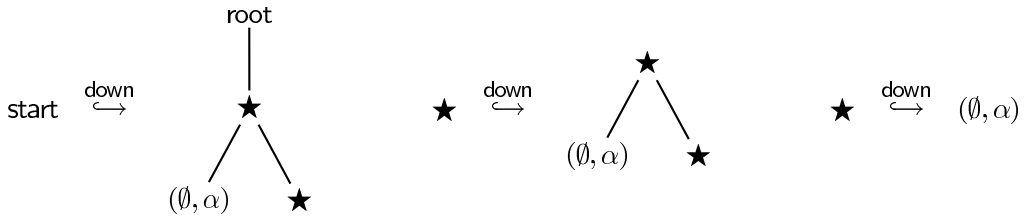
Example 6. Example for a tree in $\mathsf{Combs}_{\{2,3,5,7\}}$



Let us fix a comb $T \in \mathsf{Combs}$. Intuitively, think of the sequence of the *second components* of leaf labels of T (i.e. the second-component projection of the labels of the yield of T) to correspond to a binary word, and moreover, for each leaf x of T, think of the *first component* of $T(x)$ to correspond to the index set of variables $\{x_1, \ldots, x_{2n}\}$ of φ that have been bound to the corresponding position in the word. Hence every comb in Combs_I corresponds to a unique binary word along with a variable valuation with domain I.

By Combs_φ (resp. $\mathsf{Combs}_{\overline{\varphi}}$) we denote the trees from $\mathsf{Combs}_{[1,2n]}$ whose word and variable assignment interpretation satisfies (resp. does not satisfy) φ.

The following rules allow to reach all members of $\mathsf{Combs}_\emptyset$ from the singleton tree start:

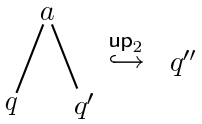


Next, we add the following rules that allow to rewrite the leafs of combs with the intended meaning to assign variable x_i to the leaf via the action a_i:

$$(I, \alpha) \overset{a_i}{\hookrightarrow} (I \cup \{i\}, \alpha) \qquad \text{for each } i \in [1, 2n] \text{ and for each } \alpha \in \{0, 1\}$$

In a next step, we compute in exponential time in $|\psi|$ a nondeterministic tree automaton $\mathcal{A} = (Q, F, A, \Delta)$ that accepts Combs_φ. We add the state set Q to A_0 of our GTRS $\mathcal{R}$. Then we add the following rewriting rules to R:

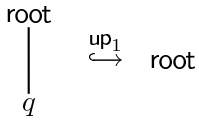
– For every rule $(q, q', a, q'') \in \Delta \cap \left(Q^2 \times A_2 \times Q\right)$ we add the following rewriting rule:



– For every rule $(a, q') \in \Delta \cap (A_0 \times Q)$ we add the following rewriting rule:

$$a \overset{\mathsf{up}_0}{\hookrightarrow} q'$$

– Finally, for each rewriting rule $(q, \mathsf{root}, q') \in \Delta \cap (Q \times \{\mathsf{root}\} \times Q)$ where $q' \in F$ we add the following rewriting rule:



Finally by setting $\theta = \langle \{\mathsf{down}\}^* \rangle \langle a_1 \rangle [a_2] \cdots \langle a_{2n-1} \rangle [a_{2n}] \langle \{\mathsf{up}_0, \mathsf{up}_2\}^* \rangle \langle \{\mathsf{up}_1\} \rangle \mathtt{true}$ one can easily check that

$$\exists w \in \{0, 1\}^* : \overline{w} \models \varphi \qquad \text{if and only if} \qquad (\mathfrak{S}(\mathcal{R}), \mathsf{start}) \models \theta.$$

Theorem 7. *Model checking* EF_2 *over GTRS is nonelementary.*

By adjusting the proof apropriately, one can show analogously the following corollary.

Corollary 8. *Model checking* EF_0 *over RGTRS is nonelementary.*

3.2 Model checking EF_1 over GTRS is in $\mathsf{P}^{\mathsf{NEXP}}$

In this section we prove the following theorem.

Theorem 9. *Model checking an* EF*-formula of the form* $\langle \Sigma^* \rangle \varphi$*, where* φ *is an* EF_0 *formula, is in* NEXP *over GTRS.*

Before proving this theorem, we shall mention the following corollary.

Corollary 10. *Model checking* EF_1 *over GTRS is in* $\mathsf{P}^{\mathsf{NEXP}}$.

Proof. The proof idea is as follows. We first show that this model checking problem is in $\mathsf{AP}^{\mathsf{NEXP}} = \mathsf{PSPACE}^{\mathsf{NEXP}}$. Since it is known that $\mathsf{PSPACE}^{\mathsf{NEXP}} = \mathsf{P}^{\mathsf{NEXP}}$ (cf. Corollary 25 in [1]), the desired upper bound follows.

Let us now show that the model checking problem is in $\mathsf{AP}^{\mathsf{NEXP}}$. We are given a GTRS $\mathcal{R} = (A, \mathbb{A}, R)$, a tree $T_0 \in \mathsf{Trees}_A$, and an EF_1-formula φ_0. We assume that φ_0 does not contain any negations, i.e. by the usage of the abbreviations $\mathtt{false}$, $\vee$,[ ] negations can be removed.

We now describe the behaviour of some alternating Turing machine $\mathcal{M}$ that checks some current tree T and some current subformula φ of φ_0 whether $(\mathfrak{S}(\mathcal{R}), T) \models \varphi$ holds.

For this proof, let us say that a subformula φ of φ_0 is *atomic* if it is of the form $\langle \Sigma^* \rangle \psi$ or $[\Sigma^*]\psi$, where ψ is an HM formula. Suppose that K is the maximum number of nodes appearing in a tree on the right hand side of any rule in $\mathcal{R}$. Our ATM $\mathcal{M}$ works by recursion on the structure the current formula φ. For a base case $\varphi = \mathtt{true}$ (resp. $\varphi = \mathtt{false}$), the machine $\mathcal{M}$ simply accepts (resp. reject). Another base case is when $\varphi = \langle \Sigma^* \rangle \psi$ (resp. $[\Sigma^*]\psi$), in which case $\mathcal{M}$ simply invokes the NEXP oracle from Theorem 9 for evaluating such formulas. We now proceed to inductive cases. On seeing a formula of the form $\varphi = \psi_1 \vee \psi_2$ (resp. $\varphi = \psi_1 \wedge \psi_2$), our ATM $\mathcal{M}$ existentially (resp. universally) guesses to explore ψ_1 or ψ_2. On seeing a formula of the form $\langle \Sigma \rangle \psi$ (resp. $[\Sigma]\psi$), our ATM $\mathcal{M}$ existentially (resp. universally) guesses an action $a \in \Sigma$ and a tree T' such that $T \xrightarrow{a} T'$. Observe that $|T'| \leq |T| + K$. Observe that, along the computation of $\mathcal{M}$, the maximum size of the guessed tree is $|T_0| + K|\varphi_0|$, which is polynomial in the size of the input. Therefore, $\mathcal{M}$ runs in polynomial time. □

Before proving Theorem 9, let us make a remark that the standard automata construction (cf. [14]) for the set of trees satisfying a given HM formula with respect to a given GTRS suffers from a nonelementary blow-up and is therefore insufficient for proving Theorem 9. We now present a proof for Theorem 9. Let us first recall the definition of modality ranks of HM formulas. Given an EF_0 formula φ, the *modality rank* of φ, denoted by $\mathsf{mrank}(\varphi)$, is the maximum nesting depth of modal operators in the formula: (i) $\mathsf{mrank}(\mathtt{true}) = 0$, (ii) $\mathsf{mrank}(\neg\varphi) = \mathsf{mrank}(\varphi)$, (iii) $\mathsf{mrank}(\varphi \wedge \psi) = \max\{\mathsf{mrank}(\varphi), \mathsf{mrank}(\psi)\}$, and (iv) $\mathsf{mrank}(\langle \Sigma \rangle \varphi) = \mathsf{mrank}(\varphi) + 1$. Also, without loss of generality, we will make the following assumption in our proof:

Convention 1 *There exists a letter in* A *with rank 2.*

This convention is done only for technical convenience and does not affect the result in this paper.

Let us now suppose that $\langle \Sigma^* \rangle \varphi$ is the given formula, $\mathcal{R} = (A, \mathbb{A}, R)$ is the given GTRS, and $T_0 \in \mathsf{Trees}_A$ is the input tree. We wish to check whether $(\mathfrak{S}(\mathcal{R}), T_0) \models \langle \Sigma^* \rangle \varphi$. First off, we can compute in polynomial time an NTA $\mathcal{A}$ recognizing the set $\mathsf{post}^{\Sigma^*}_{\mathcal{R}}(T_0)$ of configurations of $\mathcal{R}$ reachable by applications of rules with labels from Σ (cf. [14]). Therefore, to complete the proof, it suffices to show that (1) $[\![\varphi]\!]_{\mathfrak{S}(\mathcal{R})}$ can be expressed as a union of regular tree languages, each of which can be expressed by a tree automaton $\mathcal{A}_i$ of singly-exponential size, and (2) we can check whether some $L(\mathcal{A}_i)$ intersects with $L(\mathcal{A})$ in nondeterministic time polynomial in $|\mathcal{A}|$ and exponential in $|\varphi|$ and $|\mathcal{R}|$.

Lemma 11. *We have* $[\![\varphi]\!]_{\mathfrak{S}(\mathcal{R})} = \bigcup_{i \in I} L(\mathcal{A}_i)$*, for a family* $\{\mathcal{A}_i\}_{i \in I}$*, where* $|\mathcal{A}_i| = \exp(|\varphi|, |\mathcal{R}|)$*. One can nondeterministically check whether* $L(\mathcal{A})$ *intersects with some* $L(\mathcal{A}_i)$ *in time* $\mathsf{poly}(\mathcal{A}) \cdot \exp(|\varphi|, |\mathcal{R}|)$.

Remark 12. As we shall see in our proof below, the parameter $|\varphi|$ in the above lemma can be replaced by $\mathsf{mrank}(\varphi)$. As a corollary, this yields the same NEXP upper bound for the model checking problem when φ is given as a DAG.

We now give a proof of Lemma 11. Let $r = \mathsf{mrank}(\varphi)$. We start by defining a standard equivalence relation on Trees_A based on the modality rank of EF_0 formulas.

Definition 13 (First equivalence). *Given two trees $T, T' \in \mathsf{Trees}_A$ and $i \in \mathbb{N}$, we write $T \simeq_i T'$ if and only if for every EF_0 formula ψ with $\mathsf{mrank}(\psi) \leq i$:*

$$(\mathfrak{S}(\mathcal{R}), T) \models \psi \iff (\mathfrak{S}(\mathcal{R}), T') \models \psi.$$

In other words, $T \simeq_i T'$ if and only if T and T' agree on every formula ψ with modality rank at most i. It is obvious that $\simeq_i$ is an equivalence relation and that $T \simeq_{i+1} T'$ implies $T \simeq_i T'$. Furthermore, it is well-known that the equivalence relation $\simeq_i$ is of finite index, i.e., the number of equivalence classes of $\simeq_i$ is finite. For each equivalence class $\mathcal{C}$ corresponding to $\simeq_r$, it is clear that either $(\mathfrak{S}(\mathcal{R}), T) \models \varphi$ for all $T \in \mathcal{C}$, or $(\mathfrak{S}(\mathcal{R}), T) \not\models \varphi$ for all $T \in \mathcal{C}$. For the former case, we say that the equivalence class $\mathcal{C}$ is *positive*; otherwise, it is *negative*. Therefore, one idea is to define the family $\{\mathcal{A}_i\}_{i \in I}$ of NTAs by associating an NTA for each positive equivalence class $\mathcal{C}$ corresponding to $\simeq_r$. Two problems with this approach, however, are: (1) it is unclear how to compute an NTA for each positive equivalence class, and (2) this does not reveal an upper bound on the index of $\simeq_r$.

We shall now define a finer relation $\equiv_i$ (for each $i \in \mathbb{N}$) that will give extra information which will help us solve these two problems. To this end, let K be the the maximum number of nodes in trees appearing in any rewrite rule in $\mathcal{R}$. Given any tree $T : D \to A$, we define the i^{th} *type* (or simply *type*) $\mathsf{type}_i(T)$ of T as the set of all $s \in D$ such that $T^{\downarrow s}$ has at most $N_i = i \cdot K$ nodes, It is clear that, for two nodes $u, v \in D$ satisfying $u \preceq v$, we have $u \in \mathsf{type}_i(T)$ implies $v \in \mathsf{type}_i(T)$. Likewise, we have $\mathsf{type}_{i-1}(T) \subseteq \mathsf{type}_i(T)$. Given a tree $t \in \mathsf{Trees}_A$ and a subset $S \subseteq \mathsf{type}_i(T)$, we also use $m_{t,S}$ to denote the number of nodes u in S such that $T^{\downarrow u}$ is isomorphic to t. Next, we define an equivalence relation $\equiv_i$ on Trees_A.

Definition 14 (Second equivalence). *Given two trees $T, T' \in \mathsf{Trees}_A$ and $i \in \mathbb{N}$, we write $T \equiv_i T'$ if for each tree $t \in T_A$ with at most N_i nodes, either ($m_{t,\mathsf{type}_i(T)} \geq N_i$ and $m_{t,\mathsf{type}_i(T')} \geq N_i$) or $m_{t,\mathsf{type}_i(T)} = m_{t,\mathsf{type}_i(T')}$.*

In other words, $T \equiv_i T'$ if and only if each subtree with at most N_i nodes appears in T and T' the same number of times (up to the threshold N_i). As before, it is easy to check that $\equiv_i$ is an equivalence relation and that $T \equiv_{i+1} T'$ implies $T \equiv_i T'$. To complete the proof of Lemma 11, we proceed as follows: (1) show that $\equiv_i$ is finer than $\sim_i$, (2) checking whether a function $f : \mathsf{Trees}_A^{\leq N_i} \to [0, N_i]$ actually describes an equivalence class of $\equiv_i$ can be done rather efficiently, (3) testing whether an equivalence class of $\equiv_i$ is positive (with respect to φ) can be done rather efficiently, and (4) for each positive equivalence class $\mathcal{C}$ of $\equiv_i$, an NTA $\mathcal{A}_\mathcal{C}$ recognizing $\mathcal{C}$ can be computed rather efficiently. As we will see, these will imply Lemma 11. Let us first show (1).

Lemma 15. *For each tree $T, T' \in \mathsf{Trees}_A$, it is the case that $T \equiv_i T'$ implies $T \simeq_i T'$.*

Intuitively, this lemma holds since satisfaction of EF_0 formulas of modality rank i is only affected by the number of occurences of trees of depth N_i (up to some threshold).

Proof. Let D (resp. D') denote the domain of T (resp. T'). The proof is by induction on $i \in \mathbb{N}$. The base case of $i = 0$ is vacuous. Let $i > 0$ and let ψ be an EF_0 formula with $\mathsf{mrank}(\psi) = i$. We shall only prove one direction: $(\mathfrak{S}(\mathcal{R}), T) \models \psi$ implies $(\mathfrak{S}(\mathcal{R}), T') \models \psi$; the converse is symmetric. Suppose that $(\mathfrak{S}(\mathcal{R}), T) \models \psi$. We proceed by cases depending on the structure of ψ. The cases of boolean combinations are immediate. Therefore, assume that $\psi = \langle a \rangle \psi'$. Then, there exists a tree $T_1 = T[u/t_2] \in \mathsf{Trees}_A$ such that $t_1 \overset{a}{\hookrightarrow} t_2$ is a rule in $\mathcal{R}$, u is a node in T such that $T^{\downarrow u} = t_1$, and $(\mathfrak{S}(\mathcal{R}), T_1) \models \psi'$. Since $T \equiv_i T'$, there exists a node u' in T' such that $(T')^{\downarrow u'} = t_1$. In fact, u' can be chosen in such a way that if w and w' are respectively the topmost ancestors of u and u' in $\mathsf{type}_i(T)$ and $\mathsf{type}_i(T')$, then $T^{\downarrow w}$ is isomorphic to $(T')^{\downarrow w'}$; for, otherwise, $T^{\downarrow w}$ is a subtree of T that is not a subtree of T', contradicting the assumption that $T \equiv_i T'$. This situation is depicted in Figure 1. We now let $T'_1 = T'[u'/t_2]$.

It suffices to prove that $T_1 \equiv_{i-1} T'_1$; for, if this is the case, we have $(\mathfrak{S}(\mathcal{R}), T'_1) \models \psi'$ by induction hypothesis and so $(\mathfrak{S}(\mathcal{R}), T') \models \psi$. This fact shall be proven by considering separately the nodes in $\mathsf{type}_{i-1}(T_1)$ (resp. $\mathsf{type}_{i-1}(T'_1)$) that are in the subtree rooted at w (resp. w'), which contains all nodes in T (resp. T') that might be affected by the application of the rule $t_1 \overset{a}{\hookrightarrow} t_2$. Observe now that if $v \in$

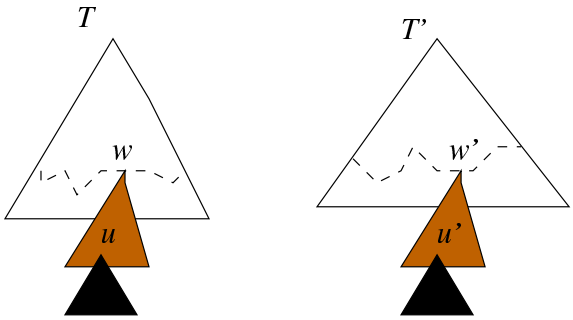


Fig. 1. A depiction of trees T and T'. The nodes below the broken lines are in $\mathsf{type}_i(T)$ and $\mathsf{type}_i(T')$ respectively. The nodes w and w' lie at the borders, i.e., their parents are no longer in $\mathsf{type}_i(T)$ and $\mathsf{type}_i(T')$. We have $T^{\downarrow w} = (T')^{\downarrow w'}$.

$\mathsf{type}_{i-1}(T_1)$ (resp. $v' \in \mathsf{type}_{i-1}(T_1')$) and v is an ancestor of u (resp. u'), then $v \in \mathsf{type}_i(T)$ (resp. $v' \in \mathsf{type}_i(T')$) since the number of nodes can increase/decrease only by at most $K-1$ (i.e. the absolute value of $|t_1| - |t_2|$ is at most $K-1$). In particular, each such v (resp. v') must be a descendant of w (resp. w'). In the following, let D_1 (resp. D_1') denote the domain of T_1 (resp. T_1'). Define

$$\begin{aligned} E &= \{v \in D : \exists s(v = ws)\} \cap \mathsf{type}_{i-1}(T), \\ E_1 &= \{v \in D_1 : \exists s(v = ws)\} \cap \mathsf{type}_{i-1}(T_1), \\ E' &= \{v \in D' : \exists s(v = w's)\} \cap \mathsf{type}_{i-1}(T'), \\ E_1' &= \{v \in D_1' : \exists s(v = w's)\} \cap \mathsf{type}_{i-1}(T_1'). \end{aligned}$$

These are precisely the nodes of the subtrees of (respectively) T, T', T_1 and T_1' rooted w or w' with at most $N_{i-1} < N_i - K$ nodes. Let

$$\begin{aligned} S &= \mathsf{type}_{i-1}(T) \setminus E, \\ S_1 &= \mathsf{type}_{i-1}(T_1) \setminus E_1 \\ S' &= \mathsf{type}_{i-1}(T') \setminus E', \\ S_1' &= \mathsf{type}_{i-1}(T_1') \setminus E_1'. \end{aligned}$$

Observe that $S = S_1$ and $S' = S_1'$. In particular, for each tree $t \in \mathsf{Trees}_A$ with at most N_{i-1} nodes, we have $m_{t,S} = m_{t,S_1}$ and $m_{t,S'} = m_{t,S_1'}$. Furthermore, since $T^{\downarrow w} = (T')^{\downarrow w'}$ and $T_1^{\downarrow w} = (T_1')^{\downarrow w'}$, we have $m_{t,E} = m_{t,E'}$ and $m_{t,E_1} = m_{t,E_1'}$. Therefore, since

$$\begin{aligned} m_{t,\mathsf{type}_{i-1}(T)} &= m_{t,S} + m_{t,E}, \\ m_{t,\mathsf{type}_{i-1}(T_1)} &= m_{t,S_1} + m_{t,E_1}, \\ m_{t,\mathsf{type}_{i-1}(T')} &= m_{t,S'} + m_{t,E'}, \\ m_{t,\mathsf{type}_{i-1}(T_1')} &= m_{t,S_1'} + m_{t,E_1'}, \end{aligned}$$

we have

$$m_{t,\mathsf{type}_{i-1}(T_1)} - m_{t,\mathsf{type}_{i-1}(T)} = m_{t,\mathsf{type}_{i-1}(E_1)} - m_{t,\mathsf{type}_{i-1}(E)}$$

which is in turn equal to

$$m_{t,\mathsf{type}_{i-1}(T_1')} - m_{t,\mathsf{type}_{i-1}(T')} = m_{t,\mathsf{type}_{i-1}(E_1')} - m_{t,\mathsf{type}_{i-1}(E')}.$$

Let us therefore analyze $m_{t,\mathsf{type}_{i-1}(E_1)} - m_{t,\mathsf{type}_{i-1}(E)}$ more precisely. We need to count the possible occurrences of subtrees t of T_1 of size at most N_{i-1} that are rooted in E_1 and compare this with the number of occurrences of t in T rooted in E . Conversely, we need to count the possible occurrence of subtrees t

of T of size at most N_{i-1} that are rooted in E and compare this with the number of occurrences of t in T_1 rooted in E_1. Firstly, assume that t occurs in T_1 and the root of t is a descendant of u. Then by the applied rewriting rule we can introduce at most K such many copies of t in T_1 which were potentially not present in T. Analogously, if t occurs in T and the root of t is a descendant of u, then by the applied rewriting rule we can remove at most K such many copies of t in T_1. Secondly, assume that t occurs in T_1 and the root of t is in E_1 and this root is an ancestor of u. Then at most one such tree can be introduced in T_1 which was potentially not present in T. Conversely, assume that t occurs in T_1 and the root of t is in E_1 and this root is an ancestor of u. Then at most one such tree can be introduced in T_1 which was potentially not present in T. Thirdly, assume that the root of t is neither a descendant nor an ancestor of u. Then the applied rewriting rule can neither have introduced nor removed this tree. Thus, we have that the absolute value of

$$m_{t,\mathsf{type}_{i-1}(E_1)} - m_{t,\mathsf{type}_{i-1}(E)}$$

and hence that of

$$m_{t,\mathsf{type}_{i-1}(T_1)} - m_{t,\mathsf{type}_{i-1}(T)}$$

is bounded by K. The latter difference is equal to $m_{t,\mathsf{type}_{i-1}(T_1')} - m_{t,\mathsf{type}_{i-1}(T')}$.

