Ordered Tree-Pushdown Systems

Lorenzo Clemente[*1], Paweł Parys[†2], Sylvain Salvati[3], and Igor Walukiewicz[‡4]

1,2 University of Warsaw, Poland
3,4 CNRS, Université de Bordeaux, INRIA, France

Abstract

We define a new class of pushdown systems where the pushdown is a tree instead of a word. We allow a limited form of lookahead on the pushdown conforming to a certain ordering restriction, and we show that the resulting class enjoys a decidable reachability problem. This follows from a preservation of recognizability result for the backward reachability relation of such systems. As an application, we show that our simple model can encode several formalisms generalizing pushdown systems, such as ordered multi-pushdown systems, annotated higher-order pushdown systems, the Krivine machine, and ordered annotated multi-pushdown systems. In each case, our procedure yields tight complexity.

1998 ACM Subject Classification D.1.1 [*Model checking*]: Software/Program Verification; D.2.4 [*Applicative (Functional) Programming*]: Programming techniques; F.1.1 [*Automata*]: Models of Computation; F.3.1 [*Specifying and Verifying and Reasoning about Programs*]: Mechanical verification; F.4.1 [*Lambda calculus and related systems*]: Mathematical Logic;

Keywords and phrases reachability analysis, saturation technique, pushdown automata, ordered pushdown automata, higher-order pushdown automata, higher-order recursive schemes, simply-typed lambda calculus, Krivine machine

Digital Object Identifier 10.4230/LIPIcs.xxx.yyy.p

1 Introduction

Context. Modeling complex systems requires to strike the right balance between the accuracy of the model, and the complexity of its analysis. A successful example is given by *pushdown systems*, which are a popular class of infinite-state systems arising in diverse contexts, such as language processing, data-flow analysis, security, computational biology, and program verification. Many interesting analyses reduce to checking reachability in pushdown systems, which can be decided in PTIME using, e.g., the popular *saturation technique* [5, 13] (cf. also the recent survey [10]). Pushdown systems have been generalized in several directions. One of them are *tree-pushdown systems* [14], where the pushdown is a tree instead of a word. Unlike for ordinary pushdown systems, non-destructive lookahead on the tree pushdown leads to undecidability. In this work we propose an ordering condition permitting a limited non-destructive lookahead on a tree pushdown.

A seemingly unrelated generalization is *ordered multi-pushdown systems* [6, 3, 2], where several linear pushdowns are available instead of just one. Since already two unrestricted

* This work was partially supported by the National Science Center (decision DEC-2013/09/B/ST6/01575)
† This work was partially supported by the National Science Center (decision DEC-2012/07/D/ST6/02443)
‡ This work was partially supported by the Technische Universität München – Institute for Advanced Study, funded by the German Excellence Initiative and the European Union Seventh Framework Programme, grant n. 291763.

Conference title on which this volume is based on.
Editors: Billy Editor and Bill Editors; pp. 1–29

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

linear pushdowns can simulate a Turing machine, an ordering restriction is put on popping transitions, requiring that all pushdowns smaller than the popped one are empty. Reachability in this model is 2-EXPTIMEc [3].

Higher-order pushdown systems provide another type of generalization. Here pushdowns can be nested inside other pushdowns [22, 19]. *Collapsible pushdown systems* [20, 16] additionally enrich pushdown symbols with *collapse links* to inner sub-pushdowns. This allows the automaton to push a new symbol and to save, at the same time, the current context in which the symbol is pushed, and to later return to this context via a collapse operation. *Annotated pushdown systems* [7] (cf. also [18]) provide a simplification of collapsible pushdown systems by replacing collapse links with arbitrary pushdown annotations[1]. The *Krivine machine* [23] is a related model which evaluates terms in simply-typed λY-calculus. Reachability in all these models is $(\mathsf{n}-1)$-EXPTIMEc [7, 23] (where n is the order of nesting pushdowns/functional parameters), and one exponential higher in the presence of alternation. Even more general, *ordered annotated multi-pushdown systems* [15] have several annotated pushdown systems under an ordering restriction similar to [3] in the first-order case. They subsume both ordered multi-pushdown systems and annotated pushdown systems. The saturation method (cf. [10]) has been adapted to most of these models, and it is the basis of the prominent MOPED tool [12] for the analysis of pushdown systems, as well as the C-SHORe model-checker for annotated pushdown systems [8].