One now verifies easily that either $m_{t,\mathsf{type}_{i-1}(T_1)} = m_{t,\mathsf{type}_{i-1}(T_1')}$ or ($m_{t,\mathsf{type}_{i-1}(T_1)} \geq N_{i-1}$ and $m_{t,\mathsf{type}_{i-1}(T_1')} \geq N_{i-1}$). □

Let us now proceed to step (2). Recall that each equivalence class $\mathcal{C}$ of $\equiv_r$ can be described by a function $f_{\mathcal{C}} : \mathsf{Trees}_A^{\leq N_r} \to [0, N_r]$. The converse, however, is false, e.g., it is impossible to have a class $\mathcal{C}$ with $f_{\mathcal{C}}(T) > 1$ for a tree T with two nodes but $f_{\mathcal{C}}(T') = 0$ for all trees T' with 1 node. Also observe that each tree has non-empty domain. Hence the function that maps every tree from $\mathsf{Trees}_A^{\leq N_r}$ to 0 cannot correspond to an equivalence class of $\equiv_r$. We will need to be able to check whether a given function $f : \mathsf{Trees}_A^{\leq N_r} \to [0, N_r]$ *actually* describes an equivalence class of $\equiv_r$.

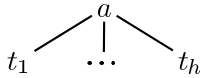
To this end, recall first that any function f that describes an equivalence class of $\equiv_r$ counts each subtree of trees T in $\mathsf{Trees}_A^{\leq N_r}$ with $f_{\mathcal{C}}(T) > 0$, i.e., if t is a subtree of T, then t contributes to the value of $f(t)$. We will first define a new function $g : \mathsf{Trees}_A^{\leq N_r} \to [0, N_r]$ that avoids this "double counting". This can be done by the following algorithm:

```
Repeat the following for each T ∈ Trees_A^{≤N_r} with f(T) > 0
(ordered by the number of nodes, starting from the largest):

(1) Let f(T) := f(T) − 1.
(2) Let g(T) := g(T) + 1.
(3) Go through all nodes u of T and substract f(T^{↓u}) by 1
(if becomes negative, then terminate abruptly).
```

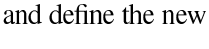
Note that $g(t)$ counts the number of times the tree t occurs as a subtree $T^{\downarrow u}$ in T such that u is the root of T or the subtree rooted at the parent of u has more that N_r nodes. Observe that if this algorithm terminates abruptly, then f does not actually describe an equivalence class of $\mathcal{C}$. Let us analyze the running time of the algorithm. By taking Cayley's formula as an upper bound, there are at most $N_r^{N_r-2}$ many unlabeled unrooted trees with N_r nodes. Hence there at most $N_r \cdot N_r^{N_r-2}$ many unlabeled rooted trees with N_r nodes. And thus there are at most $N_r^2 \cdot N_r^{N_r-2}$ many unlabeled rooted trees with *at most* N_r nodes. For each of these trees there are at most $|A|^{N_r}$ many possibilities of coloring it with labels from A. Thus, $|\mathsf{Trees}_A^{\leq N_r}| = \exp(r, |\mathcal{R}|)$. Hence, the algorithm runs in time exponential in $r \leq |\varphi|$ and $|\mathcal{R}|$. Now, suppose that the function g has been successfully computed from the given function f. This implies that g describes a *forest* F with each tree $T \in \mathsf{Trees}_A^{\leq N_r}$ occurring $g(T)$ many times. The original function f then describes an equivalence class if and only if such a forest can be further "connected into a big tree". This last check can be done using the following lemma.

Lemma 16. *The function $f : \mathsf{Trees}_A^{\leq N_r} \to [0, N_r]$ describes an equivalence class in $\equiv_r$ if and only if the function $g : \mathsf{Trees}_A^{\leq N_r}$ (and the forest F corresponding to it) has been successfully computed from f by the above algorithm and one of the following conditions is satisfied:*

1. $\sum_{t \in \mathsf{Trees}_A^{\leq N_r}} g(t) = 1$.
2. $\sum_{t \in \mathsf{Trees}_A^{\leq N_r}} g(t) > 1$, *and there must exist a letter* a *with rank* h *and trees* $t_1, \ldots, t_h$ *occuring in the forest* F *such that the tree* $T =$  *has more than* N_r *nodes.*

Observe that this lemma completes step (2) since this test can be performed in time exponential in r and $|\mathcal{R}|$.

Proof. We start with "only if" direction. Suppose that f describes an equivalence class in $\equiv_r$. The function g will then be successfully computed. Now if Condition (1) is violated, then by the construction of the function g, it must be the case that $\sum_{t \in \mathsf{Trees}_A^{\leq N_r}} g(t) > 1$. Furthermore, using the representative from the equivalence class described by f there exist a letter a with rank h and trees $t_1, \ldots, t_h$ occuring in the forest F such that the tree T (defined above) has more than N_r nodes.

Conversely, if Condition (1) is satisfied, then obviously f describes an equivalence class in $\equiv_r$ (containing the single tree counted by g). If Condition (2) is satisfied, then we may construct a tree T' that realizes the function f as follows. Let us first construct the tree $T_0 = T$, where T is given in the statement of Condition (2). For each tree t in the forest F that is still not connected, we may define a new tree T_1 by choosing a letter a with rank 2, which we assumed to exist, and define the new tree T_1 to be  obtained via linking T_0 to t. We then continue to another tree t' in the forest and define T_2 by connecting t' to T_1 in the same way. This procedure continues until all the trees in the forest have been connected. The desired tree T' is defined to be the final tree obtained by this procedure. The fact that T' witnesses f follows since each T^i has more than N_r nodes and therefore is not counted by f. □

We now proceed to step (3). This step is rather easy.

Lemma 17. *Checking whether an equivalence class* $\mathcal{C}$ *of* $\equiv_r$ *described by a function* $f_{\mathcal{C}} : \mathsf{Trees}_A^{\leq N_r} \to [0, N_r]$ *is positive can be done in time* $|f|^{\mathsf{poly}(r,|\mathcal{R}|)} \cdot \exp(r, |\mathcal{R}|)$.

Proof. The proof of Lemma 16 tells us that there exists a tree $T \in \mathcal{C}$ of size exponential in r and $|\mathcal{R}|$ that witnesses $f_{\mathcal{C}}$. This tree T can also be computed in time exponential in r and $|\mathcal{R}|$. We may then check whether $(\mathfrak{S}(\mathcal{R}), T) \models \varphi$ in as follows. Firstly, compute a finite transition system $\mathfrak{F}$ whose domain contains the set of all configurations of $\mathcal{R}$ of distance at most r in $\mathfrak{S}(\mathcal{R})$ (i.e. all r-neighbours of T in $\mathfrak{S}(\mathcal{R})$) and whose transition relations are simply the restriction of the transition relations of $\mathfrak{S}(\mathcal{R})$ to this neighborhood. Since T is exponential in r and $|\mathcal{R}|$, it follows that $|\mathfrak{F}|$ is also exponential in r and $|\mathcal{R}|$. We may then use the standard algorithm for model checking HM formulas to evaluate whether $(\mathfrak{F}, T) \models \varphi$, which runs in time polynomial in $|\mathfrak{F}|$ and $|\varphi|$. □

We now proceed to step (4), which is the final step. For this, we need to show how to compute an NTA recognizing an equivalence class $\mathcal{C}$ of $\equiv_r$ described by a function $f_{\mathcal{C}} : \mathsf{Trees}_A^{\leq N_r} \to [0, N_r]$. The following lemma says that we can do this in time exponential in r and $|\mathcal{R}|$.

Lemma 18. *Given a function* $f : \mathsf{Trees}_A^{\leq N_r} \to [0, N_r]$ *that witnesses an equivalence class* $\mathcal{C}$ *of* $\equiv_r$*, we can compute an NTA recognizing precisely* $\mathcal{C}$ *in time* $|f|^{\mathsf{poly}(r,|\mathcal{R}|)} \cdot \exp(r, |\mathcal{R}|)$.

Proof. Given f, we first compute the function $g : \mathsf{Trees}_A^{\leq N_r} \to [0, N_r]$ (using the above algorithm). For any $T \in \mathcal{C}$, recall that $g(t)$ counts the number of times t occurs as a subtree $T^{\downarrow u}$ in T such that u is the root of T or the subtree rooted at the parent of u has more than N_r nodes.

Let U denote all trees $t \in \mathsf{Trees}_A^{\leq N_r}$ such that $g(t) = N_r$. Let $t_1, \ldots, t_m$ be an enumeration of all trees $t \in \mathsf{Trees}_A^{\leq N_r}$ with $g(t) > 0$ *without* counting multiplicities. Let us fix a tree $T \in \mathsf{Trees}_A$. Define $\mathsf{yields}(T)$ to be the set of incomparable nodes u of T (with respect to $\preceq$ ordering) such that $|T^{\downarrow u}| \leq N_r$ and either u is the root or the subtree rooted at the parent of u has more than N_r nodes. We can now color $\mathsf{yields}(T)$ with the $m + 1$ colors $t_1, \ldots, t_m$ and U such that for each $u \in \mathsf{yields}(T)$ we have that if u is colored with U we have $T^{\downarrow u} \in U$ and whenever u is colored with t_i we have $T^{\downarrow u} = t_i$. It now holds that $T \in \mathcal{C}$ if and only if exactly $g(t_i)$ nodes in $\mathsf{yields}(T)$ can be colored with t_i for each t_i with $g(t_i) > 0$ and all remaining nodes from $\mathsf{yields}(T)$ can be colored with U.

We now sketch how an NTA recognizing $\mathcal{C}$ can be computed. Let T be an input tree. Initially, for each leaf node w of T, the NTA will guess one of the $m+1$ colors by which the unique ancestor u of w that is in $\mathsf{yields}(T)$ will be colored. It is now standard to check whether this guessing was consistent just until u is reached. For each t_i with with $g(t_i) > 0$ the NTA stores a counter to count the number of elements of $\mathsf{yields}(T)$ that are colored with t_i. It is easy to see that the described NTA recognizes $\mathcal{C}$ and can be computed in time exponential in r and $|\mathcal{R}|$. □

To summarize, the proof of Lemma 11 can now be done as follows. The NTAs $\mathcal{A}_i$ in the statement of Lemma 11 will correspond to positive equivalence classes $\mathcal{C}$ described by some functions $f_{\mathcal{C}} : \mathsf{Trees}_A^{\leq N_r} \to [0, N_r]$. Using the last step above, the NTA $\mathcal{A}_i$ can be computed in time exponential in r and $|\mathcal{R}|$ if f is given as an input. Checking whether $L(\mathcal{A}) \cap L(\mathcal{A}_i) \neq \emptyset$ for some i requires us to nondeterministically guess one such function f, verify if it describes a positive equivalence class, compute the NTA $\mathcal{A}_i$ corresponding to it, and check for language intersection with $\mathcal{A}$ in the standard way.

3.3 Model checking EF_1 over GTRS is hard for $\mathsf{P}^{\mathsf{NEXP}}$

Goal of this section is to provide a matching lower bound to the $\mathsf{P}^{\mathsf{NEXP}}$ upper bound for EF_1 from Section 3.2. We proceed in two steps. In step one, we prove that model checking EF_1 formulas of the kind $\langle \mathbb{A}^* \rangle \varphi$, where φ is an EF_0 formula, is hard for NEXP. In step two we extend the latter proof to prove that EF_1 model checking is hard for $\mathsf{P}^{\mathsf{NEXP}}$.

Step one

A *tiling system* is a tuple $S = (\Theta, H, V)$, where Θ is a finite set of *tile types*, $H \subseteq \Theta \times \Theta$ is a *horizontal matching relation*, and $V \subseteq \Theta \times \Theta$ is a *vertical matching relation*. For each word $w = w_1 \cdots w_n \in \Theta^n$, $n \geq 1$, we say a mapping $\sigma : [0, k-1] \times [0, k-1] \to \Theta$ (where $k \geq n$) is a *k-solution to w with respect to S* if for all $(x, y) \in [0, k-1] \times [0, k-1]$ the following holds:

- if $\sigma(x, y) = \theta$ and $\sigma(x+1, y) = \theta'$, then $(\theta, \theta') \in H$ and
- if $\sigma(x, y) = \theta$ and $\sigma(x, y+1) = \theta'$, then $(\theta, \theta') \in V$ and
- $\sigma(x, 0) = w_{x+1}$ for each $x \in [0, n-1]$.

For such a tiling system S we define the $2^n \times 2^n$ *tiling problem for S* as follows:

$2^n \times 2^n$ TILING PROBLEM FOR TILING SYSTEM $S = (\Theta, H, V)$

INPUT: A word $w \in \Theta^n$.
QUESTION: Is there a 2^n-solution to w with respect to S?

The following proposition is folklore, see also [18].

Proposition 19 ([18]). *There is some fixed tiling system S_0 whose $2^n \times 2^n$ tiling problem is NEXP-hard.*

We now proceed to prove step one.

Theorem 20. *The following problem is hard for NEXP:*

INPUT: *A GTRS $\mathcal{R} = (A, \mathbb{A}, R)$, a tree $T_0 \in \mathsf{Trees}_A$, and a formula $\varphi \in \mathsf{EF}_0$.*
QUESTION: $(\mathfrak{S}(\mathcal{R}), T_0) \models \langle \mathbb{A}^* \rangle \varphi$*?*

Proof. Let us fix the tiling system $S_0 = (\Theta, H, V)$ of Proposition 19 whose $2^n \times 2^n$ tiling problem is NEXP-hard. Let $w = \theta_1 \cdots \theta_n \in \Theta^n$ be an input of the $2^n \times 2^n$ tiling problem w.r.t. S_0. We will compute in polynomial time a GTRS $\mathcal{R} = \mathcal{R}(S_0, w) = (A, \mathbb{A}, R)$, some tree $T_0 = T_0(S_0, w) \in \mathsf{Trees}_A$ and some formula $\varphi \in \mathsf{EF}_0$ such that

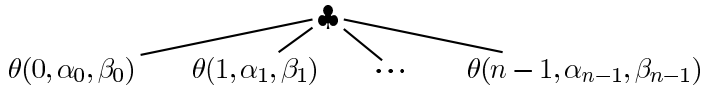
$$\text{there is a } 2^n\text{-solution to } w \text{ w.r.t. } S_0 \qquad \text{if and only if} \qquad (\mathfrak{S}(\mathcal{R}), T_0) \models \langle \mathbb{A}^* \rangle \varphi.$$

First, let us define the ranked alphabet A. We put $A = A_0 \uplus A_2 \uplus A_n$, where

– $A_0 = \{\bigstar\} \cup \{\theta(i), \overline{\theta(i)}, \widehat{\theta(i)}, \theta(i,\alpha,\beta), \overline{\theta(i,\alpha,\beta)}, \widehat{\theta(i,\alpha,\beta)} \mid \theta \in \Theta, i \in [0,n-1], \alpha,\beta \in \{0,1\}\}$,
– $A_2 = \{\blacksquare\}$, and
– $A_n = \{\clubsuit\}$.

Before we define the rewriting rules R, we introduce some auxiliary notions.

Let $\mathbb{L} = \{\theta(i,\alpha,\beta) \mid \theta \in \Theta, i \in [0,n-1], \alpha,\beta \in \{0,1\}\} \subseteq A_0$ denote the set of all *real leaves*. A *superleaf* is a tree of the following kind, where $\theta \in \Theta$ and $\alpha_i, \beta_i \in \{0,1\}$:

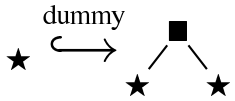


The intuition is that each superleaf corresponds to the θ-labeled grid element $(x,y) \in [0,2^n-1] \times [0,2^n-1]$, the α_i represent x and the β_i represent y in binary, more precisely we have $x = \sum_{i=0}^{n-1} \alpha_i \cdot 2^i$ and $y = \sum_{i=0}^{n-1} \beta_i \cdot 2^i$. We also call this a θ-labeled (x,y)-superleaf. The idea is to reach from T_0 via $\xrightarrow{\mathbb{A}}{}^*$ some tree T with superleafs that satisfies $\psi(S_0,n)$, where $\psi(S_0,n)$ is now a conjunction of EF_0 formulas expressing the following:

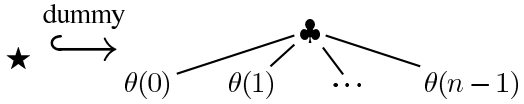
(1) Every leaf (that has a label from A_0)[2] is a real leaf.
(2) A $(0,0)$-superleaf exists.
(3) For each $(x,y) \in [0,2^n-1] \times [0,2^n-1]$ our tree T has a θ-labeled (x,y)-superleaf for at most one $\theta \in \Theta$.
(4) If T has a θ-labeled (x,y)-superleaf, where $x < 2^n-1$, then T has a θ'-labeled $(x+1,y)$-superleaf with $(\theta,\theta') \in H$.
(5) If T has a θ-labeled (x,y)-superleaf, where $y < 2^n-1$, then T has a θ'-labeled $(x,y+1)$-superleaf with $(\theta,\theta') \in V$.

Let us now give the rewrite rules of $\mathcal{R}$. These implictly give us the set of actions $\mathbb{A}$. We have the following rules for each $i \in [0,n-1]$, $\theta \in \Theta$ and $\alpha,\beta \in \{0,1\}$.

– $a \overset{a}{\hookrightarrow} a$ for each $a \in A_0$: We make the presence of each leaf (label) visible to $\mathfrak{S}(\mathcal{R})$.
– Expansion of the tree:



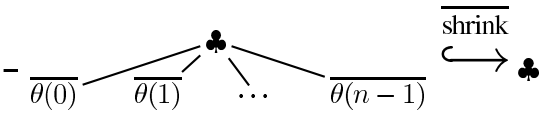
– A stepwise creation of a superleaf without specifying the information of the children.



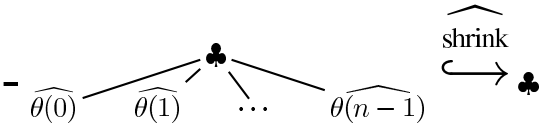
– $\theta(i) \overset{\text{dummy}}{\hookrightarrow} \theta(i,\alpha,\beta)$: We allow to generate all kinds of real leaves but stay with the index i.

– $\theta(i,\alpha,\beta) \overset{\overline{\text{mark}}}{\hookrightarrow} \overline{\theta(i,\alpha,\beta)}$: We can mark each real leaf with $\overline{\ }$.
– $\overline{\theta(i,\alpha,\beta)} \overset{\overline{\text{forget}}}{\hookrightarrow} \overline{\theta(i)}$: We can forget the information of each $\overline{\ }$-labeled child of a superleaf.
– $\theta(i,\alpha,\beta) \overset{\widehat{\text{mark}}}{\hookrightarrow} \widehat{\theta(i,\alpha,\beta)}$: A second way to mark a real leaf with $\widehat{\ }$.
– $\widehat{\theta(i,\alpha,\beta)} \overset{\widehat{\text{forget}}}{\hookrightarrow} \widehat{\theta(i)}$: We can forget the information of each $\widehat{\ }$-labeled child of a superleaf.

[2] In our $\mathsf{P}^{\mathsf{NEXP}}$ lower bound proof in step two we will allow leaves with labels that are not in A_0, for those leaves, we require nothing.

– 

Allows to check if marked real leaves belong to the same superleaf.

– 

Allows to check if marked real leaves belong to the same superleaf.

Let us realize each of the above-mentioned properties **(1)** up to **(6)** by EF_0 formulas:

(1) Every leaf is a real leaf:

$$\bigwedge_{a\in A_0\setminus\mathbb{L}} \neg\langle a\rangle \mathtt{true}$$

Note that by construction of $\mathcal{R}$ this implies that all leaves of T are unmarked real leaves and are children of superleaves.

(2) A $(0,0)$-superleaf exists:

$$\bigvee_{\theta\in\Theta} \langle\overline{\text{mark}}\rangle^n \ \langle\overline{\theta(0,0,0)}\rangle\langle\overline{\theta(1,0,0)}\rangle\cdots\langle\overline{\theta(n-1,0,0)}\rangle \ \langle\overline{\text{forget}}\rangle^n\langle\overline{\text{shrink}}\rangle\mathtt{true}$$

(3) For each $(x,y)\in[0,2^n-1]\times[0,2^n-1]$ our tree T has a θ-labeled (x,y)-superleaf for at most one $\theta\in\Theta$.

$$[\overline{\text{mark}}]^n\left[\widehat{\text{mark}}\right]^n\left(\langle\overline{\text{forget}}\rangle^n\left\langle\widehat{\text{forget}}\right\rangle^n\langle\overline{\text{shrink}}\rangle\left\langle\widehat{\text{shrink}}\right\rangle\mathtt{true}\longrightarrow\neg\chi_{\text{same}}\right),\qquad\text{where}$$

$$\chi_{\text{same}} \;=\; \bigvee_{\substack{\theta,\theta'\in\Theta\\ \theta\neq\theta'}}\ \bigwedge_{\substack{i\in[0,n-1]\\ \alpha,\beta\in\{0,1\}}}\left(\left\langle\overline{\theta(i,\alpha,\beta)}\right\rangle\mathtt{true}\longleftrightarrow\left\langle\theta'\widehat{(i,\alpha,\beta)}\right\rangle\mathtt{true}\right)$$

(4) If T has a θ-labeled (x,y)-superleaf, where $x<2^n-1$, then T has a θ'-labeled $(x+1,y)$-superleaf with $(\theta,\theta')\in H$.

$$[\overline{\text{mark}}]^n\left(\left(\left\langle\overline{\text{forget}}\right\rangle^n\left\langle\overline{\text{shrink}}\right\rangle\mathtt{true}\ \wedge\bigvee_{\substack{i\in[0,n-1],\theta\in\Theta\\ \beta\in\{0,1\}}}\overline{\theta(i,0,\beta)}\right)\longrightarrow\right.$$

$$\left.\left\langle\widehat{\text{mark}}\right\rangle^n\left(\left\langle\widehat{\text{forget}}\right\rangle^n\left\langle\widehat{\text{shrink}}\right\rangle\mathtt{true}\ \wedge\bigvee_{(\theta,\theta')\in H}\text{same}_y(\theta,\theta')\wedge\text{inc}_x(\theta,\theta')\right)\right)$$

where

$$\text{same}_y(\theta,\theta') \;=\; \bigwedge_{i\in[0,n-1]}\ \bigvee_{\substack{\alpha,\alpha'\in\{0,1\}\\ \beta\in\{0,1\}}}\left\langle\overline{\theta(i,\alpha,\beta)}\right\rangle\mathtt{true}\longleftrightarrow\left\langle\theta'\widehat{(i,\alpha',\beta)}\right\rangle\mathtt{true}$$

$$\text{inc}_x(\theta,\theta') \;=\; \bigvee_{\substack{i\in[0,n-1]\\ \beta\in\{0,1\}}}\left(\left\langle\overline{\theta(i,0,\beta)}\right\rangle\mathtt{true}\wedge\left\langle\theta'\widehat{(i,1,\beta)}\right\rangle\mathtt{true}\wedge\text{carry}(i,\theta,\theta')\wedge\text{same}_x(>i,\theta,\theta')\right)$$

$$\text{carry}(i,\theta,\theta') \;=\; \bigwedge_{0\le j<i}\ \bigvee_{\beta\in\{0,1\}}\left\langle\overline{\theta(j,1,\beta)}\right\rangle\mathtt{true}\wedge\left\langle\theta'\widehat{(j,0,\beta)}\right\rangle\mathtt{true}$$

$$\text{same}_x(>i,\theta,\theta') \;=\; \bigwedge_{i<j\le n-1}\ \bigvee_{\substack{\alpha\in\{0,1\}\\ \beta\in\{0,1\}}}\left\langle\overline{\theta(j,\alpha,\beta)}\right\rangle\mathtt{true}\longleftrightarrow\left\langle\theta'\widehat{(j,\alpha,\beta)}\right\rangle\mathtt{true}$$

(5) If T has a θ-labeled (x,y)-superleaf, where $y < 2^n - 1$, then T has a θ'-labeled $(x, y+1)$-superleaf with $(\theta, \theta') \in V$: This can be done similarly as the formula for case **(4)**.

Recall that $w = \theta_1 \cdots \theta_n$. We now define a formula $\mathsf{input}(S_0, w)$ that expresses that T has a θ_{i+1}-labeled $(i,0)$-superleaf for each $i \in [0, n-1]$. Let us assume that $\alpha_0(x-1)\alpha_1(x-1)\cdots\alpha_{n-1}(x-1)$ is the binary encoding of $x-1$, i.e. $x - 1 = \sum_{i \in [0,n-1]} \alpha_i(x-1)2^i$, for each positon $x \in [1,n]$ of w. Recall that $w = \theta_1 \cdots \theta_n$. The required formula is

$$\mathsf{input}(S_0, w) \;=\; \bigwedge_{x \in [1,n]} \langle \overline{\mathsf{mark}} \rangle^n \left(\langle \overline{\mathsf{forget}} \rangle^n \langle \overline{\mathsf{shrink}} \rangle \, \mathtt{true} \;\wedge \bigwedge_{i \in [0,n-1]} \left\langle \overline{\theta_x(i, \alpha_i(x-1), 0)} \right\rangle \mathtt{true} \right).$$

By putting $T_0 = \bigstar$ and $\varphi = \psi(S_0, n) \wedge \mathsf{input}(S_0, w)$ one can now verify that

$$\text{there is a } 2^n\text{-solution to } w \text{ w.r.t. } S_0 \qquad \text{if and only if} \qquad (\mathfrak{S}(\mathcal{R}), T_0) \models \langle \mathbb{A}^* \rangle \varphi.$$

□

Step two

Without loss of generality we assume that for the fixed tiling system $S_0 = (\Theta, H, V)$ of Proposition 19 whose $2^n \times 2^n$-tiling problem is NEXP-hard we have $\Theta = \{0,1\}^k$ for some (fixed) $k \in \mathbb{N}$. We define the predicate

$$\mathbb{D}(w) \quad = \quad \begin{cases} 1 & \text{there is } 2^n \times 2^n \text{ solution to } w \text{ with respect to } S_0 \\ 0 & \text{otherwise} \end{cases} \qquad \text{for each } w \in \Theta^n.$$

In the following we treat elements of Θ^* as elements of $\{0,1\}^*$ and conversely elements $w \in \{0,1\}^*$, where $|w|$ is a multiple of k, as elements of Θ^*. A *domino circuit of oracle size* n is a sequence of assignments $\mathcal{C} = (x_i := y_i)_{i \in [1,l]}$, where each y_i is either of the following expressions,

(1) $y_i = 1$,
(2) $y_i = \neg x_j$ for some $j \in [1, i-1]$,
(3) $y_i = x_j \wedge x_h$, where $j, h \in [1, i-1]$, or
(4) $y_i = \mathbb{D}(w)$ where $w \in \{x_j \mid j \in [1, i-1]\}^{k \cdot n}$.

We define inductively $\mathsf{eval}(x_i) \in \{0,1\}$ as follows:

$$\mathsf{eval}(x_i) \quad = \quad \begin{cases} 1 & \text{if } y_i = 1 \\ 1 - \mathsf{eval}(x_j) & \text{if } y_i = \neg x_j \\ \min\{\mathsf{eval}(x_j), \mathsf{eval}(x_h)\} & \text{if } y_i = x_j \wedge x_h \\ \mathbb{D}\left(\prod_{r=1}^{k \cdot n} \mathsf{eval}(x_{j_r})\right) & \text{if } y_i = \mathbb{D}(x_{j_1} \cdots x_{j_{k \cdot n}}), \end{cases}$$

where $\prod$ denotes the concatenation of bits. We define $\mathsf{eval}(\mathcal{C}) = \mathsf{eval}(x_l)$. We can now formulate the following decision problem.

DOMINO CIRCUIT
INPUT: A domino cicuit $\mathcal{C}$.
QUESTION: $\mathsf{eval}(\mathcal{C}) = 1$?

The following propsition can be proven by standard arguments.

Proposition 21. *Domino circuit is* $\mathsf{P}^{\mathsf{NEXP}}$*-complete.*

Theorem 22. *Over GTRS model checking* EF_1 *is* $\mathsf{P}^{\mathsf{NEXP}}$*-hard.*

Proof (sketch). We reduce from domino cicuit. Let $\mathcal{C} = (x_i := y_i)_{i\in[1,l]}$ be an instance of domino circuit. Our goal is to compute a GTRS $\mathcal{R}' = (A', \mathbb{A}', R')$, some initial tree $T'_0 \in \mathsf{Trees}_{A'}$ and an EF_1 formula φ' such that

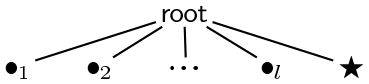
$$\mathsf{eval}(\mathcal{C}) = 1 \quad \text{if and only if} \quad (\mathfrak{S}(\mathcal{R}'), T'_0) \models \varphi'.$$

Let us compute in polynomial time the GTRS $\mathcal{R} = (A, \mathbb{A}, R)$ (we recycle the proof of Theorem 20) such that for each $w \in \Theta^n$ we have $\mathbb{D}(w)$ if and only if $(\mathfrak{S}(\mathcal{R}), \bigstar) \models \langle \mathbb{A}^* \rangle (\psi(S_0, n) \wedge \mathsf{input}(S_0, w))$. Recall that $A = A_0 \uplus A_2 \uplus A_n$. Assume w.l.o.g. that $n \notin \{0, 2, l+1\}$. We put $A' = A'_0 \uplus A'_2 \uplus A'_n \uplus A'_{l+1}$, where

- $A'_0 = A_0 \uplus \{\bullet_i, \bullet_i(0), \bullet_i(1) \mid i \in [1, l]\}$,
- $A'_2 = A_2$,
- $A'_n = A_n$, and
- $A'_{l+1} = \{\mathsf{root}\}$.

Let us first consider our initial tree T'_0:

$$T'_0 \quad = \quad$$



The idea is now simple. The rewriting rules R' consist of those already present in R and the rules that we will now explain. We allow the leaf rewriting rules $\bullet_i \overset{\mathsf{set}_i}{\hookrightarrow} \bullet_i(0)$ and $\bullet_i \overset{\mathsf{set}_i}{\hookrightarrow} \bullet_i(1)$ that correspond to setting the variable x_i to 0 and 1 respectively, for each $i \in [1, l]$. Additionally, via the rules $\bullet_i(0) \overset{\mathsf{get}_i(0)}{\hookrightarrow} \bullet_i(0)$ and $\bullet_i(1) \overset{\mathsf{get}_i(1)}{\hookrightarrow} \bullet_i(1)$ we allow to get the value of each of the variables, for each $i \in [1, l]$. Note that by definition, for the circuit $\mathcal{C}$ we have $y_1 = 1$. Now we define the following formulas i, for each $i \in [1, l]$:

$$\mathsf{value}_1 \quad = \quad \langle \mathsf{set}_1 \rangle \langle \mathsf{get}_1(1) \rangle \mathtt{true}$$

and for each $i \in [2, l]$ we define

$$\mathsf{value}_i \;=\; \mathsf{value}_{i-1} \wedge \langle \mathsf{set}_i \rangle \begin{cases} \langle \mathsf{get}_i(1) \rangle \mathtt{true} & \text{if } y_i = 1 \\ \langle \mathsf{get}_i(1) \rangle \mathtt{true} \leftrightarrow \neg \langle \mathsf{get}_j(1) \rangle \mathtt{true} & \text{if } y_i = \neg x_j \\ \langle \mathsf{get}_i(1) \rangle \mathtt{true} \leftrightarrow (\langle \mathsf{get}_j(1) \rangle \mathtt{true} \wedge \langle \mathsf{get}_h(1) \rangle \mathtt{true}) & \text{if } y_i = x_j \wedge x_h \\ \langle \mathsf{get}_i(1) \rangle \mathtt{true} \leftrightarrow \langle \mathbb{A}^* \rangle (\psi(S_0, n) \wedge \mathsf{input}(S_0, w)) & \text{if } y_i = \mathbb{D}(w) \end{cases}$$

where the formula $\mathsf{input}(S_0, w)$ needs to adjusted in such a way that the values each of the x_j that appear in w are reflected in the tree, that is reached via $\xrightarrow{\mathbb{A}^*}$, in the appropriate superleaf. Details are left to the reader. We put $\varphi' = \mathsf{value}_l$. Observe that $|\varphi'|$ can be computed from $\mathcal{C}$ in polynomial time. We have

$$\mathsf{eval}(\mathcal{C}) = 1 \quad \text{if and only if} \quad (\mathfrak{S}(\mathcal{R}'), T'_0) \models \varphi'.$$

□

4 Strong bisimilarity against finite systems

Let us first state upper bound for strong bisimilarity checking.

Theorem 23. *Strong bisimilarity checking of RGTRS against finite systems is decidable in time* $\mathsf{TOWER}(O(n))$.

Proof. By Proposition 3 strong bisimilarirty checking against finite systems can be reduced to EF model checking of formulas in DAG-representation. By Theorem 2 EF model checking is decidable in time $\mathsf{TOWER}(O(n))$. □

However, for GTRS we can prove an elementary upper bound for strong bisimilarity checking against finite systems.

Theorem 24. *Strong bisimilarity checking of GTRS against finite systems is in* coNEXP.

Proof. Let $\mathcal{R} = (A, \mathbb{A}, R)$ be a GTRS, let $T \in \mathsf{Trees}_A$ be an initial tree, let $\mathfrak{F} = (C, \mathbb{A}, \{\xrightarrow{a}_{\mathfrak{F}} | \ a \in \mathbb{A}\})$ be a finite system and let $c \in C$ be an initial configuration of $\mathfrak{F}$. By Proposition 3 we can compute in polynomial time EF_0 formulas φ_1, φ_2 in DAG-representation such that

$$T \sim c \quad \text{if and only if} \quad \mathfrak{S}(\mathcal{R}) \models (\varphi_1 \wedge [\mathbb{A}^*]\varphi_2).$$

By a simple polynomial space procedure one can even check if $(\mathfrak{S}(\mathcal{R}), T) \models \varphi_1$ holds. By Theorem 9 checking $(\mathfrak{S}(\mathcal{R}), T) \models [\mathbb{A}^*]\varphi_2$ is in coNEXP. Hence checking if $(\mathfrak{S}(\mathcal{R}), T) \models \varphi_1 \wedge [\mathbb{A}^*]\varphi_2$ holds, is in coNEXP. □

As a main result of this section we prove that strong bisimilarity checking between a RGTRS and a finite system has nonelementary complexity. We reuse some of the notation that was introduced in Section 3. Again, let us fix a first-order sentence interpreted over binary words

$$\psi \quad = \quad \exists x_1 \forall x_2 \ \ldots \ \exists x_{2n-1} \forall x_{2n} \varphi(x_1, \ldots, \varphi_{2n})$$

and let us assume again $\overline{w} \not\models \psi$ for each binary word w with $|w| < 2$. Our goal is to compute in exponential time a RGTRS $\mathcal{R} = (A, \mathbb{A}, R)$, some initial tree $\mathsf{start} \in \mathsf{Trees}_A$, some finite transition system $\mathfrak{F} = (C, \mathbb{A}, \{\xrightarrow{a}_{\mathfrak{F}} | \ a \in \mathbb{A}\})$, and a configuration $s_\emptyset \in C$ such that

$$\exists w \in \{0,1\}^* : \ \overline{w} \models \varphi \qquad \text{if and only if} \qquad \mathsf{start} \not\sim s_\emptyset.$$

We call a subset $I \subseteq [1, 2n]$ *game-conform* if $I = [1, k]$ for some $k \in [0, 2n]$. Any other set $I \subseteq [1, 2n]$ is called *non-game-conform*. Analogously, we call a comb $T \in \mathsf{Combs}_I$, where $I \subseteq [1, 2n]$, *game-conform* (resp. *non-game-conform*) if I is game-conform (resp. non-game-conform). Note that the empty set is game-conform. Let us fix some game-conform subset $I = [1, k]$, i.e. $k \in [0, 2n]$, and some binary word $w = w_1 \cdots w_l$ that we want to know whether it satisfies ψ. An *I-assignment of w* is a mapping $\nu : I \to [1, l]$ that assigns each variable from the index set I some position in w. The *ν-restriction* $\varphi[\nu]$ of φ is obtained from φ by relacing, for each $i \in I$,

- each occurrence of $P_0(x_i)$ by `true` if $w_{\nu(i)} = 0$ and by `false` if $w_{\nu(i)} = 1$.
- each occurrence of $P_1(x_i)$ by `true` if $w_{\nu(i)} = 1$ and by `false` if $w_{\nu(i)} = 0$.
- each occurrence of x_i that occurs next to '$<$' by $\nu(x_i)$.

We analogously define

$$\psi[\nu] \quad = \quad Q_{k+1} x_{k+1} \cdots Q_{2n} x_{2n} \ \varphi[\nu], \quad \text{where} \begin{cases} Q_i = \exists & \text{if } i \text{ is odd and} \\ Q_i = \forall & \text{if } i \text{ is even.} \end{cases}$$

In particular, note that in case $I = [1, 2n]$ we have $\psi[\nu] = \varphi[\nu]$.

In case $I \subset [1, 2n]$ and $i \in [1, 2n] \setminus I$ we say a tree $T' \in \mathsf{Combs}_{I \cup \{i\}}$ is an *i-extension* of T if T' can be obtained from T by choosing exactly one leaf x and replacing its label $T(x) = (J, \alpha)$ by $(J \cup \{i\}, \alpha)$. Recall that by Combs_φ (resp. $\mathsf{Combs}_{\overline{\varphi}}$) we denote the trees from $\mathsf{Combs}_{[1,2n]}$ whose word and variable assignment interpretation satisfies (resp. does not satisfy) φ. Recall that every $T \in \mathsf{Combs}_I$ corresponds to a unique pair (w, ν), where w is a binary word and $\nu = \nu_T$ is an I-assignment of w that is inherited from T. Thus, we can write $T \models \psi[\nu_T]$ to mean that the yield string of T satisfies ψ under the valuation ν_T.

In our finite system $\mathfrak{F}$ for each game-conform I we have *two* configurations s_I and $\overline{s_I}$. For each non-game-conform I we have *one* corresponding configuration u_I in $\mathfrak{F}$. In addition our finite system $\mathfrak{F}$ has the configurations succ and fail. We define the set of actions as $\mathbb{A} = \{a_i \mid i \in [1, 2n]\} \cup \{\varphi\}$.

4.1 The idea of the bisimulation game and difficulties that arise

The high level idea of the strong bisimulation game goes as follows, and uses Defender's forcing techniques as e.g. in [13]:

(0) Attacker plays $\mathsf{start} \xrightarrow{a_1} T$ in the infinite system $\mathfrak{S}(\mathcal{R})$ and hereby chooses a comb T from $\mathsf{Combs}_{[1,1]}$ for which he claims that $T \models \psi[\nu_T]$ holds. Defender can only respond $s_\emptyset \xrightarrow{a_1}_{\mathfrak{F}} s_{[1,1]}$ in $\mathfrak{F}$. Hence the new pebble configuration is $(T, s_{[1,1]})$.

We repeat the following round, where the current pebble configuration is $(T, s_{[1,k]})$ where $T \in \mathsf{Combs}_{[1,k]}$ for each round $k = 1, \ldots, 2n-1$:

($\forall$) If k is odd, then Attacker is supposed to move in $\mathfrak{F}$, namely $s_{[1,k]} \xrightarrow{a_{k+1}}_{\mathfrak{F}} s_{[1,k+1]}$ although the move $s_{[1,k]} \xrightarrow{a_{k+1}}_{\mathfrak{F}} \overline{s_{[1,k+1]}}$ is possible. Defender is now forced to move in $\mathfrak{S}(\mathcal{R})$, namely $T \xrightarrow{a_{k+1}} T'$ for some $k+1$-extension T' of T. This response corresponds to the universal quantification $\forall x_{k+1}$ in ψ.

($\exists$) If k is even, then Attacker is supposed to move in $\mathfrak{S}(\mathcal{R})$, namely $T \xrightarrow{a_{k+1}} T'$ for some $k+1$-extension T' of T. This move corresponds to the existential quantification $\exists x_{k+1}$ in ψ. Defender's only possible response in $\mathfrak{F}$ is $s_{[1,k]} \xrightarrow{a_{k+1}}_{\mathfrak{F}} s_{[1,k+1]}$.

Finally, when we are in the pebble configuration $(T, s_{[1,2n]})$, where $T \in \mathsf{Combs}_{[1,2n]}$, the action φ can be performed that allows Attacker to win (via a rule in $\mathcal{R}$ that contains Combs_φ on the left-hand side) if and only if $T \models \varphi[\nu_T]$.

When trying to implement such a game, several obstacles arise. Let us discuss these obstacles for the step ($\forall$) and give solutions to them: In step ($\forall$), how can we force Attacker make in $\mathfrak{F}$ the move $s_{[1,k]} \xrightarrow{a_{k+1}}_{\mathfrak{F}} s_{[1,k+1]}$?

1. **Difficulty:** What if Attacker moves $s_{[1,k]} \xrightarrow{a_{k+1}}_{\mathfrak{F}} \overline{s_{[1,k+1]}}$ (which will exist in $\mathfrak{F}$)? **Solution:** We add the rule $\mathsf{Combs}_{[1,k]} \overset{a_{k+1}}{\hookrightarrow} \overline{s_{[1,k+1]}}$ to $\mathcal{R}$ such that Defender has the possibility to establish syntactic equivalence by responding $T \xrightarrow{a_{k+1}} \overline{s_{[1,k+1]}}$ in $\mathfrak{S}(\mathcal{R})$ and hence wins.
2. **Difficulty:** What if Attacker moves in $\mathfrak{S}(\mathcal{R})$, namely $T \xrightarrow{a_{k+1}} T'$ for some $k+1$-extension of T rather that playing in $\mathfrak{F}$? **Solution:** Defender can react in $\mathfrak{F}$, depending on whether $T' \models \psi[\nu_{T'}]$ or $T' \not\models \psi[\nu_{T'}]$. In case $T' \models \psi[\nu_{T'}]$ she can move to $\overline{s_{[1,k+1]}}$ and can win. In case $T' \not\models \psi[\nu_{T'}]$ she can move to $s_{[1,k]}$ and can win.
3. **Difficulty:** What if Attacker plays $T \xrightarrow{a_i} T'$ where $i \in [1,k]$? I.e. Attacker plays an action that has already been played (i.e. assigns a variable in the tree that has already been assigned). **Solution:** We allow a simple transition to the configuration $s_{[1,2n]}$ in $\mathfrak{F}$ from which Defender can surely win.
4. **Difficulty:** What if Attacker plays in $T \xrightarrow{a_i} T'$ in $\mathfrak{S}(\mathcal{R})$ where $i > k+1$? I.e. Attacker deviates from playing a sequence of actions $a_1 \cdots a_k$ that correspond to assigning variables to the tree. We also say that the current pebble configuration is non-game-conform. **Solution:** We allow in $\mathfrak{F}$ a special transition for Defender $s_{[1,k]} \xrightarrow{a_i} u_{[1,k]\cup\{i\}}$ that allows her to win.

The solutions to Difficulty 1 and 2 are standard and are similar to a technique elaborated in [13]. The solution to Difficulty 3 is straightforward. The real difficulty *in the absence of a finite control* (pushdown systems have a finite control) in the game is Difficulty 4. We have to provide configurations in $\mathfrak{F}$ that allow to remember the set of variables in the current tree T that have been assigned. The difficulty that now arises is that Attacker can continue labeling leafs in T and pretend some moves later that the current tree T is game-conform all of a sudden (and hence threaten to play the above-mentioned punishing moves for instance). We have to carefully design transitions in $\mathfrak{F}$ that sooner or later punish Attacker since he was the one who deviated from playing game-conform.

4.2 The finite system

We now define the outgoing transitions of $s_{[1,k]}$ and of $\overline{s_{[1,k]}}$, for each possible $k \in [0, 2n-1]$

$$
\begin{array}{lll}
s_{[1,k]} \xrightarrow{a_{k+1}}_{\mathfrak{F}} s_{[1,k+1]} & & \\
\overline{s_{[1,k]}} \xrightarrow{a_{k+1}}_{\mathfrak{F}} \overline{s_{[1,k+1]}} & & \\
s_{[1,k]} \xrightarrow{a_{k+1}}_{\mathfrak{F}} \overline{s_{[1,k+1]}} & & \text{if } k \text{ is odd} \\
\overline{s_{[1,k]}} \xrightarrow{a_{k+1}}_{\mathfrak{F}} s_{[1,k+1]} & & \text{if } k \text{ is even} \\
s_{[1,k]} \xrightarrow{a_i}_{\mathfrak{F}} s_{[1,2n]} & & \text{for each } i \in [1,k] \\
\overline{s_{[1,k]}} \xrightarrow{a_i}_{\mathfrak{F}} s_{[1,2n]} & & \text{for each } i \in [1,k] \\
s_{[1,k]} \xrightarrow{a_i}_{\mathfrak{F}} u_{[1,k]\cup\{i\}} & & \text{for each } i \in [k+2,2n] \\
\overline{s_{[1,k]}} \xrightarrow{a_i}_{\mathfrak{F}} u_{[1,k]\cup\{i\}} & & \text{for each } i \in [k+2,2n] \\
s_{[1,2n]} \xrightarrow{\varphi}_{\mathfrak{F}} \mathsf{succ} & & \\
\overline{s_{[1,2n]}} \xrightarrow{a_i}_{\mathfrak{F}} s_{[1,2n]} & & \text{for each } i \in [1,2n] \\
\overline{s_{[1,2n]}} \xrightarrow{\varphi}_{\mathfrak{F}} \mathsf{fail} & &
\end{array}
$$

Moreover we have the following transition

$$\mathsf{fail} \xrightarrow{a_i}_{\mathfrak{F}} \mathsf{fail} \qquad \text{for each } i \in [1,2n].$$

We now define the outgoing transitions of u_I for each non-game-conform $I \subseteq [1,2n]$:

$$
\begin{array}{lll}
u_I \xrightarrow{a_i}_{\mathfrak{F}} s_{[1,2n]} & & \text{for each } i \in I \\
u_I \xrightarrow{a_i}_{\mathfrak{F}} u_{I\cup\{i\}} & & \text{for each } i \notin I \text{ for which } I \cup \{i\} \text{ is non-game-conform} \\
u_I \xrightarrow{a_i}_{\mathfrak{F}} s_{I\cup\{i\}} & & \text{for each } i \notin I \text{ for which } I \cup \{i\} \text{ is game-conform} \\
u_I \xrightarrow{a_i}_{\mathfrak{F}} \overline{s_{I\cup\{i\}}} & & \text{for each } i \notin I \text{ for which } I \cup \{i\} \text{ is game-conform}
\end{array}
$$

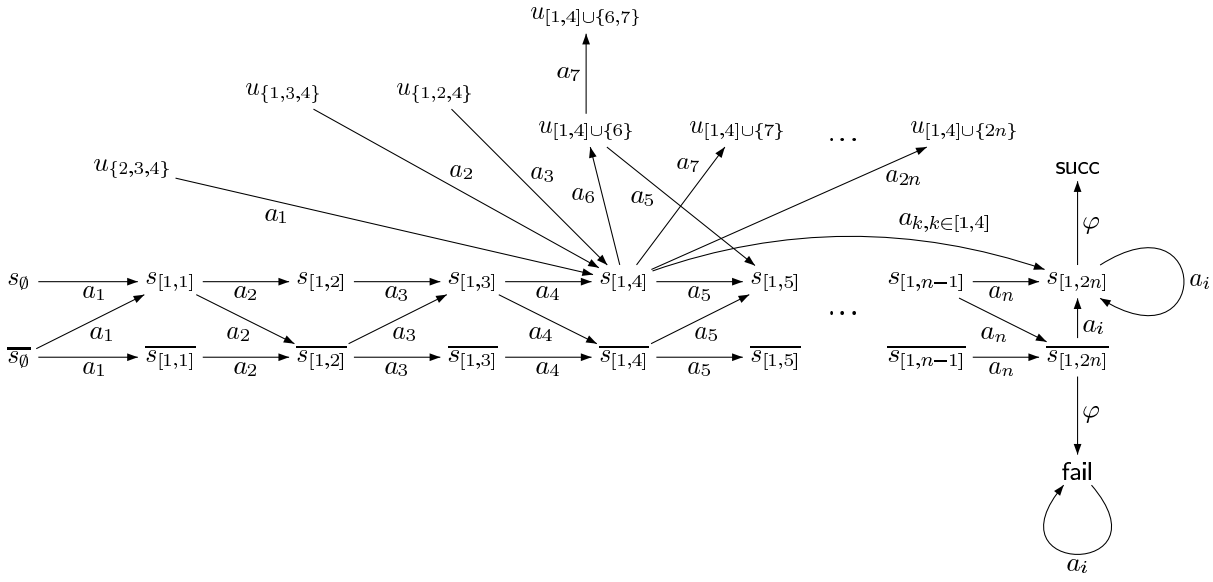


Fig. 2. A snapshot of $\mathfrak{F}$ and the outgoing transitions of $s_{[1,4]}$ in the strong bisimulation game (i ranges over $[1,2n]$).

4.3 The infinite system

Recall that C denotes the set of configurations of $\mathfrak{F}$ and that P denotes the set of proper leaf labels from Definition 5, i.e. $P = ((2^{[1,2n]} \cup \{\bot\}) \times \{0,1\})$. We define the ranked alphabet $A = (A_i)_{i\in\{0,1,2\}}$ of $\mathcal{R}$ as follows:

$$\begin{aligned} A_0 &= \{\mathsf{start}\} \cup P \cup C \\ A_1 &= \{\mathsf{root}\} \\ A_2 &= \{\bigstar\} \end{aligned}$$

We note that the only relevant trees (i.e. configurations) in $\mathfrak{S}(\mathcal{R})$ in our reduction whose leafs are labeled with C are *singleton trees*. More precisely, we will have that every such singleton tree c will be bisimilar to c in the finite system $\mathfrak{F}$. Hence, to R we add the rewriting rules

$$c \overset{b}{\hookrightarrow} c' \qquad \text{for each transition } c \xrightarrow{b}_{\mathfrak{F}} c' \text{ in } \mathfrak{F}.$$

We add the following leaf rewriting rules to R:

$$\begin{array}{lll} (I,\alpha) \overset{a_i}{\hookrightarrow} (I \cup \{i\},\alpha) & & \text{for each } i \in [1,2n] \setminus I \text{ and for each } \alpha \in \{0,1\} \\ (I,\alpha) \overset{a_i}{\hookrightarrow} (\bot,\alpha) & & \text{for each } i \in I \text{ and for each } \alpha \in \{0,1\} \\ (\bot,\alpha) \overset{a_i}{\hookrightarrow} (\bot,\alpha) & & \text{for each } i \in I \text{ and for each } \alpha \in \{0,1\} \end{array}$$

The first rewriting rule allows to mimic the assignment of a variable x_i to the current leaf (which corresponds to some position of the word that is interpreted as a tree). The latter two rules label the current leaf with $\bot$ in order to mimic that the current leaf is already assigned with this variable or the leaf already is mal-formatted.

We define the regular tree language $\mathsf{Combs}_\bot = \mathsf{Combs} \setminus \bigcup_{I\subseteq[1,2n]} \mathsf{Combs}_I$. In other words, $\mathsf{Combs}_\bot$ consists of those combs $T \in \mathsf{Combs}$ that satisfy at least one of the following conditions:

- There is some leaf x of T with $T(x) \in \{\bot\} \times \{0,1\}$.
- There are two distinct leaves x, x' of T with $T(x) = (J,\alpha)$ and $T(x') = (J',\alpha')$ such that $J \cap J' \neq \emptyset$.

Let us add for each possible $I \subseteq [1,2n]$ and $k \in [0,2n-1]$, the following rules to R:

$$\begin{array}{lll} \mathsf{Combs}_{[1,k]} \overset{a_{k+1}}{\hookrightarrow} \overline{s_{[1,k+1]}} & & \text{if } k \text{ is odd} \\ \mathsf{Combs}_{[1,k]} \overset{a_{k+1}}{\hookrightarrow} s_{[1,k+1]} & & \text{if } k \text{ is even} \\ \mathsf{Combs}_{[1,k]\setminus\{i\}} \overset{a_i}{\hookrightarrow} s_{[1,k]} & & \text{for each } i \in [1,k-2] \\ \mathsf{Combs}_{[1,k]\setminus\{i\}} \overset{a_i}{\hookrightarrow} \overline{s_{[1,k]}} & & \text{for each } i \in [1,k-2] \\ \mathsf{Combs}_{I\setminus\{i\}} \overset{a_i}{\hookrightarrow} u_I & & \text{for each } i \in I \text{ if } I \text{ is non-game-conform} \\ \mathsf{Combs}_I \overset{a_i}{\hookrightarrow} s_{[1,2n]} & & \text{for each } i \in I \\ \mathsf{Combs}_\varphi \overset{\varphi}{\hookrightarrow} \mathsf{fail} & & \\ \mathsf{Combs}_{\overline{\varphi}} \cup \mathsf{Combs}_\bot \overset{\varphi}{\hookrightarrow} \mathsf{succ} & & \end{array}$$

4.4 The correctness proof

Lemma 25. *For every $T \in \mathsf{Combs}_{\overline{\varphi}} \cup \mathsf{Combs}_\bot$ we have $T \sim s_{[1,2n]}$.*

Proof. Let $X = \mathsf{Combs}_{\overline{\varphi}} \cup \mathsf{Combs}_\bot$. One easily verifies that

$$\big((X \cup \{s_{[1,2n]}\}) \times \{s_{[1,2n]}\}\big) \quad \cup \quad (\{\mathsf{succ}\} \times \{\mathsf{succ}\})$$

is a strong bisimulation that relates $T \in X$ and $s_{[1,2n]}$. □

The following lemma is the central correctness proof of our reduction.

Lemma 26. *Let $I \subseteq [1, 2n]$.*

1. *If I is game-conform, then the following three statements hold:*
   (a) $s_I \not\sim \overline{s_I}$.
   (b) $\forall T \in \mathsf{Combs}_I$*:* $T \not\sim s_I$ *if and only if* $T \models \psi[\nu_T]$.
   (c) $\forall T \in \mathsf{Combs}_I$*:* $T \sim \overline{s_I}$ *if and only if* $T \models \psi[\nu_T]$.
2. *If I is non-game-conform, then for each $T \in \mathsf{Combs}_I$ we have $T \sim u_I$.*

Proof. We prove the lemma by downward induction on $|I| = 2n, 2n-1, \ldots$.

Induction base. Let $|I|$ be maximal, i.e. $I = [1, 2n]$ and so I is game-conform. Thus, we only have to prove Point 1.

a) We have to prove that $s_{[1,2n]} \not\sim \overline{s_{[1,2n]}}$. Attacker moves from $s_{[1,2n]} \xrightarrow{\varphi}_{\mathfrak{F}} \mathsf{succ}$ and hence reaches a dead-end. Defender can only respond with $\overline{s_{[1,2n]}} \xrightarrow{\varphi}_{\mathfrak{F}} \mathsf{fail}$ and does not reach a dead-end. Thus $s_{[1,2n]} \not\sim \overline{s_{[1,2n]}}$.

b) Let $T \in \mathsf{Combs}_{[1,2n]}$. On the one hand, assume $T \models \psi[\nu_T]$ or equivalently $T \models \varphi[\nu_T]$. Hence, $T \in \mathsf{Combs}_\varphi$ by definition, so Attacker can move $T \xrightarrow{\varphi} \mathsf{fail}$ in $\mathfrak{S}(\mathcal{R})$ and thus reaches a configuration that is not a dead-end, whereas Defender can only respond with $s_{[1,2n]} \xrightarrow{\varphi}_{\mathfrak{F}} \mathsf{succ}$ in F which is a dead-end. Hence $T \not\sim s_{[1,2n]}$. On the other hand, assume $T \not\models \varphi[\nu_T]$. Hence $T \in \overline{\mathsf{Combs}_\varphi}$. By Lemma 25 we have $T \sim s_{[1,2n]}$.

c) Let $T \in \mathsf{Combs}_{[1,2n]}$. On the one hand assume $T \models \varphi[\nu_T]$. Hence, $T \in \mathsf{Combs}_\varphi$. If Attacker plays $T \xrightarrow{\varphi} \mathsf{fail}$ in $\mathfrak{S}(\mathcal{R})$, then Defender responds in F by playing $\overline{s_{[1,2n]}} \xrightarrow{\varphi}_{\mathfrak{F}} \mathsf{fail}$. Analogously, if Attacker plays $\overline{s_{[1,2n]}} \xrightarrow{\varphi}_{\mathfrak{F}} \mathsf{fail}$, then Defender responds $T \xrightarrow{\varphi} \mathsf{fail}$. If Attacker moves $T \xrightarrow{a_i} T'$ in $\mathfrak{S}(\mathcal{R})$, then surely $T' \in \mathsf{Combs}_\perp$. Thus, Defender can respond with $\overline{s_{[1,2n]}} \xrightarrow{a_i}_{\mathfrak{F}} s_{[1,2n]}$ in F and hence establishes the pair $(T', s_{[1,2n]})$ which is bisimilar by Lemma 25. Analogously, if Attacker plays $\overline{s_{[1,2n]}} \xrightarrow{a_i}_{\mathfrak{F}} s_{[1,2n]}$ in $\mathfrak{F}$, then Defender can always move $T \xrightarrow{a_i} T'$ in $\mathfrak{S}(\mathcal{R})$ for some $T' \in \mathsf{Combs}_\perp$. Thus, Defender wins analogously.

Induction step. Let $I \subset [1, 2n]$.

1. Let us assume that I is game-conform, i.e. $I = [1, k]$ for some $k \in [2n-1]$.
   (a) If k is odd, then Attacker plays $s_{[1,k]} \xrightarrow{a_{k+1}}_{\mathfrak{F}} s_{[1,k+1]}$ and Defender can only respond with $\overline{s_{[1,k]}} \xrightarrow{a_{k+1}}_{\mathfrak{F}} \overline{s_{[1,k+1]}}$. By induction hypothesis we have $s_{[1,k+1]} \not\sim \overline{s_{[1,k+1]}}$, so $s_{[1,k]} \not\sim \overline{s_{[1,k]}}$. If k is even, then Attacker plays $\overline{s_{[1,k]}} \xrightarrow{a_{k+1}}_{\mathfrak{F}} \overline{s_{[1,k+1]}}$ and Defender can only respond with $s_{[1,k]} \xrightarrow{a_{k+1}}_{\mathfrak{F}} s_{[1,k+1]}$. By induction hypothesis we have $s_{[1,k+1]} \not\sim \overline{s_{[1,k+1]}}$, so $s_{[1,k]} \not\sim \overline{s_{[1,k]}}$.
   (b) Let $T \in \mathsf{Combs}_{[1,k]}$. We make a case distinction on the parity of k.
   - k is odd: Then recall that

$$\psi[\nu_T] \quad = \quad \forall x_{k+1} \exists x_{k+2} \cdots \forall x_{2n} \varphi[\nu_T].$$

   On the one hand, assume $T \models \psi[\nu_T]$. Hence, in other words, for each $k+1$-extension T' of T we have $T' \models \psi[\nu_{T'}]$. Attacker can now play $s_{[1,k]} \xrightarrow{a_{k+1}}_{\mathfrak{F}} s_{[1,k+1]}$. Defender cannot play $T \xrightarrow{a_{k+1}} \overline{s_{[k+1]}}$ since $s_{[1,k+1]} \not\sim \overline{s_{[1,k+1]}}$ by Point 1(a) of induction hypothesis. Defender only has the possibility to respond $T \xrightarrow{a} T'$ in $\mathfrak{S}(\mathcal{R})$ for some $k+1$-extension T' of T. Since $T' \models \psi[\nu_{T'}]$ we have $T' \sim s_{[1,k+1]}$ by Point 1(b) of induction hypothesis, hence $T \not\sim s_{[1,k]}$.

   On the other hand, assume $T \not\models \psi[\nu_T]$. Thus, there is some extension $T' \in \mathsf{Combs}_{[k+1]}$ of T such that $T' \not\models \psi[\nu_{T'}]$. We have to show that $T \sim s_{[1,k]}$.

* If Attacker plays $s_{[1,k]} \xrightarrow{a_{k+1}}_{\mathfrak{F}} s_{[1,k+1]}$, then Defender responds $T \xrightarrow{a_{k+1}} T'$ and wins by Point 1(b) of induction hypothesis. Conversely, if Attacker plays $T \xrightarrow{a_{k+1}} T''$ for some extension $T'' \in \mathsf{Combs}_{[1,k+1]}$ of T, we distinguish two cases. In case $T'' \not\models \psi[\nu_{T''}]$, then Defender plays $s_{[1,k]} \xrightarrow{a_{k+1}}_{\mathfrak{F}} s_{[1,k+1]}$ and wins by Point 1(b) of induction hypothesis. In case $T'' \models \psi[\nu_{T''}]$, then Defender responds $s_{[1,k]} \xrightarrow{a_{k+1}}_{\mathfrak{F}} \overline{s_{[1,k+1]}}$ and wins by Point 1(c) of induction hypothesis.
* If Attacker plays $s_{[1,k]} \xrightarrow{a_{k+1}}_{\mathfrak{F}} \overline{s_{[1,k+1]}}$, then Defender plays $T \xrightarrow{a_{k+1}} \overline{s_{[1,k+1]}}$ in $T(\mathcal{R})$ and vice versa. Defender wins by establishing syntactic equivalence.
* If Attacker plays $s_{[1,k]} \xrightarrow{a_i}_{\mathfrak{F}} s_{[1,2n]}$ for some $i \in [1,k]$, then Defender responds $T \xrightarrow{a_i} T''$ for some $T'' \in \mathsf{Combs}_{\bot}$ by labeling a leaf of T and vice versa. By Lemma 25 we have $T'' \sim s_{[1,2n]}$, hence Defender wins.
* If Attacker plays $s_{[1,k]} \xrightarrow{a_i}_{\mathfrak{F}} u_{[1,k]\cup\{i\}}$ for some $i \in [k+2, 2n]$, then Defender responds $T \xrightarrow{a_i} u_{[1,k]\cup\{i\}}$ and vice versa. Hence Defender by establishing syntactic equivalence.
* If Attacker plays $T \xrightarrow{a_i} T''$ for some i-extension $T' \in \mathsf{Combs}_{[1,k]\cup\{i\}}$ of T where $i \in [k+2, 2n]$, then Defender responds $s_{[1,k]} \xrightarrow{a_i}_{\mathfrak{F}} u_{[1,k]\cup\{i\}}$ and vice versa. By Point 2 of induction hypothesis, we have $T'' \sim u_{[1,k]}$, hence Defender wins.

- k is even: Then recall that

$$\psi[\nu_T] \quad = \quad \exists x_{k+1} \forall x_{k+2} \cdots \exists x_{2n-1} \forall x_{2n} \varphi[\nu_T].$$

On the one hand, assume $T \not\models \psi[\nu_T]$. Hence, in other words, for every $k+1$-extension $T' \in \mathsf{Combs}_{[1,k+1]}$ of T we have $T' \not\models \psi[\nu_{T'}]$. We have to prove $T \not\sim \overline{s_{[1,k]}}$. Attacker can now play $\overline{s_{[1,k]}} \xrightarrow{a_{k+1}} \overline{s_{[1,k+1]}}$. If Defender responds with $T \xrightarrow{a_{k+1}} s_{[1,k+1]}$ then she loses, since $s_{[1,k+1]} \not\sim \overline{s_{[1,k+1]}}$ by Point 1(a) of induction hypothesis. If Defender responds with $T \xrightarrow{a_{k+1}} T'$ for some $k+1$-extension $T' \in \mathsf{Combs}_{[1,k+1]}$ of T, then she loses as well since due to $T' \not\models \psi[\nu_{T'}]$ we have $T' \not\sim \overline{s_{[1,k+1]}}$ by Point 1(c) of induction hypothesis. Hence $T \not\sim \overline{s_{[1,k]}}$.

On the other hand, assume $T \models \psi[\nu_T]$. Thus, there is a $k+1$-extension $T' \in \mathsf{Combs}_{[k+1]}$ of T such that $T' \models \psi[\nu_{T'}]$. We have to show $T \sim \overline{s_{[1,k]}}$.

* If Attacker plays $\overline{s_{[1,k]}} \xrightarrow{a_{k+1}}_{\mathfrak{F}} \overline{s_{[1,k+1]}}$, then Defender plays $T \xrightarrow{a_{k+1}} T'$ and wins by Point 1(c) of induction hypothesis.
  Conversely, if Attacker plays $T \xrightarrow{a_{k+1}} T''$ for some $k+1$-extension $T'' \in \mathsf{Combs}_{[1,k+1]}$ of T, we distinguish two cases. In case $T'' \not\models \psi[\nu_{T''}]$ Defender responds $\overline{s_{[1,k]}} \xrightarrow{a_{k+1}}_{\mathfrak{F}} s_{[1,k+1]}$ and wins by Point 1(b) of induction hypothesis. In case $T'' \models \psi[\nu_{T''}]$, then Defender responds $\overline{s_{[1,k]}} \xrightarrow{a_{k+1}}_{\mathfrak{F}} \overline{s_{[1,k+1]}}$ and wins by Point 1(c) of induction hypothesis.
* If Attacker plays $\overline{s_{[1,k]}} \xrightarrow{a_{k+1}}_{\mathfrak{F}} s_{[1,k+1]}$, then Defender responds $T \xrightarrow{a_{k+1}} s_{[1,k+1]}$ and vice versa. Defender hence by establishing syntactic equivalence.
* If Attacker plays $\overline{s_{[1,k]}} \xrightarrow{a_i}_{\mathfrak{F}} s_{[1,2n]}$ for some $i \in [1,k]$, then Defender responds $T \xrightarrow{a_i} T''$ for some $T'' \in \mathsf{Combs}_{\bot}$ by labeling a leaf of T and vice versa. By Lemma 25 we have $T'' \sim s_{[1,2n]}$, hence Defender wins.
* If Attacker plays $\overline{s_{[1,k]}} \xrightarrow{a_i}_{\mathfrak{F}} u_{[1,k]\cup\{i\}}$ for some $i \in [k+2, 2n]$, then Defender responds $T \xrightarrow{a_i} u_{[1,k]\cup\{i\}}$ and vice versa. Hence Defender wins by establishing syntactic equivalence.
* If Attacker plays $T \xrightarrow{a_i} T''$ for some $T'' \in \mathsf{Combs}_{[1,k]\cup\{i\}}$ where $i \in [k+2, 2n]$, then Defender responds $\overline{s_{[1,k]}} \xrightarrow{a_i}_{\mathfrak{F}} u_{[1,k]\cup\{i\}}$ and vice versa. Defender wins by Point 2 of induction hypothesis.

(c) This case can be proven analogously as (b).

2. Let $I \subseteq [1,k]$ be non-game-conform and let $T \in \mathsf{Combs}_I$. We have to prove that $T \sim u_I$.

- If Attacker moves $T \xrightarrow{a_i} T'$ for some $i \in I$ and some $T' \in \mathsf{Combs}_\perp$, then Defender responds with $u_I \xrightarrow{a_i}_{\mathfrak{F}} s_{[1,2n]}$ and vice versa. Defender wins by Lemma 25.
- If Attacker moves $T \xrightarrow{a_i} T'$ where $T' \in \mathsf{Combs}_{I\cup\{i\}}$ is an i-extension of T, where $i \in [1,2n] \setminus I$ and $I \cup \{i\}$ is non-game-conform, then Defender responds with $u_I \xrightarrow{a_i}_{\mathfrak{F}} u_{I\cup\{i\}}$ and vice versa. Defender wins by Point 2 of induction hypothesis.
- If Attacker plays $u_I \xrightarrow{a_i}_{\mathfrak{F}} s_{[1,k]}$ (resp. $u_I \xrightarrow{a_i}_{\mathfrak{F}} \overline{s_{[1,k]}}$), i.e. in particular $I \cup \{i\}$ is game-conform, then Defender responds with $T \xrightarrow{a_i} s_{[1,k]}$ (resp. $T \xrightarrow{a_i} \overline{s_{[1,k]}}$) and vice versa. Hence Defender wins by establishing syntactic equivalence.
- If Attacker plays $T \xrightarrow{a_i} T'$ for some i-extension $T' \in \mathsf{Combs}_{[1,k]}$ of T, i.e. $I \cup \{i\} = [1,k]$ is game-conform, then Defender responds $u_I \xrightarrow{a_i}_{\mathfrak{F}} s_{[1,k]}$ in case $T' \models \psi[\nu_{T'}]$ and with $u_I \xrightarrow{a_i}_{\mathfrak{F}} \overline{s_{[1,k]}}$ in case $T' \not\models \psi[\nu_{T'}]$. In the former case Defender wins since $T' \sim s_{[1,k]}$ by Point 1(b) of induction hypothesis. In the latter case Defender wins since $T' \sim \overline{s_{[1,k]}}$ by Point 1(c) of induction hypothesis.

□

Finally, we add the following two rules to R:

$$\mathsf{start} \overset{a_1}{\hookrightarrow} s_{[1,1]}$$
$$\mathsf{start} \overset{a_1}{\hookrightarrow} \mathsf{Combs}_{[1,1]}$$

Theorem 27. *Strong bisimilarity of RGTRS against finite systems is nonelementary.*

Proof. We first prove correctness of our reduction. We will show

$$\exists T \in \mathsf{Combs}_\emptyset : T \models \psi \quad \text{if and only if} \quad \mathsf{start} \not\sim s_\emptyset.$$

Recall that $\psi = \exists x_1 \forall x_2 \cdots \exists x_{2n-1} \forall x_{2n} \varphi(x_1, \ldots, x_{2n})$.
On the one hand, assume $T \models \psi$ for some $T \in \mathsf{Combs}_\emptyset$. Since $T \models \psi$ there is some 1-extension $T' \in \mathsf{Combs}_{[1,1]}$ of T such that $T' \models \psi[\nu_{T'}]$. Attacker plays $\mathsf{start} \xrightarrow{a_1} T'$. Defender can only respond $s_\emptyset \xrightarrow{a_1}_{\mathfrak{F}} s_{[1,1]}$. Since $T' \models \psi[\nu_{T'}]$ we have $s_{[1,k]} \not\sim T'$ by Point 1(b) of Lemma 26. Hence $\mathsf{start} \not\sim s_\emptyset$.
On the other hand, assume that for all $T \in \mathsf{Combs}_\emptyset$ we have $T \not\models \psi$. This is equivalent to saying that for all $T \in \mathsf{Combs}_{[1,1]}$ we have $T \not\models \psi[\nu_T]$. We have to show $\mathsf{start} \sim s_\emptyset$. If Attacker plays $\mathsf{start} \xrightarrow{a_1} s_{[1,1]}$, then Defender responds $s_\emptyset \xrightarrow{a_1}_{\mathfrak{F}} s_{[1,1]}$ and vice versa. If Attacker plays $\mathsf{start} \xrightarrow{a_1} T$ for some $T \in \mathsf{Combs}_{[1,1]}$, then Defender responds $s_\emptyset \xrightarrow{a_1}_{\mathfrak{F}} s_{[1,1]}$ and vice versa. Since $T \not\models \psi[\nu_T]$ we know that $T \sim s_{[1,1]}$ by Point 1(b) of Lemma 26. Hence $\mathsf{start} \sim s_\emptyset$

Let us now argue that the whole translation is computable in exponential time. The size of the label set $\mathbb{A}$ is linearly bounded in $|\psi|$. The finite system $\mathfrak{F}$ has $\exp(|\psi|)$ many configurations and transitions. The size of the alphabet A of $\mathcal{R}$ is bounded by $\exp(|\psi|)$. In $\mathcal{R}$ we have $\exp(|\psi|)$ many rules that can each be represented by pairs of NTAs each of size $\exp(|\psi|)$. It is straighforward to see that the whole reduction is computable in exponential time. □

5 Weak bisimilarity against finite systems

Before we discuss a nonelementary lower bound for weak bisimilarity of GTRS against finite systems, let us state an upper bound (which follows from known results).

Theorem 28. *Weak bisimilarity checking of GTRS against finite systems is decidable in time* $\mathsf{TOWER}(O(n))$.

Proof. By Proposition 4 weak bisimilarity checking against finite systems can be reduced to EF model checking of formulas in DAG-representation. By Theorem 2 EF model checking is decidable in time $\mathsf{TOWER}(O(n))$. □

The main result of this section is that weak bisimilarity checking of GTRS against finite systems has nonelementary complexity. We reuse definitions and notions that were introduced in Section 4. In analogy to the nonelementary lower bound presented in Section 4 let us fix a first-order sentence interpreted over binary words

$$\psi \quad = \quad \exists x_1 \forall x_2 \ \ldots \ \exists x_{2n-1} \forall x_{2n} \varphi(x_1, \ldots, \varphi_{2n}).$$

As in Section 4, we will assume without loss of generality that $\overline{w} \not\models \psi$ for each binary word w with $|w| < 2$. Our goal is to compute in exponential time a GTRS $\mathcal{R} = (A, \mathbb{A}_\tau, R)$, some initial tree $\mathsf{start} \in \mathsf{Trees}_A \cap A_0$, some finite system $\mathfrak{F} = (C, \mathbb{A}_\tau, \{\xrightarrow{a}_{\mathfrak{F}} | \ a \in \mathbb{A}_\tau\})$, and a configuration $s_\emptyset \in C$ such that

$$\exists w \in \{0,1\}^* : \ \overline{w} \models \varphi \qquad \text{if and only if} \qquad \mathsf{start} \not\approx s_\emptyset.$$

We put $\mathbb{A} = \{a_i \mid i \in [1, 2n]\} \cup \{\varphi, \mathsf{reject}\}$. Recall that $\Delta_\tau = \Delta \cup \{\tau\}$.

Organization of this section In Section 5.1 we discuss the idea behind the overall reduction, name the difficulties that arise (especially in comparison to Section 4), formally define $\mathfrak{F}$ and finally prove that certain configurations of $\mathfrak{F}$ are not weakly bisimilar. In Section 5.2 we add to R some rewriting rules that mimic bottom-up computations of the tree languages of Section 4. In Section 5.3 we prove the main correctness lemma of our reduction. We conclude the reduction in Section 5.4.

5.1 How to design the finite system and why

Recall that in Section 4 we provided in the RGTRS rules of the kind

$$\mathsf{Combs}_{[1,k]} \overset{a_{k+1}}{\hookrightarrow} s_{[1,k+1]}, \quad \mathsf{Combs}_{[1,k]\setminus\{i\}} \xrightarrow{a_i} s_{[1,k]}, \quad \mathsf{Combs}_{I\setminus\{i\}} \xrightarrow{a_i} u_I \quad \text{and} \quad \mathsf{Combs}_\varphi \xrightarrow{\varphi} \mathsf{fail}$$

and the same for $s_{[1,k]}$ (resp. $s_{[1,k+1]}$) replaced by $\overline{s_{[1,k]}}$ (resp. $\overline{s_{[1,k+1]}}$). These rules allowed Defender to punish Attacker in case e.g. Attacker played in $\mathfrak{F}$ rather than in $\mathfrak{S}(\mathcal{R})$ or Attacker deviates from playing game-conform. However, since in GTRS the rewriting rules only allow to rewrite explictly given (sub)trees rather than (trees from) a tree language, we have to find a way way of simulating the computations of tree automata. We follow an obvious approach and simulate each such rewriting rule (such as $\mathsf{Combs}_{[1,k]} \overset{a_{k+1}}{\hookrightarrow} s_{1,k+1}$) by a sequence of τ-transitions in $\mathfrak{S}(\mathcal{R})$ that correspond to a *bottom-up computation* of the respective tree automaton.
Let us fix a, say game-conform, comb T. However, the **major difficulties** that now arise are the following:

1. Due to the *absence* of a finite control unit in our GTRS $\mathcal{R}$ we must provide in the rewriting rules of $\mathcal{R}$ the possibility to simulate from T *all possible* tree automata. Hence Defender must have the possibility to react in $\mathfrak{F}$ to each of such computations.
2. Assume that we construct in our finite system $\mathfrak{F}$ a response for Defender to react to an initialization of a bottom-up computation of a tree language. We have to guarantee in $\mathfrak{F}$ that each two such responses are not weakly-bisimilar in order to distinguish different (intializations of) tree languages.
3. Since we allow the bottom-up computation for Combs_φ, Attacker in theory initialize from T a bottom-up computation for Combs_φ *although* $T \in \mathsf{Combs}_I$ for some strict subset $I \subset [1, 2n]$, i.e. not all of the variables have yet been assigned. In doing so, Attacker might add to T the remaining variables that have not yet been assigned and threaten to label the leaves of the comb *during the bottom-up computation*. We have to provide a mechanism for Defender to win always as soon as Attacker initializes Combs_φ too early.

Let us now define the set of configurations C of our finite system $\mathfrak{F}$.

Firstly, we add the following configurations which mostly already appeared in the configuration set of the finite system of Section 4:

- reject: This configuration mimics that the current bottom-up computation for the languages Combs_J (where $J \subseteq [1, 2n]$) is rejecting.

- succ, fail_1, and fail_2: These configurations can be seen as analogies to the configurations succ and fail of the finite system of Section 4.
- s_I and $\overline{s_I}$ for each game-conform subset $I \subseteq [1, 2n]$: Analogously to the corresponding configurations of the finite transition system of Section 4.
- u_I for each non-game-conform subset $I \subset [1, 2n]$: Analogously to the corresponding configurations of the finite transition system of Section 4.

To simulate the bottom-up computation of these NTA, we provide in R rules that allow us to rewrite the right-most subtree of size three of each comb via some non-silent action. From this point on τ-transitions are executed and eventually allow to pass the computed information to the root and eventually move to some appropriate configuration of the finite system $\mathfrak{F}$. To which configuration of $\mathfrak{F}$ we concretely move, depends on the information of the run on the tree.

The following configurations are designed to overcome the above-mentioned difficulties.

- v_I for each subset $I \subset [1, 2n]$: These configurations allow Defender to react to difficulty 3, i.e. these configurations allow Defender to win always in case Attacker initialized a bottom-up computation for accepting Combs_φ just in case not all of the variables have yet been assigned. So if the bottom-up computation for Combs_φ has just been initialized from some comb from Combs_I (by rewriting the rightmost subtree of size three of this comb) via the action φ, then this initialized tree will be weakly bisimilar to v_I.
- $z_{I,J,l}$ for each subsets $I \subseteq J \subset [1, 2n]$ and each $l \in [1, 2n] \setminus J$: These configurations allow Defender to react (via the action $a_l \in \mathbb{A}$) in case Attacker wants to initialize a bottom-up computation for accepting Combs_J *and* the current comb T belongs to Combs_I. More precisely, the tree from which this bottom-up computation has been initialized (via the action a_l) is from Combs_I, where $I \subseteq J$. This computation might be accepting if additional leaves will be labeled *during* the bottom-up computation. We will have that this initialized tree will be weakly bisimilar to $z_{I,J,l}$.
- $z_\perp$: This configuration is designed to be weakly bisimilar to each $T \in \mathsf{Combs}_\perp$— in particular *every* bottom-up computation that starts in such a T will be rejecting.

Let us define the outgoing transitions of each $z_{I,J,l}$ and reject:

$$
\begin{array}{lll}
z_{J,J,l} \xrightarrow{\tau}_{\mathfrak{F}} \begin{cases} u_{J\cup\{l\}} & \text{if } J \cup \{l\} \text{ is non-game-conform} \\ \overline{s_{[1,k+1]}} & \text{if } J = [1,k] \text{ and } l = k+1 \text{ is even} \\ s_{[1,k+1]} & \text{if } J = [1,k] \text{ and } l = k+1 \text{ is odd} \\ s_{[1,k+1]}, \overline{s_{[1,k+1]}} & \text{if } J = [1,k+1] \setminus \{l\} \text{ and } l \leq k \end{cases} \\
z_{I,J,l} \xrightarrow{a_j}_{\mathfrak{F}} z_{I\cup\{j\},J,l} & & \text{for each } j \in J \setminus I \\
z_{I,J,l} \xrightarrow{\tau}_{\mathfrak{F}} \mathsf{reject} & & \text{for each } I \subset J \\
z_{I,J,l} \xrightarrow{a_k}_{\mathfrak{F}} \mathsf{reject} & & \text{for each } k \in I \cup ([1,2n] \setminus J) \\
\mathsf{reject} \xrightarrow{\mathsf{reject}}_{\mathfrak{F}} \mathsf{reject} & & \\
\mathsf{reject} \xrightarrow{a_i}_{\mathfrak{F}} \mathsf{reject} & & \text{for each } i \in [1,2n]
\end{array}
$$

Next, let us define the outgoing transitions of each v_I, of succ, fail_1, fail_2, and of $z_\perp$:

$$
\begin{array}{lll}
v_I \xrightarrow{a_i}_{\mathfrak{F}} \mathsf{succ} & & \text{for each } i \in I \\
v_{I\setminus\{i\}} \xrightarrow{a_i}_{\mathfrak{F}} v_I & & \text{for each } i \in I \\
v_{[1,2n]\setminus\{i\}} \xrightarrow{a_i}_{\mathfrak{F}} \mathsf{succ} & & \text{for each } i \in [1,2n] \\
v_{[1,2n]\setminus\{i\}} \xrightarrow{a_i}_{\mathfrak{F}} \mathsf{fail}_1 & & \text{for each } i \in [1,2n] \\
\mathsf{fail}_1 \xrightarrow{a_i}_{\mathfrak{F}} \mathsf{succ} & & \text{for each } i \in [1,2n] \\
\mathsf{succ} \xrightarrow{a_i}_{\mathfrak{F}} \mathsf{succ} & & \text{for each } i \in [1,2n] \\
\mathsf{fail}_1 \xrightarrow{\tau}_{\mathfrak{F}} \mathsf{fail}_2 & & \\
z_\perp \xrightarrow{a_i}_{\mathfrak{F}} z_\perp & & \text{for each } i \in [1,2n] \\
z_\perp \xrightarrow{\varphi}_{\mathfrak{F}} \mathsf{succ} & & \\
z_\perp \xrightarrow{a_i}_{\mathfrak{F}} \mathsf{reject} & & \text{for each } i \in [1,2n]
\end{array}
$$

Next, let us define the outgoing transitions of s_I and of $\overline{s_I}$ for each game-conform $I \subseteq [1,2n]$ and each possible $k \in [0,2n]$:

$s_{[1,k]}$	$\xrightarrow{a_{k+1}}_{\mathfrak{F}}$	$s_{[1,k+1]}$	
$\overline{s_{[1,k]}}$	$\xrightarrow{a_{k+1}}_{\mathfrak{F}}$	$\overline{s_{[1,k+1]}}$	
$s_{[1,k]}$	$\xrightarrow{a_{k+1}}_{\mathfrak{F}}$	$\overline{s_{[1,k+1]}}$	if k is odd
$\overline{s_{[1,k]}}$	$\xrightarrow{a_{k+1}}_{\mathfrak{F}}$	$s_{[1,k+1]}$	if k is even
$s_{[1,k]}, \overline{s_{[1,k]}}$	$\xrightarrow{a_i}_{\mathfrak{F}}$	reject	for each $i \in [1,2n]$
$s_{[1,k]}, \overline{s_{[1,k]}}$	$\xrightarrow{a_i}_{\mathfrak{F}}$	$z_\perp$	for each $i \in [1,k]$
$s_{[1,k]}, \overline{s_{[1,k]}}$	$\xrightarrow{a_i}_{\mathfrak{F}}$	$u_{[1,k]\cup\{i\}}$	for each $i \in [k+2,2n]$
$s_{[1,k]}, \overline{s_{[1,k]}}$	$\xrightarrow{a_l}_{\mathfrak{F}}$	$z_{[1,k],J,l}$	for each $[1,k] \subseteq J \subset [1,2n]$ and each $l \in [1,2] \setminus J$
$s_{[1,k]}, \overline{s_{[1,k]}}$	$\xrightarrow{\varphi}_{\mathfrak{F}}$	$v_{[1,k]}$	if $k \in [0,2n-1]$
$s_{[1,2n]}$	$\xrightarrow{\varphi}_{\mathfrak{F}}$	succ	
$\overline{s_{[1,2n]}}$	$\xrightarrow{\varphi}_{\mathfrak{F}}$	fail_1	

Finally, we conclude the definition of $\mathfrak{F}$ by defining outgoing transitions of each u_I:

u_I	$\xrightarrow{a_i}_{\mathfrak{F}}$	$u_{I\cup\{i\}}$	if $i \notin I$ and both I and $I \cup \{i\}$ are are non-game-conform
$u_{[1,k+1]\setminus\{i\}}$	$\xrightarrow{a_i}_{\mathfrak{F}}$	$s_{[1,k+1]}, \overline{s_{[1,k+1]}}$	for each $i \in [1,k]$
u_I	$\xrightarrow{a_i}_{\mathfrak{F}}$	reject	for each $i \in [1,2n]$
u_I	$\xrightarrow{a_l}_{\mathfrak{F}}$	$z_{I,J,l}$	for each $I \subseteq J$ and $l \in [1,2n] \setminus J$
u_I	$\xrightarrow{\varphi}_{\mathfrak{F}}$	v_I	
u_I	$\xrightarrow{a_i}_{\mathfrak{F}}$	$z_\perp$	for each $i \in I$

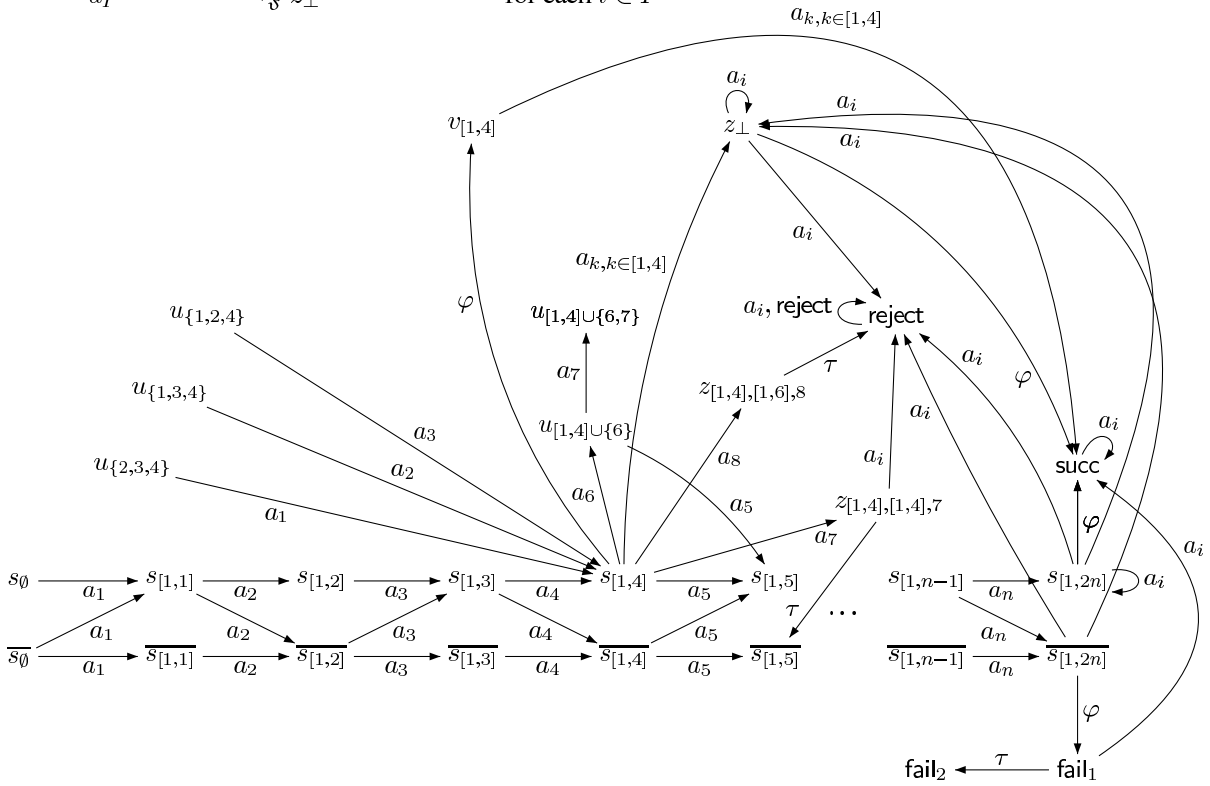


Fig. 3. A snapshot of $\mathfrak{F}$ and the outgoing transitions of $s_{[1,4]}$ (i ranges over $[1,2n]$).

Lemma 29. *Let $k \in [1, 2n]$. Then the following configurations if $\mathfrak{F}$ are not weakly bisimilar:*

1. $s_{[1,k]} \not\approx \overline{s_{[1,k]}}$,
2. *if k is even and $z_{[1,k-1],J,k} \xrightarrow{\tau}_{\mathfrak{F}}^{*} c$, then $s_{[1,k]} \not\approx c$.*
3. *if k is odd and $z_{[1,k-1],J,k} \xrightarrow{\tau}_{\mathfrak{F}}^{*} c$, then $\overline{s_{[1,k]}} \not\approx c$.*

Proof. We prove the lemma by downward induction on $k = 2n, 2n-1, \ldots$.

Induction base: For the induction base we have $k = 2n$.

1. Attacker plays $s_{[1,2n]} \xrightarrow{\varphi}_{\mathfrak{F}} \mathsf{succ}$. If Defender responds with $\overline{s_{[1,2n]}} \xrightarrow{\varphi\tau}_{\mathfrak{F}} \mathsf{fail}_2$, then Attacker obviously wins since fail_2 is a dead-end and succ is not. If Defender responds with $\overline{s_{[1,2n]}} \xrightarrow{\varphi}_{\mathfrak{F}} \mathsf{fail}_1$, then from the resulting game position $(\mathsf{succ}, \mathsf{fail}_1)$ Attacker plays in the next round $\mathsf{fail}_1 \xrightarrow{\tau}_{\mathfrak{F}} \mathsf{fail}_2$. Defender can only respond by staying in succ. As above, the resulting position $(\mathsf{succ}, \mathsf{fail}_2)$ is winning for Attacker.
2. Since $k = 2n$ and $k \notin J$ only $I = J = [1, 2n-1]$ is possible. Assume $z_{[1,2n-1],[1,2n-1],2n} \xrightarrow{\tau}_{\mathfrak{F}}^{*} c$. In case $c = \overline{s_{[1,2n]}}$ Attacker wins by Point 1 of induction base. In case $c = z_{[1,2n-1],[1,2n-1],2n}$ Attacker plays $z_{[1,2n-1],[1,2n-1],2n} \xrightarrow{\tau}_{\mathfrak{F}} \overline{s_{[1,2n]}}$ and Defender can only stay in $s_{[1,k]}$. Again, Attacker wins by Point 1 of induction base.
3. Nothing to be proven since $k = 2n$ is even.

Induction step: Let $0 \leq k < 2n$.

1. First, let us consider the case when k is odd. We will prove that Attacker has a winning strategy when he makes the move $s_{[1,k]} \xrightarrow{a_{k+1}}_{\mathfrak{F}} s_{[1,k+1]}$. We distinguish the possible responses of Defender.
   - Assume Defender responds $\overline{s_{[1,k]}} \xrightarrow{a_{k+1}}_{\mathfrak{F}} \mathsf{reject}$. Then in the next move Attacker plays $\mathsf{reject} \xrightarrow{\mathsf{reject}}_{\mathfrak{F}} \mathsf{reject}$ and wins since $s_{[1,k+1]}$ has no outgoing reject-transition.
   - Assume Defender responds $\overline{s_{[1,k]}} \xrightarrow{a_{k+1}}_{\mathfrak{F}} \overline{s_{[1,k+1]}}$. Then Attacker wins by Point 1 of induction hypothesis.
   - Assume Defender responds $\overline{s_{[1,k]}} \xrightarrow{a_{k+1}}_{\mathfrak{F}} z_{[1,k],J,k+1} \xrightarrow{\tau}_{\mathfrak{F}}^{*} c$. Then $k+1$ is even and Attacker wins by Point 2 of induction hypothesis.

   Now, let us assume k is even. We prove that Attacker has a winning strategy when he plays $\overline{s_{[1,k]}} \xrightarrow{a_{k+1}}_{\mathfrak{F}} \overline{s_{[1,k+1]}}$. We distinguish the possible responses of Defender.
   - Assume Defender responds $s_{[1,k]} \xrightarrow{a_{k+1}}_{\mathfrak{F}} \mathsf{reject}$. Then in the next move Attacker plays $\mathsf{reject} \xrightarrow{\mathsf{reject}}_{\mathfrak{F}} \mathsf{reject}$ and wins since $\overline{s_{[1,k+1]}}$ has no outgoing reject-transition.
   - Assume Defender responds $s_{[1,k]} \xrightarrow{a_{k+1}}_{\mathfrak{F}} s_{[1,k+1]}$.Then Attacker wins by Point 1 of induction hypothesis.
   - Assume Defender responds $s_{[1,k]} \xrightarrow{a_{k+1}}_{\mathfrak{F}} z_{[1,k],J,k+1} \xrightarrow{\tau}_{\mathfrak{F}}^{*} c$. Then $k+1$ is odd and Attacker wins by Point 3 of induction hypothesis.
2. Assume k is even. On the one hand, if $[1, k-1] = J$ and $z_{[1,k-1],J,k} \xrightarrow{\tau}_{\mathfrak{F}}^{*} c$, we distinguish the following possibilities for c:
   - $c = \overline{s_{[1,k]}}$: Then Attacker wins by Point 1 (no induction hypothesis used).
   - $c = z_{[1,k-1],[1,k-1],k}$: Then Attacker plays $z_{[1,k-1],[1,k-1],k} \xrightarrow{\tau}_{\mathfrak{F}} \overline{s_{[1,k]}}$ and Defender can only stay in $s_{[1,k]}$, hence Attacker wins by Point 1 (no induction hypothesis used).

   On the other hand, if $[1, k-1] \subset J$ and $z_{[1,k-1],J,k} \xrightarrow{\tau}_{\mathfrak{F}}^{*} c$, we distinguish the following possibilites for c:
   - $c = \mathsf{reject}$: Then Attacker plays $\mathsf{reject} \xrightarrow{\mathsf{reject}}_{\mathfrak{F}} \mathsf{reject}$ and wins since $s_{[1,k]}$ has no outgoing reject-transition.
   - $c = z_{[1,k-1],J,k}$: Then Attacker plays $z_{[1,k-1],J,k} \xrightarrow{\tau}_{\mathfrak{F}} \mathsf{reject}$ and Defender can only stay in $s_{[1,k]}$. In the next round, Attacker plays $\mathsf{reject} \xrightarrow{\mathsf{reject}}_{\mathfrak{F}} \mathsf{reject}$ and wins since $s_{[1,k]}$ has no outgoing reject-transition.
3. Assume k is odd. On the one hand, if $[1, k-1] = J$ and $z_{[1,k-1],J,k} \xrightarrow{\tau}_{\mathfrak{F}}^{*} c$, we distinguish the following possibilities for c:

- $c = s_{[1,k]}$: Then Attacker wins by Point 1 (no induction hypothesis used).
- $c = z_{[1,k-1],[1,k-1],k}$: Then Attacker plays $z_{[1,k-1],[1,k-1],k} \xrightarrow{\tau}_{\mathfrak{F}} s_{[1,k]}$ and Defender can only stay in $\overline{s_{[1,k]}}$, hence Attacker wins by Point 1 (no induction hypothesis used).

On the other hand, if $[1, k-1] \subset J$ and $z_{[1,k-1],J,k} \xrightarrow{\tau}{}^{*}_{\mathfrak{F}} c$, we distinguish the following possibilites for c:

- $c = \mathsf{reject}$: Then Attacker plays $\mathsf{reject} \xrightarrow{\mathsf{reject}}_{\mathfrak{F}} \mathsf{reject}$ and wins since $\overline{s_{[1,k]}}$ has no outgoing reject-transition.
- $c = z_{[1,k-1],J,k}$: Then Attacker plays $z_{[1,k-1],J,k} \xrightarrow{\tau}_{\mathfrak{F}} \mathsf{reject}$ and Defender can only stay in $\overline{s_{[1,k]}}$. In the next round, Attacker plays $\mathsf{reject} \xrightarrow{\mathsf{reject}}_{\mathfrak{F}} \mathsf{reject}$ and wins since $\overline{s_{[1,k]}}$ has no outgoing reject-transition.

□

5.2 The GTRS and its bottom-up computations

Recall that C denotes the set of configurations of $\mathfrak{F}$ and P denotes the set of proper leaf labels (Definition 5). Let us define the ranked alphabet $A = (A_i)_{i\in\{0,1,2\}}$ of our GTRS $\mathcal{R}$ as follows, where the set $\mathfrak{Q}$ will be specified later:

$$\begin{aligned} A_0 &= \{\mathsf{start}\} \cup C \cup P \cup \mathfrak{Q} \\ A_1 &= \{\mathsf{root}\} \\ A_2 &= \{\bigstar, \clubsuit\} \end{aligned}$$

For reasons of presentation, we will define the rewriting rules of $\mathcal{R}$ step by step. First we allow the following rewriting rules that allow to label leafs in analogy to Section 4.

$$\begin{array}{lll} (I,\alpha) \overset{a_i}{\hookrightarrow} (I \cup \{i\}, \alpha) & & \text{for each } i \in [1,2n] \setminus I \text{ and for each } \alpha \in \{0,1\} \\ (I,\alpha) \overset{a_i}{\hookrightarrow} (\bot, \alpha) & & \text{for each } i \in I \text{ and for each } \alpha \in \{0,1\} \\ (\bot,\alpha) \overset{a_i}{\hookrightarrow} (\bot, \alpha) & & \text{for each } i \in I \text{ and for each } \alpha \in \{0,1\} \end{array}$$

Moreover, we allow each transition of the finite system $\mathfrak{F}$, i.e.

$$c \overset{\lambda}{\hookrightarrow} c' \qquad \text{for each } c \xrightarrow{\lambda}_{\mathfrak{F}} c' \text{ where } \lambda \in \mathbb{A}_\tau.$$

Before we explicitly define them, let us give some explanation of the rewriting rules (J,l) for each $J \subset [1,2n]$ and each $l \in [1,2n] \setminus J$. These rewriting rules will be applied to combs.

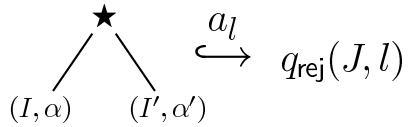
1. The initial rewriting rule $\texttt{INIT}(J,l)$ is done via the action a_l at the rightmost part of the comb.
2. If the tree automaton scanned some tree from $\mathsf{Combs}_\bot$ or from Combs_I where $I \not\subseteq J$, then the copy reject of $\mathfrak{F}$ can be reached via τ-transitions.
3. If the tree automaton scanned some tree from Combs_I, where $I \subseteq J$, then $z_{I,J,l}$ can be reached via τ-transitions.

Let us define the rewriting rules precisely. For each $J \subset [1,2n]$ and each $l \in [1,2n] \setminus J$ we add the following rewrite rules to $\mathcal{R}$ by first adding to $\mathfrak{Q}$ (and hence to A_0) the following additional nullary symbols

$$\mathfrak{Q}(J,l) \quad = \quad \{q_{\mathsf{rej}}(J,l)\} \cup \{q_I(J,l) \mid I \subseteq J\}.$$

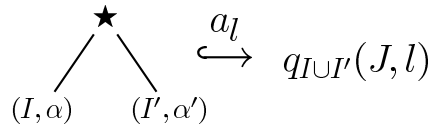
The state $q_{\mathsf{rej}}(J,l)$ is intended to express that there is no more way (by labeling leaves of the tree) of accepting any tree from Combs_I, where $I \subseteq J$, whereas states $q_I(J,l)$ express that so far we have collected the subset $I \subseteq J$.
If $I = \bot$ or $I' = \bot$ or $I \cap I' \neq \emptyset$ or $(I \cup I') \not\subseteq J$ we add the following rule

$\texttt{INIT}(J,l)$

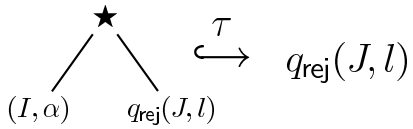


and otherwise we add

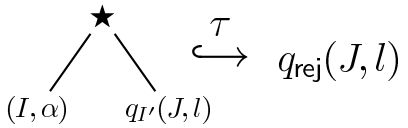
$\texttt{INIT}(J,l)$



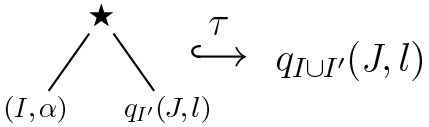
Let us give some intuition on the previously defined rules. The idea of applying these rules is as follows. Let us fix some comb T. The only rule that is applicable at T is the first rule (one of the two that is labeled with $\texttt{INIT}(J,l)$). Having done this (necessarily with action label a_l), we have replaced the three rightmost nodes of T by some state of a deterministic bottom-up tree automaton that intends to accept some tree $T \in \mathsf{Combs}_I$ for some $I \subseteq J$ and then goes to the configuration $z_{I,J,l}$ of $\mathfrak{F}$ (to reject otherwise). The following rules mimic this bottom-up computation and should be self-explanatory.



If $I = \bot$ or $I \cap I' \neq \emptyset$ or $I \not\subseteq J$ we add

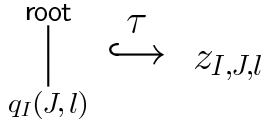


and otherwise

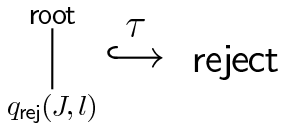


If the automaton is in state $q_I(J,l)$, then the automaton has collected from the bottom of the tree to the current node, the set $I \subseteq J$ of indices. If it scans a leaf that is labeled by $(\bot,\alpha)$ or with (I,α) where $I \not\subseteq J$ and where $\alpha \in \{0,1\}$ we move to the rejecting state $q_{\mathsf{rej}}(J,l)$ since then T surely is not in Combs_I for any $I \subseteq J$. The transitions of the tree automaton (except for the initial a_l-transition) are realized by τ-transitions in $\mathcal{R}$. The following rules are transitions to states of the finite system.

For each $I \subseteq J$ we add



and finally we add the rule



Note that once a_l has been done, all of the bottom-up rules are τ-rules. However, due to the possibility of rewriting leafs, it is possible that leafs are modified by executing a_i transitions intermediately.

Definition 30. *The* (J,l)-successor $T_{(J,l)}$ *of a comb* T *is the unique tree that is obtained by applying rewrite rule* $\texttt{INIT}(J,l)$ *at (the rightmost part of)* T.

The following lemmas relate all possible $T_{(J,l)}$ with the configurations of our finite transition system $\mathfrak{F}$.

Lemma 31. *Let* $T \in \mathsf{Combs}_I$ *for some* $I \subseteq [1,2n]$. *If* $I \not\subseteq J$ *and* $T_{(J,l)} \xrightarrow{\tau}{}^{*} T'$, *then* $T' \approx \mathsf{reject}$.

Proof. Let $U = \{T'' \in \mathsf{Trees}_A \setminus C \mid T_{(J,l)} \longrightarrow^{*} T''\}$ be the trees (and not any of the singleton trees that are copies of the configurations C of the finite system $\mathfrak{F}$) that are reachable from $T_{(J,l)}$. Then notice easily that

$$(U \cup \{\mathsf{reject}\}) \times \{\mathsf{reject}\}$$

is a weak bisimulation that relates $T' \in U$ and reject. □

Lemma 32. *Let* $T \in \mathsf{Combs}_I$ *for some* $I \subset [1,2n]$. *If* $I \subseteq J$ *and* $T_{(J,l)} \xrightarrow{\tau}{}^{*} T'$, *then* $T' \approx z_{I,J,l}$.

Proof. Let $U = \{T'' \in \mathsf{Trees}_A \setminus C \mid T_{(J,l)} \longrightarrow^{*} T''\}$ be the set of trees (which are not configurations of the finite system) that are reachable from $T_{(J,l)}$ in $\mathfrak{S}(\mathcal{R})$. One can now partition U into the following subsets:

$$\begin{aligned} U_\bot &= \{T'' \in R \mid T'' \xrightarrow{\tau}{}^{*} \mathsf{reject}\} \\ U_{I'} &= \{T'' \in R \mid T'' \xrightarrow{\tau}{}^{*} z_{I',J,l}\} \qquad \text{for each } I \subseteq I' \subseteq J \end{aligned}$$

One now verifies that

$$U_\bot \times \{\mathsf{reject}\} \quad \cup \quad \bigcup_{I \subseteq I' \subseteq J} U_{I'} \times \{z_{I',J,l}\} \quad \cup \quad \{(c,c) \mid c \in C\}$$

is a weak bisimulation that relates $T' \in U_I$ and $z_{I,J,l}$. □

Recall that Combs_φ denotes the set of all combs $T \in \mathsf{Combs}_{[1,2n]}$ such that $T \models \psi[\nu_T]$. In analogy to each of the rewrite rules (J,l) we will add a rule set with additional nullary symbols $\mathfrak{Q}(\varphi)$ to $\mathfrak{Q}$ (and hence to A_0) that mimics a deterministic bottom-up tree automaton that realizes the tree language Combs_φ, runs on combs and that does the following (details are given in the appendix):

1. The initial rewrite rule $\texttt{INIT}(\varphi)$ is labeled with the action φ (instead of a_l)
2. If the tree automaton scanned some tree from Combs_φ, then fail_1 can be reached via τ-transitions.
3. If the tree automaton scanned some tree from $\mathsf{Combs}_{\overline{\varphi}} \cup \mathsf{Combs}_\bot$, then succ can be reached via τ-transitions.
4. If the tree automaton scanned a tree from Combs_I for some $I \subset [1,2n]$, then v_I can be reached via τ-transitions.

We can now define the remaining nullary symbols $\mathfrak{Q}$ that A_0 contains. We define

$$\mathfrak{Q} \quad = \quad \mathfrak{Q}(\varphi) \cup \bigcup \{\mathfrak{Q}(J,l) \mid J \subset [1,2n], l \in [1,2n] \setminus J\}\,.$$

In analogy to a (J,l) successor $T_{(J,l)}$ of a comb T, we define the φ-*successor* T_φ of a comb T.

Definition 33. *The* φ-successor T_φ *of a comb* T *is the unique tree that is obtained by applying the rewrite rule* $\texttt{INIT}(\varphi)$ *to* T.

Next, we relate all possible T_φ with configurations of our finite transition system $\mathfrak{F}$.

Lemma 34. *For every* $T \in \mathsf{Combs}_\bot$ *the following three statements hold:*

1. $T_{(J,l)} \approx \mathsf{reject}$ *for each* J, l.
2. $T_\varphi \approx \mathsf{succ}$.
3. $T \approx z_\bot$.

Proof. Points 1 and 2 are immediate since the bottom-up computations are rejecting and are designed to go to reject or to succ, respectively. For Point 3, Defender has the following winning strategy: As long as the current tree T stays in $\mathsf{Combs}_\perp$, Defender loops in $z_\perp$ and vice versa. In case Attacker moves $T \xrightarrow{a_l} T_{(J,l)}$, then Defender responds $z_\perp \xrightarrow{a_l}_{\mathfrak{F}} \mathsf{reject}$ and wins by Point 1 and vice versa. In case Attacker moves $T \xrightarrow{\varphi} T_\varphi$, then Defender repsonds $z_\perp \xrightarrow{\varphi}_{\mathfrak{F}} \mathsf{succ}$ and vice versa and hence wins by Point 2. □

Lemma 35. *Let $T \in \mathsf{Combs}_{[1,2n]}$. Then the following two statements hold:*

1. *$T \in \mathsf{Combs}_\varphi$ if and only if $T_\varphi \approx \mathsf{fail}_1$.*
2. *$T \in \mathsf{Combs}_{\overline{\varphi}}$ if and only if $T_\varphi \approx \mathsf{succ}$.*

Proof. Let $U = \{T' \in \mathsf{Trees}_A \setminus C \mid T_\varphi \longrightarrow^* T'\}$ denote the set of all trees $T' \in \mathsf{Trees}_A \setminus C$ that are reachable from T_φ.

1. On the one hand assume $T \in \mathsf{Combs}_\varphi$. We partition U into the following two sets:

$$U_\varphi = \left\{T' \in U : T' \xrightarrow{\tau}{}^* \mathsf{fail}_1\right\} \qquad \text{and} \qquad \overline{U_\varphi} = U \setminus U_\varphi.$$

   One can now verify that

$$((U_\varphi \cup \{\mathsf{fail}_1\}) \times \{\mathsf{fail}_1\}) \quad \cup \quad ((\overline{U_\varphi} \cup \{\mathsf{succ}\}) \times \{\mathsf{succ}\}) \quad \cup \quad (\{\mathsf{fail}_2\} \times \{\mathsf{fail}_2\})$$

   is a weak bisimulation that relates fail_1 and $T_\varphi \in U_\varphi$.
   On the other hand, assume $T \in \mathsf{Combs}_{\overline{\varphi}}$. This implies $T_\varphi \not\longrightarrow^* \mathsf{fail}_1$. Attacker can play $\mathsf{fail}_1 \xrightarrow{\tau}_{\mathfrak{F}} \mathsf{fail}_2$. For every response $T_\varphi \xrightarrow{\tau}{}^* T''$ of Defender we have that T'' has an a_i-successor (note that $T'' = \mathsf{succ}$ is possible), whereas fail_2 is a dead-end. Hence Attacker has a winning strategy.
2. On the one hand, assume $T \in \mathsf{Combs}_{\overline{\varphi}}$. One verifies that

$$(U \cup \{\mathsf{succ}\}) \times \{\mathsf{succ}\}$$

   is a weak bisimulation that relates succ and $T_\varphi \in U$.
   On the other hand assume $T \in \mathsf{Combs}_\varphi$. Attacker can play a finite sequence $T_\varphi \xrightarrow{\tau}{}^* \mathsf{fail}_1 \xrightarrow{\tau} \mathsf{fail}_2$ of τ-moves ending in a dead-end and Defender can only stay in succ. Hence Attacker wins.

□

Lemma 36. *Let $T \in \mathsf{Combs}_I$ for some $I \subset J$. Then $T_\varphi \approx v_I$.*

Proof. Let $U = \{T' \in \mathsf{Trees}_A \setminus C \mid T \longrightarrow^* T'\}$ denote the set of trees that are reachable from T not being copies of configurations from $\mathfrak{F}$. Observe that by definition of the rewriting rules for simulating Combs_φ, we partition U into the following sets

$$\begin{aligned} U_\perp &= \{T' \in U \mid T \xrightarrow{\tau}{}^* \mathsf{succ}\} \\ U_\varphi &= \{T' \in U \mid T \xrightarrow{\tau}{}^* \mathsf{fail}_1\} \\ U_{I'} &= \{T' \in U \mid T \xrightarrow{\tau}{}^* v_{I'}\} \qquad \text{for each } I \subseteq I' \subset [1,2n] \end{aligned}$$

One can now easily verify that

$$(U_\perp \cup \{\mathsf{succ}\}) \times \{\mathsf{succ}\} \quad \cup \quad (U_\varphi \cup \{\mathsf{fail}_1\}) \times \{\mathsf{fail}_1\} \quad \cup \quad \{\mathsf{fail}_2\} \times \{\mathsf{fail}_2\}$$

$$\cup \bigcup_{I \subseteq I' \subset [1,2n]} (U_{I'} \cup \{v_{I'}\}) \times \{v_{I'}\}$$

is a weak bisimulation that relates $T_\varphi \in U_I$ and v_I. □

5.3 Correctness of the reduction

The next lemma is the main correctness lemma of our reduction.

Lemma 37. *Let $I \subseteq [1,2n]$ be non-empty.*

1. *If I is game-conform, then the following two statements hold:*
   (a) $\forall T \in \mathsf{Combs}_I : T \not\approx s_I$ if and only if $T \models \psi[\nu_T]$.
   (b) $\forall T \in \mathsf{Combs}_I : T \approx \overline{s_I}$ if and only if $T \models \psi[\nu_T]$.
2. *If I is non-game-conform, then for each $T \in \mathsf{Combs}_I$ we have $T \approx u_I$.*

Proof. We prove the lemma by downward induction on $|I| = 2n, 2n-1, \ldots$.

Induction base: Assume $|I| = 2n$, i.e. $I = [1,2n]$ and so I is game-conform. Thus, we only have to prove Point 1.

(a) Let $T \in \mathsf{Combs}_{[1,2n]}$. On the one hand, assume $T \models \psi[\nu_T]$ or equivalently $T \in \mathsf{Combs}_\varphi$. Attacker can move $T \xrightarrow{\varphi} T_\varphi$ and Defender can only respond with $s_{[1,2n]} \xrightarrow{\varphi}_{\mathfrak{F}} \mathsf{succ}$ establishing the pair $(T_\varphi, \mathsf{succ})$ which is winning for Attacker by Point 2 of Lemma 35.

On the other hand, assume $T \not\models \varphi[\nu_T]$ or equivalently $T \in \mathsf{Combs}_{\overline{\varphi}}$. We have to prove $T \approx s_{[1,2n]}$.

1. If Attacker plays $T \xrightarrow{\varphi} T_\varphi$, then Defender responds $s_{[1,2n]} \xrightarrow{\varphi}_{\mathfrak{F}} \mathsf{succ}$ and wins by Point 2 of Lemma 35 and vice versa.
2. If Attacker plays $T \xrightarrow{a_l} T_{(J,l)}$ (resp. $s_{[1,2n]} \xrightarrow{a_l} \mathsf{reject}$), then Defender responds $s_{[1,2n]} \xrightarrow{a_l} \mathsf{reject}$ (resp. $T \xrightarrow{a_l} T_{(J,l)}$ for some arbitrary possible J, say $J = \emptyset$) and wins since $T_{(J,l)} \approx \mathsf{reject}$ by Lemma 31 (Lemma 31 is applicable since $T \in \mathsf{Combs}_{[1,2n]}$ and $[1,2n] \not\subseteq J$ due to $J \subset [1,2n]$).
3. If Attacker plays $T \xrightarrow{a_i} T'$ for some $T' \in \mathsf{Combs}_\perp$ (resp. Attacker plays $s_{[1,2n]} \xrightarrow{a_i}_{\mathfrak{F}} z_\perp$), then Defender responds with $s_{[1,2n]} \xrightarrow{a_i}_{\mathfrak{F}} z_\perp$ (resp. with $T \xrightarrow{a_i} T'$ for some $T' \in \mathsf{Combs}_\perp$) and Defender wins by Point 3 of Lemma 34.

(b) On the one hand, assume $T \not\models \varphi[\nu_T]$. Then Attacker plays $T \xrightarrow{\varphi} T_\varphi$. If Defender responds $\overline{s_{[1,2n]}} \xrightarrow{\varphi}_{\mathfrak{F}} \mathsf{fail}_1$, then Attacker wins by Point 1 of Lemma 35. If, however, Defender responds $\overline{s_{[1,2n]}} \xrightarrow{\varphi}_{\mathfrak{F}} \mathsf{fail}_1 \xrightarrow{\tau}_{\mathfrak{F}} \mathsf{fail}_2$, then from the established pair $(T_\varphi, \mathsf{fail}_2)$ Attacker can win by forcing the game via τ-moves to $(\mathsf{succ}, \mathsf{fail}_2)$ which is winning for him.

On the other hand, assume $T \models \varphi[\nu_T]$ or equivalently $T \in \mathsf{Combs}_\varphi$. We have to show $T \approx \overline{s_{[1,2n]}}$.

1. If Attacker plays $T \xrightarrow{\varphi} T_\varphi$, then Defender responds $\overline{s_{[1,2n]}} \xrightarrow{\varphi}_{\mathfrak{F}} \mathsf{fail}_1$ and wins by Point 1 of Lemma 35 and vice versa.
2. If Attacker plays $T \xrightarrow{a_l} T_{(J,l)}$ (resp. $\overline{s_{[1,2n]}} \xrightarrow{a_l} \mathsf{reject}$), then Defender responds $\overline{s_{[1,2n]}} \xrightarrow{a_l} \mathsf{reject}$ (resp. $T \xrightarrow{a_l} T_{(J,l)}$ for some arbitrary possible J, say $J = \emptyset$) and wins since $T_{(J,l)} \approx \mathsf{reject}$ by Lemma 31 (Lemma 31 is applicable since $T \in \mathsf{Combs}_{[1,2n]}$ and $[1,2n] \not\subseteq J$ due to $J \subset [1,2n]$).
3. If Attacker plays $T \xrightarrow{a_i} T'$ for some $T' \in \mathsf{Combs}_\perp$ (resp. Attacker plays $s_{[1,2n]} \xrightarrow{a_i}_{\mathfrak{F}} z_\perp$), then Defender responds with $s_{[1,2n]} \xrightarrow{a_i}_{\mathfrak{F}} z_\perp$ (resp. with $T \xrightarrow{a_i} T'$ for some $T' \in \mathsf{Combs}_\perp$) and Defender wins by Point 3 of Lemma 34.

Induction step: Let $I \subset [1,2n]$ be non-empty.

1. Assume that I is game-conform, i.e. $I = [1,k]$ for some $k \in [2n-1]$. We make a case distinction on the parity of k:
   (a) This case can be proven analogously to case (b) below.
   (b) We make a case distinction on the parity of k.
   - k is even: Recall that

$$\psi[\nu_T] \quad = \quad \exists x_{k+1} \forall x_{k+2} \cdots \exists x_{2n-1} \forall x_{2n} \varphi[\nu_T].$$

On the one hand, assume $T \models \psi[\nu_T]$. This means that there is some $k+1$-extension $T' \in \mathsf{Combs}_{[1,k+1]}$ of T such that $T' \models \psi[\nu_{T'}]$. We have to show that $T \approx \overline{s_{[1,k]}}$.

Attacker plays in the finite system

- If Attacker plays $\overline{s_{[1,k]}} \xrightarrow{a_{k+1}}_{\mathfrak{F}} \overline{s_{[1,k+1]}}$, then Defender responds with $T \xrightarrow{a_{k+1}} T'$ and Defender wins by Point 1(b) of induction hypothesis.
- Assume Attacker plays $\overline{s_{[1,k]}} \xrightarrow{a_{k+1}}_{\mathfrak{F}} s_{[1,k+1]}$. Recall that $T_{([1,k],k+1)}$ is the a_{k+1}-successor of T that initializes a bottom-up computation for accepting $\mathsf{Combs}_{[1,k]}$. Since we have $T \in \mathsf{Combs}_{[1,k]}$ Defender can do the bottom-up computation via τ-rules and can respond with $T \xrightarrow{a_{k+1}} T_{([1,k],k+1)} \xrightarrow{\tau}{}^{*} z_{[1,k],[1,k],k+1} \xrightarrow{\tau}{}^{*} s_{[1,k+1]}$ and wins by establishing syntactic equivalence.
- If Attacker moves $\overline{s_{[1,k]}} \xrightarrow{a_i}_{\mathfrak{F}} \mathsf{reject}$ for some $i \in [1,2n]$, then Defender responds with $T \xrightarrow{a_i} T_{(\emptyset,i)}$ and wins since $T_{(\emptyset,l)} \approx \mathsf{reject}$ by Lemma 31 (note that since $T \in \mathsf{Combs}_{[1,k]}$ and $[1,k]$ is non-empty thus $[1,k] \not\subseteq \emptyset$, Lemma 31 is applicable).
- If Attacker moves $\overline{s_{[1,k]}} \xrightarrow{a_i}_{\mathfrak{F}} u_{[1,k]\cup\{i\}}$ for some $i \in [k+2,2n]$, then $[1,k] \cup \{i\}$ is non-game-conform and thus Defender repsonds with $T \xrightarrow{a_i} T'$ for some i-extension $T' \in \mathsf{Combs}_{[1,k]\cup\{i\}}$ of T and wins by Point 2 of induction hypothesis.
- If Attacker moves $\overline{s_{[1,k]}} \xrightarrow{\varphi} v_{[1,k]}$, then Defender respnds with $T \xrightarrow{\varphi} T_\varphi$ and wins by Lemma 36.
- If Attacker plays $\overline{s_{[1,k]}} \xrightarrow{a_l} z_{[1,k],J,l}$, where $[1,k] \subseteq J \subset [1,2n]$ and $l \in [1,2n] \setminus J$, then Defender responds with $T \xrightarrow{a_l} T_{(J,l)}$ and wins by Lemma 32 (note that $T \in \mathsf{Combs}_{[1,k]}$).
- If Attacker plays $\overline{s_{[1,k]}} \xrightarrow{a_i}_{\mathfrak{F}} z_\bot$ for some $i \in [1,k]$, then Defender responds $T \xrightarrow{a_i} T'$ for some $T' \in \mathsf{Combs}_\bot$ and wins by Point 3 of Lemma 34.

Attacker plays in the infinite system

- If Attacker plays $T \xrightarrow{a_{k+1}} T''$ for some $k+1$-extension T'' of T, then we distinguish two cases. If $T'' \models \psi[\nu_{T''}]$, then Defender responds with $\overline{s_{[1,k]}} \xrightarrow{a_{k+1}}_{\mathfrak{F}} \overline{s_{[1,k+1]}}$ and wins by Point 1(b) of induction hypothesis. If $T'' \not\models \psi[\nu_{T''}]$, then Defender responds with $\overline{s_{[1,k]}} \xrightarrow{a_{k+1}}_{\mathfrak{F}} s_{[1,k+1]}$ and wins by Point 1(a) of induction hypothesis.
- If Attacker plays $T \xrightarrow{a_i} T''$, where $i \in [k+2,2n]$ and $T'' \in \mathsf{Combs}_{[1,k]\cup\{i\}}$, then $[1,k] \cup \{i\}$ is non-game-conform and Defender responds with $\overline{s_{[1,k]}} \xrightarrow{a_i}_{\mathfrak{F}} u_{[1,k]\cup\{i\}}$ and wins by Point 2 of induction hypothesis.
- If Attacker plays $T \xrightarrow{a_i} T''$ for some $i \in [1,k]$ and some $T'' \in \mathsf{Combs}_\bot$, then Defender responds with $\overline{s_{[1,k]}} \xrightarrow{a_i} z_\bot$ and wins by Point 3 of Lemma 34.
- Assume Attacker plays $T \xrightarrow{a_l} T_{(J,l)}$. Then we distinguish two cases. In case $[1,k] \subseteq J$, then Defender responds $\overline{s_{[1,k]}} \xrightarrow{a_l}_{\mathfrak{F}} z_{[1,k],J,l}$ and wins by Lemma 32 (note that $T \in \mathsf{Combs}_{[1,k]}$). In case $[1,k] \not\subseteq J$, then we have $T_{(J,l)} \approx \mathsf{reject}$ by Lemma 31. Hence Defender wins by responding with $\overline{s_{[1,k]}} \xrightarrow{a_l}_{\mathfrak{F}} \mathsf{reject}$.
- If Attacker plays $T \xrightarrow{\varphi}_{\mathfrak{F}} T_\varphi$, then Defender responds with $\overline{s_{[1,k]}} \xrightarrow{\varphi} v_{[1,k]}$ and wins by Lemma 36.

On the other hand assume $T \not\models \varphi[\nu_T]$. Thus for each $k+1$-extension T' of T we have $T' \not\models \psi[\nu_{T'}]$. We have to show that $T \not\approx \overline{s_{[1,k]}}$. We will prove that Attacker can win by playing $\overline{s_{[1,k]}} \xrightarrow{a_{k+1}}_{\mathfrak{F}} \overline{s_{[1,k+1]}}$ (recall that k is even):

- If Defender responds $T \xrightarrow{a_{k+1}} T'$, where $T' \in \mathsf{Combs}_{[1,k+1]}$ is some $k+1$-extension of T, we have that $T' \not\models \psi[\nu_{T'}]$. Hence Attacker wins by Point 1(b) of induction hypothesis.
- If Defender responds $T \xrightarrow{a_{k+1}} T_{(J,k+1)} \xrightarrow{\tau}{}^{*} T'$ we distinguish two cases. In case $[1,k] \not\subseteq J$, then $T' \approx \mathsf{reject}$ by Lemma 31. Attacker can now force the game from T' to reject via τ-moves and then move $\mathsf{reject} \xrightarrow{\mathsf{reject}}_{\mathfrak{F}} \mathsf{reject}$ whereas Defender has to stay in $\overline{s_{[1,k+1]}}$, which has no outgoing reject-transition. Hence Attacker wins. In case $[1,k] \subseteq J$, then we have $T' \approx z_{[1,k],J,k+1}$ by Lemma 32. But note that $k+1$ is odd and thus $z_{[1,k],J,k+1} \not\approx \overline{s_{[1,k+1]}}$ by Point 3 of Lemma 29.

- k is odd: Recall that

$$\psi[\nu_T] \quad = \quad \forall x_{k+1} \exists x_{k+2} \cdots \forall x_{2n} \varphi[\nu_T].$$

On the one hand, assume $T \models \psi[\nu_T]$. This means that for each $k+1$-extension $T' \in \mathsf{Combs}_{[1,k+1]}$ of T we have $T' \models \psi[\nu_{T'}]$. We have to show that $T \approx \overline{s_{[1,k]}}$.

Attacker plays in the finite system

- If Attacker plays $\overline{s_{[1,k]}} \xrightarrow{a_{k+1}}_{\mathfrak{F}} \overline{s_{[1,k+1]}}$, then Defender responds with $T \xrightarrow{a_{k+1}} T'$ for an arbitrary $k+1$-extension T' of T and wins by Point 1(b) of induction hypothesis.
- If Attacker moves $\overline{s_{[1,k]}} \xrightarrow{a_i}_{\mathfrak{F}} \mathsf{reject}$ for some $i \in [1,2n]$, then Defender responds with $T \xrightarrow{a_i} T_{(\emptyset,i)}$ and wins since $T_{(\emptyset,l)} \approx \mathsf{reject}$ by Lemma 31 (note that since $T \in \mathsf{Combs}_{[1,k]}$ and $[1,k]$ is non-empty thus $[1,k] \not\subseteq \emptyset$, Lemma 31 is applicable).
- If Attacker moves $\overline{s_{[1,k]}} \xrightarrow{a_i}_{\mathfrak{F}} u_{[1,k]\cup\{i\}}$ for some $i \in [k+2,2n]$, then $[1,k] \cup \{i\}$ is non-game-conform and thus Defender repsonds with $T \xrightarrow{a_i} T'$ for some i-extension $T' \in \mathsf{Combs}_{[1,k]\cup\{i\}}$ of T wins by Point 2 of induction hypothesis.
- If Attacker plays $\overline{s_{[1,k]}} \xrightarrow{\varphi}_{\mathfrak{F}} v_{[1,k]}$, then Defender responds with $T \xrightarrow{\varphi} T_\varphi$ and wins by Lemma 36 (recall that $T \in \mathsf{Combs}_{[1,k]}$).
- If Attacker plays $\overline{s_{[1,k]}} \xrightarrow{a_l} z_{[1,k],J,l}$, where $[1,k] \subseteq J \subset [1,2n]$ and $l \in [1,2n] \setminus J$, then Defender responds with $T \xrightarrow{a_l} T_{(J,l)}$ and wins by Lemma 32 (note that $T \in \mathsf{Combs}_{[1,k]}$).
- If Attacker plays $\overline{s_{[1,k]}} \xrightarrow{a_i}_{\mathfrak{F}} z_\bot$ for some $i \in [1,k]$, then Defender responds $T \xrightarrow{a_i} T'$ for some $T' \in \mathsf{Combs}_\bot$ and wins by Point 3 of Lemma 34.

Attacker plays in the infinite system

- If Attacker plays $T \xrightarrow{a_{k+1}} T'$ for some $k+1$-extension T' of T, then we know that $T' \models \psi[\nu_{T'}]$. Hence Defender can respond with $\overline{s_{[1,k]}} \xrightarrow{a_{k+1}}_{\mathfrak{F}} \overline{s_{[1,k+1]}}$ and wins by Point 1(b) of induction hypothesis.
- If Attacker plays $T \xrightarrow{a_i} T''$, where $i \in [k+2,2n]$ and $T'' \in \mathsf{Combs}_{[1,k]\cup\{i\}}$, then $[1,k] \cup \{i\}$ is non-game-conform and Defender responds with $\overline{s_{[1,k]}} \xrightarrow{a_i}_{\mathfrak{F}} u_{[1,k]\cup\{i\}}$ and wins by Point 2 of induction hypothesis.
- If Attacker plays $T \xrightarrow{a_i} T''$ for some $i \in [1,k]$ and some $T'' \in \mathsf{Combs}_\bot$, then Defender responds with $\overline{s_{[1,k]}} \xrightarrow{a_i} z_\bot$ and wins by Point 3 of Lemma 34.
- Assume Attacker plays $T \xrightarrow{a_l} T_{(J,l)}$. Then we distinguish two cases. In case $[1,k] \subseteq J$, then Defender responds $\overline{s_{[1,k]}} \xrightarrow{a_l}_{\mathfrak{F}} z_{[1,k],J,l}$ and wins by Lemma 32 (note that $T \in \mathsf{Combs}_{[1,k]}$). In case $[1,k] \not\subseteq J$, then we have $T_{(J,l)} \approx \mathsf{reject}$ by Lemma 31. Hence Defender wins by responding with $\overline{s_{[1,k]}} \xrightarrow{a_l}_{\mathfrak{F}} \mathsf{reject}$.
- If Attacker plays $T \xrightarrow{\varphi}_{\mathfrak{F}} T_\varphi$, then Defender responds with $\overline{s_{[1,k]}} \xrightarrow{\varphi} v_{[1,k]}$ and wins by Lemma 36.

On the other hand assume $T \not\models \psi[\nu_T]$. Thus there is a $k+1$-extension T' of T such that $T' \not\models \psi[\nu_{T'}]$. We have to show that $T \not\approx \overline{s_{[1,k]}}$. We will prove that Attacker can win by playing $T \xrightarrow{a_{k+1}} T'$ (recall that k is odd):

- If Defender responds with $\overline{s_{[1,k]}} \xrightarrow{a_{k+1}}_{\mathfrak{F}} \overline{s_{[1,k+1]}}$ then Attacker wins by Point 1(b) of induction hypothesis.
- If Defender responds $\overline{s_{[1,k]}} \xrightarrow{a_{k+1}}_{\mathfrak{F}} \mathsf{reject}$, then in the next round Attacker plays $\mathsf{reject} \xrightarrow{\mathsf{reject}}_{\mathfrak{F}} \mathsf{reject}$ and wins since Defender cannot respond from T with a reject-transition.
- Assume Defender responds with $\overline{s_{[1,k]}} \xrightarrow{a_{k+1}}_{\mathfrak{F}} z_{[1,k],J,k+1} \xrightarrow{\tau}{}^*_{\mathfrak{F}} c$. Note that since $T' \not\models \psi[\nu_{T'}]$ we have $T' \approx s_{[1,k+1]}$ by Point 1(a) of induction hypothesis. Furthermore $k+1$ is even, hence $s_{[1,k+1]} \not\approx c$ by Point 2 of Lemma 29. Thus $T' \not\approx c$.

2. Assume I is non-game-conform. Let $T \in \mathsf{Combs}_I$. We have to prove $T \approx u_I$.

Attacker plays in the finite system

- If Attacker moves $u_I \xrightarrow{a_i}_{\mathfrak{F}} u_{I\cup\{i\}}$ where $i \notin I$ and $I \cup \{i\}$ is non-game-conform, then Defender responds $T \xrightarrow{a_i}_{\mathfrak{F}} T'$ for some i-extension $T' \in \mathsf{Combs}_{I\cup\{i\}}$ and wins by Point 2 of induction hypothesis.
- If $I = [1,k] \setminus \{i\}$ for some $1 \leq i < k$ and Attacker moves $u_I \xrightarrow{a_i}_{\mathfrak{F}} s_{[1,k]}$ $\left(\text{resp. } u_I \xrightarrow{a_i}_{\mathfrak{F}} \overline{s_{[1,k]}}\right)$ then Defender establishes syntactic equivalence by responding $T \xrightarrow{a_i} T_{(I,I,i)} \xrightarrow{\tau}{}^{*} z_{I,I,i} \xrightarrow{\tau} s_{[1,k]}$ $\left(\text{resp. } T \xrightarrow{a_i} T_{(I,I,i)} \xrightarrow{\tau}{}^{*} z_{I,I,i} \xrightarrow{\tau} \overline{s_{[1,k]}}\right)$ and thus wins.
- If Attacker moves $u_I \xrightarrow{a_i}_{\mathfrak{F}} \mathsf{reject}$ for some $i \in [1,2n]$, then Defender responds with $T \xrightarrow{a_i} T_{(\emptyset,i)}$ and since $I \not\subseteq \emptyset$ we have $T_{(\emptyset,i)} \approx \mathsf{reject}$ by Lemma 31 and thus Defender wins.
- If Attacker moves $u_I \xrightarrow{a_l} z_{I,J,l}$, then Defender responds $T \xrightarrow{a_l} T_{(J,l)}$ and wins by Lemma 32.
- If Attacker moves $u_I \xrightarrow{a_i}_{\mathfrak{F}} z_\bot$ for some $i \in I$, then Defender responds with $T \xrightarrow{a_i} T'$ where $T' \in \mathsf{Combs}_\bot$ and wins by Point 3 of Lemma 34.
- If Attacker moves $u_I \xrightarrow{\varphi} v_I$, then Defender responds $T \xrightarrow{\varphi} T_\varphi$ and wins by Lemma 36.

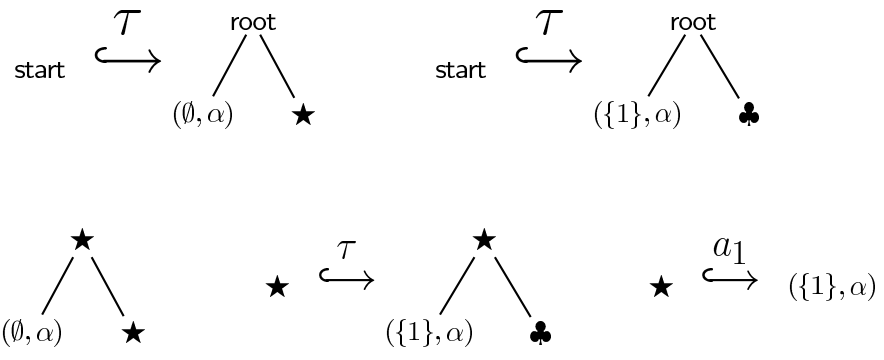
Attacker plays in the infinite system

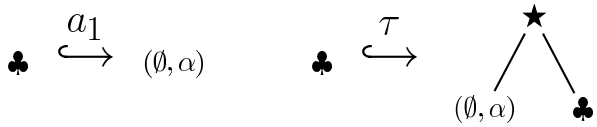
- If Attacker plays $T \xrightarrow{a_i} T'$ where $i \in [1,2n] \setminus I$, $I \cup \{i\}$ is non-game-conform, then Defender responds with $u_I \xrightarrow{a_i}_{\mathfrak{F}} u_{I\cup\{i\}}$ and wins by Point 2 of induction hypothesis.
- If $I = [1,k] \setminus \{i\}$ and Attacker plays $T \xrightarrow{a_i} T'$ for some $T' \in \mathsf{Combs}_{[1,k]}$, then we distinguish two cases. In case $T' \models \psi[\nu_{T'}]$, then Defender responds $u_{[1,k]\setminus\{i\}} \xrightarrow{a_i}_{\mathfrak{F}} \overline{s_{[1,k]}}$ and wins by Point 1(b) of induction hypothesis. In case $T' \not\models \psi[\nu_{T'}]$, then Defender responds $u_{[1,k]\setminus\{i\}} \xrightarrow{a_i}_{\mathfrak{F}} s_{[1,k]}$ and wins by Point 1(a) of induction hypothesis.
- If Attacker plays $T \xrightarrow{a_i} T'$ for some $i \in [1,k]$ and some $T' \in \mathsf{Combs}_\bot$, then Defender responds with $u_I \xrightarrow{a_i}_{\mathfrak{F}} z_\bot$ and wins by Point 3 of Lemma 34.
- Attacker plays $T \xrightarrow{a_l} T_{(J,l)}$. Then we distinguish two cases. In case $I \subseteq J$, then Defender responds $u_I \xrightarrow{a_l}_{\mathfrak{F}} z_{I,J,l}$ and wins by Lemma 32 (note that $T \in \mathsf{Combs}_I$). In case $I \not\subseteq J$, then we have $T_{(J,l)} \approx \mathsf{reject}$ by Lemma 31. Hence Defender wins by responding with $u_I \xrightarrow{a_l}_{\mathfrak{F}} \mathsf{reject}$.
- If Attacker plays $T \xrightarrow{\varphi} T_\varphi$, then Defender responds with $u_I \xrightarrow{\varphi} v_I$ and wins by Lemma 36.

□

5.4 Putting the pieces together

Finally, we add the the following rewriting rules that force Attacker to move from the singleton tree start to some comb $T \in \mathsf{Combs}_{[1,1]}$ such that $T \models \psi[\nu_T]$. This will formally be proven below. We add the following rules to R, where $\alpha \in \{0,1\}$:





Theorem 38. *Weak bisimilarity of GTRS against finite systems is nonelementary.*

Proof. We first prove the correctness of our reduction. We will prove

$$\exists T \in \mathsf{Combs}_{[1,1]} : T \models \psi[\nu_T] \qquad \text{if and only if} \qquad \mathsf{start} \not\approx s_\emptyset$$

On the one hand assume $T \models \psi[\nu_T]$ for some $T \in \mathsf{Combs}_{[1,1]}$. Attacker can now win as follows: He first plays only in $\mathfrak{S}(\mathcal{R})$ and forces to game from start to T via $\mathsf{start} \xrightarrow{\tau^* a_1} T$. Defender can only respond by looping in $s_\emptyset$ and then respond $s_\emptyset \xrightarrow{a_1} s_{[1,1]}$. From the resulting pair $(T, s_{[1,1]})$ Attacker has a winning strategy by Point 1(a) of Lemma 37.

On the other hand, assume that for all $T \in \mathsf{Combs}_{[1,1]}$ we have $T \not\models \psi[\nu_T]$. Defender can now win as follows: As long only τ-transitions have been played she stays in $s_\emptyset$. Attacker cannot do this forever, otherwise he loses. Hence, as soon as Attacker plays the first a_1-transition, Defender responds as follows, depending on the situation: If Attacker's a_1-move is $T \xrightarrow{a_1} T'$, where $T' \in \mathsf{Combs}_{[1,1]}$, then Defender just moves $s_\emptyset \xrightarrow{a_1}_{\mathfrak{F}} s_{[1,1]}$ and wins by Point 1(b) of Lemma 37 (since $T' \not\models \psi[\nu_{T'}]$). Conversely, if Attacker moves $s_\emptyset \xrightarrow{a_1}_{\mathfrak{F}} s_{[1,1]}$, then Defender responds $T \xrightarrow{\tau^* a_1} T'$ for some $T' \in \mathsf{Combs}_{[1,1]}$ and wins analogously. □

6 Applications: PA-processes

PA is a common generalization of basic process algebras and basic parallel processes, but is incomparable to pushdown systems and Petri nets [16]. PA has found applications in the interprocedural dataflow analysis of parallel programs [10]. It is known that over PA model checking EF-logic and strong bisimulation against finite systems are decidable in time $\mathsf{TOWER}(O(n))$ (cf. [15, 16, 20]), but it is open whether these upper bounds are optimal. In this section, we will briefly mention how to adapt our upper bound techniques to obtain elementary complexity for model checking EF_1 as well as strong bisimulation against finite systems over the class of PA-processes. That our techniques can also be adapted to PA is not that surprising since PA can be viewed as term-rewrite systems and have similar types of rewrite rules as GTRS (cf. [15]).

6.1 Definition of PA-processes

We review the basic definitions, following the presentation of [15]: we initially distinguish terms that are equivalent up to simplification laws. Fix a finite set $\mathsf{VAR} = \{X, Y, Z, \ldots\}$ of process variables. *Process terms* over VAR, denoted by $\mathcal{F}_{\mathsf{VAR}}$, are generated by the grammar:

$$t, t' \;:=\; 0 \;\mid\; X,\;\; X \in \mathsf{VAR} \;\mid\; t.t' \;\mid\; t \| t'$$

where 0 denotes a "nil" process, and $t.t'$ and $t \| t'$ are sequential and parallel compositions, respectively. Process terms can be viewed as trees over the ranked alphabet A consisting of two binary node labels $\{\|, \cdot\}$ and nullary node labels $\{0\} \cup \mathsf{VAR}$. In particular, inner nodes are always labeled by '.' or '$\|$', while leaves are labeled by elements in $\mathsf{VAR} \cup \{0\}$. Observe that $\mathcal{F}_{\mathsf{VAR}}$ is a regular tree language, for which a small NTA can be easily computed from any given VAR. A PA *declaration* over $\mathbb{A}$ is a tuple $\mathcal{P} = (\mathbb{A}, \mathcal{F}_{\mathsf{VAR}}, \Delta)$, where Δ is a finite of rewrite rules of the form (X, a, t), where $X \in \mathsf{VAR}$, $a \in \mathbb{A}$, and $t \in \mathcal{F}_{\mathsf{VAR}}$. We set $\mathsf{Dom}(\Delta) = \{X : (X, a, t) \in \Delta, \text{ for some } t \in \mathcal{F}_{Var} \text{ and } a \in \mathbb{A}\}$, and $\mathsf{VAR}_\emptyset = \mathsf{VAR} \setminus \mathsf{Dom}(\Delta)$. A PA declaration $\mathcal{P}$ generates a transition relation $\mathfrak{S}(\mathcal{P}) = (S, \mathbb{A}, \{\xrightarrow{a} \mid a \in \mathbb{A}\})$, where $S = \mathcal{F}_{\mathsf{VAR}}$ and $\xrightarrow{a}$ is the smallest relation satisfying following inference rules:

$$\frac{t_1 \xrightarrow{a} t_1'}{t_1 \| t_2 \xrightarrow{a} t_1' \| t_2} \qquad \frac{t_1 \xrightarrow{a} t_1'}{t_1 . t_2 \xrightarrow{a} t_1' . t_2} \qquad \frac{}{X \xrightarrow{a} t}\ (X,a,t) \in \Delta$$

$$\frac{t_2 \xrightarrow{a} t_2'}{t_1 \| t_2 \xrightarrow{a} t_1 \| t_2'} \qquad \frac{t_2 \xrightarrow{a} t_2'}{t_1 . t_2 \xrightarrow{a} t_1 . t_2'}\ t_1 \in \text{IsNil}$$

Here IsNil is the set of "terminated" process terms, i.e., those in which all variables are in $\mathsf{VAR}_\emptyset$. PA is very similar to GTRS in the sense that rewrite rules rewrite subtrees. However, this rewriting can only be done if this subtree t has no ancestor which is a right child of a '$\cdot$'-labeled node whose left child is not a member of IsNil.

In the study of PA processes, it is common to use a structural equivalence on process terms. Let $\equiv$ be the smallest equivalence relation on $\mathcal{F}_{Var}$ that satisfies the following:

$$t.0 \equiv t \qquad 0.t \equiv t \qquad t \| 0 \equiv t \qquad t \| t' \equiv t' \| t$$
$$(t \| t') \| t'' \equiv t \| (t' \| t'') \qquad (t.t').t'' \equiv t.(t'.t'')$$

We let $[t]_\equiv$ stand for the equivalence class of t and $[L]_\equiv$ for $\bigcup_{t \in L} [t]_\equiv$. We write $L/\equiv$ for $\{[t]_\equiv : t \in L\}$. It was shown in [15] that, for each $t \in \mathcal{F}_{Var}$, $[t]_\equiv$ is a regular tree language of size exponential in t, although the set $[L]_\equiv$ need not be regular even for regular L. Given a PA declaration $\mathcal{P}$ and the transition system $\mathfrak{S}(\mathcal{P}) = (S, \mathbb{A}, \{\xrightarrow{a} | \ a \in \mathbb{A}\})$ generated by $\mathcal{P}$, the equivalence $\equiv$ generates a new transition system $(\mathfrak{S}_\mathcal{P}/\equiv) = (S', \mathbb{A}, \{\stackrel{a}{\Longrightarrow} | \ a \in \mathbb{A}\})$, where $S' = [S]/\equiv$, and $[t]_\equiv \stackrel{a}{\Longrightarrow} [u]_\equiv$ iff there exist $t' \in [t]_\equiv$ and $u' \in [u]_\equiv$ such that $t' \xrightarrow{a} u'$. We need the following result:

Lemma 39 ([15]). *The relation $\equiv$ is bisimulation: for all $t, t', u \in \mathcal{F}_{Var}$, if $t \equiv t'$ and $t \xrightarrow{a} u$, then there exists $u' \in \mathcal{F}_{Var}$ such that $t' \xrightarrow{a} u'$ and $u \equiv u'$.*

This lemma allows us to directly reduce EF-logic model checking as well as strong bisimilarity checking against finite systems over PA modulo $\equiv$ to PA-processes (i.e. simply forget the congruence $\equiv$).

6.2 Two elementary upper bounds

The main result in this section is the following theorem:

Theorem 40. *Model checking an* EF*-formula of the form $\langle \Sigma^* \rangle \varphi$, where φ is an* EF_0 *formula, is in* NEXP *over PA.*

An immediate corollary of this theorem is the following result, which can be proven in the same way as Corollary 10.

Corollary 41. *Model checking* EF_1 *over PA is in* $\mathsf{P}^{\mathsf{NEXP}}$.

Another corollary of Theorem 40 is the following theorem, which can be proven in the same way as Theorem 24.

Theorem 42. *Strong bisimilarity checking of PA against finite systems is in* coNEXP.

The proof of Theorem 40 is analogous to the proof of Theorem 9 and so we will describe the differences. Let us suppose that $\langle \Sigma^* \rangle \varphi$ is the given formula, $\mathcal{P} = (\mathbb{A}, \mathcal{F}_{\mathsf{VAR}}, \Delta)$, and a tree $T_0 \in \mathcal{F}_{\mathsf{VAR}}$. We wish to check whether $(\mathfrak{S}(\mathcal{P}), T_0) \models \langle \Sigma^* \rangle \varphi$. We may use the result of [15, 10] that $\mathsf{post}^{\Sigma^*}_\mathcal{P}(A)$, for a tree automaton A describing PA terms, can be computed in polynomial time. A crucial observation now is that if $T \in \mathsf{post}^{\Sigma^*}_\mathcal{P}(T_0)$, then T *cannot* have a subtree of the form $t_1.t_2$ such that $t_1 \not\equiv \epsilon$ and t_2 is *not* a subtree of T_0 or a subtree that occurs in a rule of $\mathcal{P}$; the reason for this being that, since $t_1 \not\equiv \epsilon$, t_2 must be either a subtree of T_0 or a result of an application of a rewrite rule in $\mathcal{P}$. Therefore, when following the approach of the proof of Theorem 9 for proving Theorem 40, we may safely omit such unreachable configurations when defining the equivalence relation $\equiv_i$. Since a tree automaton for $[T_0]_\equiv$ can be computed in exponential time, by Lemma 39 it suffices for each of our equivalence classes corresponding to $\equiv_i$ to subsume all reachable configurations *modulo* $\equiv$.

Let S be the set of all subtrees of T_0 and trees occuring in rules in $\mathcal{P}$. Note that $|S|$ is linear in $|T_0|$ and $|\mathcal{P}|$. Let $\Gamma = S \cup \{||\} \cup \mathsf{Dom}(\Delta)$ be a ranked alphabet where S are unary labels, $||$ a binary label, and $\mathsf{Dom}(\Delta)$ be 0-ary labels. In this way, each tree in $\mathsf{post}^*_{\mathcal{P}}(T_0)$ can be represented as a tree over Γ as follows: recursively replace each subtree of the form $t_1.t_2$ (in infix notation) by $t_2(t_1)$ (i.e. t_2 being a node in the new alphabet Γ is a parent of the tree t_1). If r is the nesting depth of φ, we may adapt the proof of Theorem 9 as follows. Define N_r to be r. The equivalence class $\equiv_i$ can be defined over trees over Γ (and therefore over reachable PA terms) in the same way as in the proof of Theorem 9: we look at subtrees with at most N_i non-nil atomic terms that occur in $\mathcal{P}$, counting them up to threshold N_i, and connect them via nodes labeled $||$ and S. The rest of the proof is very similar to the proof of Theorem 9.

6.3 $\mathsf{P}^{\mathsf{NEXP}}$ lower bound

We could also use the $\mathsf{P}^{\mathsf{NEXP}}$ lower bound technique for the GTRS to show that model checking EF_1 over PA processes is hard for $\mathsf{P}^{\mathsf{NEXP}}$.

Theorem 43. *Model checking* EF_1 *over PA-processes is hard for* $\mathsf{P}^{\mathsf{NEXP}}$.

To this end, we need to show that model checking formulas of the form $EF(\varphi)$, where $\varphi \in \mathsf{EF}_0$ is NEXP-hard. The proof of this can be adapted from the proof of Theorem 20 as follows. We replace the definition of a superleaf by a comb-shaped tree whose internal nodes are labeled by sequential compositions (i.e. '.') and whose leaves are labeled by bit information, as well as tile type, of the coordinates of squares in the $2^n \times 2^n$ region. Furthermore, the internal nodes of the guessed huge tree (above the superleaves) are all labeled by parallel compositions (i.e. '$||$'). The rest of the proof is similar.

References

1. E. Allender, M. Koucky, D. Ronneburger and S. Roy. The Pervasive Reach of Resource-Bounded Kolmogorov Complexity in Computational Complexity Theory. To appear at *JCSS'10*.
2. A. Bouajjani, J. Esparza and O. Maler. Reachability Analysis of Pushdown Automata: Application to Model-Checking. In *CONCUR'97*, p. 135–150.
3. A. Bouajjani, M. Müller-Olm and T. Touili. Regular Symbolic Analysis of Dynamic Networks of Pushdown Systems. In *CONCUR'05*, p. 473–487.
4. W. S. Brainerd. Tree Generating Regular Systems. *Information and Control*, 14(2):217–231, 1969.
5. D. Caucal. On infinite transition graphs having a decidable monadic theory. *TCS* 290(1):79–115, 2003.
6. H. Comon *et al.* *Tree Automata Techniques and Applications*. Available at: `http://www.grappa.univ-lille3.fr/tata`.
7. J. -L. Coquidé, M. Dauchet, R. Gilleron and S. Vágvölgyi. Bottom-Up Tree Pushdown Automata: Classification and Connection with Rewrite Systems. *Theor. Comput. Sci.*, 127(1):69–98, 1994.
8. M. Dauchet, T. Heuillard, P. Lescanne and S. Tison. Decidability of the Confluence of Finite Ground Term Rewrite Systems and of Other Related Term Rewrite Systems. *Inf. Comput.*, 88(2):187–201, 1990.
9. M. Dauchet and S. Tison. The Theory of Ground Rewrite Systems is Decidable. In *LICS'90*, p. 242–248.
10. J. Esparza and A. Podelski. Efficient algorithms for pre* and post* on interprocedural parallel flow graphs. In *POPL'00*, p. 1–11.
11. P. Jancar, A. Kucera and R. Mayr. Deciding bisimulation-like equivalences with finite-state processes. *Theor. Comput. Sci.*, 258(1–2):409–433, 2001.
12. V. Kahlon and A. Gupta. On the analysis of interacting pushdown systems. In *POPL'07*, p. 303–314.
13. A. Kucera and R. Mayr. On the Complexity of Checking Semantic Equivalences between Pushdown Processes and Finite-state Processes. Inf. Comput., 208(7):772–796, 2010.
14. C. Löding. *Infinite Graphs Generated by Tree Rewriting*. PhD Thesis, RWTH Aachen, 2003.
15. D. Lugiez and Ph. Schnoebelen. The regular viewpoint on PA-processes. *Theor. Comput. Sci.*, 274(1–2):89–115,2002.
16. R. Mayr. Decidability and Complexity of Model Checking Problems for Infinite-State Systems. PhD thesis, TU-Munich, 1998.
17. D. E. Muller and P. E. Schupp. The Theory of Ends, Pushdown Automata, and Second-Order Logic. *Theor. Comput. Sci.*, 37:51–75, 1985.
18. C. Papadimitriou. Computational Complexity. *Addison Wesley, 1994*.
19. S. Qadeer and J. Rehof. Context-Bounded Model Checking of Concurrent Software. In *TACAS'05*, p. 93–107.

20. J. Srba. Roadmap of Infinite Results. *Bulletin of the EATCS*, 78:163–175, 2002. Updated version: `http://www.brics.dk/~srba/roadmap/`
21. L. J. Stockmeyer. *The complexity of decision problems in automata theory and logic*. PhD Thesis, MIT, 1974.
22. S. Tison. Fair Termination is Decidable for Ground Systems. In *RTA'89*, p. 462–476.
23. A. W. To and L. Libkin. Algorithmic Metatheorems for Decidable LTL Model Checking over Infinite Systems. In FoSSaCS'10, p. 221–236.
24. A. W. To. *Model Checking Infinite-State Systems: Generic and Specific Approaches*. PhD Thesis, University of Edinburgh, 2010.
25. I. Walukiewicz. Pushdown Processes: Games and Model Checking. In *CAV'96*, pages 62–74.
26. I. Walukiewicz. Model Checking CTL Properties of Pushdown Systems. In FSTTCS'00, p.127–138.

APPPENDIX

Definition of the bottom-up computation for φ

Before we define these rewriting rules, we have to introduce the notion of a protocol.
A *protocol* is a sequence $\pi = \pi_1 \cdots \pi_l$ over the alphabet $2^{[1,2n]} \times \{0,1\}$ such that

- If $\pi_i = (I, \alpha)$, then $I \neq \emptyset$ for each $i \in [1, l]$.
- If $\pi_i = (I_i, \alpha_i)$ and $\pi_j = (I_j, \alpha_j)$, then $I_i \cap I_j = \emptyset$ for each $0 \leq i < j \leq l$.

Observe that a protocol has length $0 \leq l \leq 2n$ and that the empty word ε is a protocol. By Π we denote the set of all protocols. By $I(\pi)$ we denote $\bigcup_{i \in [1,l]} I_i$ we denote the set of all variable indices that have been assigned. For each $I \subseteq [1, 2n]$ we denote by $\Pi_I = \{\pi \in \Pi \mid I(\pi) = I\}$ the set of all *I-protocols*.
Fix some word $w = w_1 \cdots w_m$ with $w_i \in \{0,1\}$ for each $i \in [1, m]$ and some valuation $\nu : I \to [1, m]$ for some set of variable indices $I \subseteq [1, 2n]$. Moreover, let $m_1 < m_2 \cdots < m_k$ be a strictly increasing enumeration of $\mathrm{ran}(\nu)$. The pair (w, ν) then corresponds to the (unique) protocol $\pi(w, \nu) = (\nu^{-1}(m_1), w_{m_1}) \cdots (\nu^{-1}(m_k), w_{m_k})$. In other words, a protocol not only collects the subset of variable indices that were assigned by ν, it additionally contains the information in which order the variables occur in the word and which $0/1$-value they have. In particular, when $|\nu^{-1}(m_i)| \geq 2$, then at position m_i of w the valuation ν assigned at least two distinct variable (positions) to m_i. Moreover if ν assigns all variables some position (i.e. $I = [1, 2n]$), then the corresponding protocol is a $[1, 2n]$-protocol. Note that this $[1, 2n]$-protocol suffices for determining w satisfies φ under ν. Hence we write $\pi \models \varphi$ whenever $\overline{w} \models \varphi[\nu]$. Analogously every comb $T \in \mathsf{Combs}_I$ defines a unique protocol $\pi(T)$. In particular, for each $T \in \mathsf{Combs}_{[1,k]}$ and we have $T \models \varphi[\nu_T]$ if and only if $\pi(T) \models \varphi$.
We extend B_0 with the following nullary symbols:

$$\mathfrak{Q}(\varphi) \quad = \quad \{q_{\mathsf{rej}}(\varphi)\} \cup \{q_\pi \mid \pi \in \Pi\}$$

For each leaf label $I \in 2^{[1,2n]} \times \{0,1\}$ we define

$$\pi(I, \alpha) \quad = \quad \begin{cases} \varepsilon & \text{if } I = \emptyset \\ (I, \alpha) & \text{otherwise} \end{cases}$$

In case $I = \bot$ or $I' = \bot$ or $I \cap I' \neq \emptyset$ we introduce the rule

$$\texttt{INIT}(\varphi) \qquad \begin{array}{ccc} & \bigstar & \\ (I,\alpha) & & (I',\alpha') \end{array} \quad \overset{\varphi}{\hookrightarrow} \quad q_{\mathsf{rej}}(J, l)$$

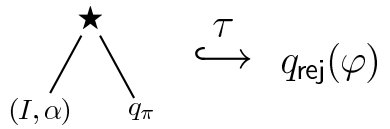
and otherwise

$$\texttt{INIT}(\varphi) \qquad \begin{array}{ccc} & \bigstar & \\ (I,\alpha) & & (I',\alpha') \end{array} \quad \overset{\varphi}{\hookrightarrow} \quad q_{\pi(I,\alpha)\cdot\pi(I',\alpha')}$$

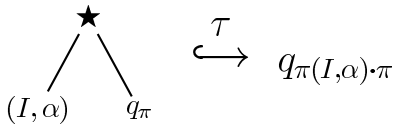
Next, we define the rules

$$\begin{array}{ccc} & \bigstar & \\ (I,\alpha) & & q_{\mathsf{rej}}(\varphi) \end{array} \quad \overset{\tau}{\hookrightarrow} \quad q_{\mathsf{rej}}(\varphi)$$

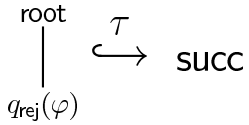
In case $I = \bot$ or $I \cap I(\pi) \neq \emptyset$ we have the rules



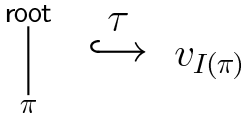
and otherwise the rules



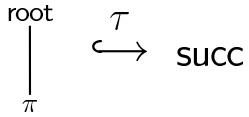
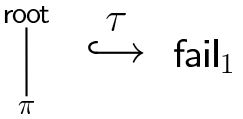
We introduce the following final rule:



In case $\pi \in \Pi_I$ where $I \subset [1, 2n]$ we add



In case $\pi \in \Pi_{[1,2n]}$ and $\pi \not\models \varphi$ we add



Finally, in case $\pi \in \Pi_{[1,2n]}$ and $\pi \models \varphi$ we add