Contributions. Motivated by a unification of the results above, we introduce *ordered tree-pushdown systems*. These are tree-pushdown systems with a limited destructive lookahead on the pushdown. We introduce an order between pushdown symbols, and we require that, whenever a sub-pushdown is read, all sub-pushdowns of smaller order must be discarded. The obtained model is expressive enough to simulate all the systems mentioned above, and is still not Turing-powerful thanks to the ordering condition. Our contributions are: i) A general preservation of recognizability result for ordered tree-pushdown systems. ii) A conceptually simple saturation algorithm working on finite tree automata representing sets of configurations (instead of more ad-hoc automata models), subsuming and unifying previous constructions. iii) A short and simple correctness proof. iv) Direct encodings of several popular extensions of pushdown systems, such as ordered multi-pushdown systems, annotated pushdown systems, the Krivine machine, and ordered annotated multi-pushdown systems. v) Encoding of our model into Krivine machines with states, that in turn are equivalent to collapsible pushdown automata. vi) A complete complexity characterization of reachability in ordered tree-pushdown systems and natural subclasses thereof.

Related work. Our work can be seen as a generalization of the saturation method for collapsible pushdown automata [7] to a broader class of rewriting systems. This method has been already generalized in [15] to multi-stack higher-order systems; in particular for ordered, phase-bounded, and scope-bounded restrictions. Another related work is a saturation method for recursive program schemes [9]. Schemes are equivalent to λY-calculus, so our formalism can be used to obtain a saturation method for schemes.

Ordered tree-pushdown systems proposed in the present paper unify these approaches.

[1] Collapsible and annotated systems generate the same configuration graphs when started from the same initial configuration, since new annotations can only be created to sub-pushdowns of the current pushdown. However, annotated pushdown systems have a richer backward reachability set which includes non-constructible pushdowns.

The encodings of the above mentioned systems are direct and work step-to-step. By contrast, the encoding of the Krivine machine to higher-order pushdowns is rather sophisticated [16, 25], and even more so its proof of correctness. The converse encoding of annotated higher-order pushdowns into Krivine machines is conceptually easier, but technically quite long for at least two reasons: a state has to be encoded by a tuple of terms, and transitions of the automaton need to be implemented with beta-reduction.

Concerning multi-pushdown systems, there exist restrictions that we do not cover in this paper. In [15] decidability is proved for annotated multi-pushdowns with phase-bounded and scope-bounded restrictions. For standard multi-pushdown systems, split-width has been proposed as a unifying restriction [11].

Outline. In Sec. 2 we introduce common notions. In Sec. 3 we define our model and we present our saturation-based algorithm to decide reachability. In Sec. 4 we show that ordered systems can optimally encode several popular formalisms. In Sec. 5 we discuss the notion of safety from the Krivine machine and higher-order pushdown automata, and how it relates to our model. In Sec. 6 we conclude with some perspectives on open problems.

2 Preliminaries

We work with rewriting systems on ranked trees, and with alternating tree automata. The novelty is that every letter of the ranked alphabet will have an order. A tree has the order determined by the letter in the root. The order itself is used to constrain rewriting rules.

An *alternating transition system* is a tuple $\mathcal{S} = \langle \mathcal{C}, \rightarrow \rangle$, where $\mathcal{C}$ is the set of configurations and $\rightarrow \subseteq \mathcal{C} \times 2^{\mathcal{C}}$ is the alternating transition relation. For two sets of configurations $A, B \subseteq \mathcal{C}$ we define $A \rightarrow_1 B$ iff, for every $c \in A$, either $c \in B$, or there exists $C \subseteq B$ s.t. $c \rightarrow C$, and we denote by $\rightarrow_1^*$ its reflexive and transitive closure. The set of *predecessors* of a set of configurations $C \subseteq \mathcal{C}$ is $\mathrm{Pre}^*(C) = \{c \mid \{c\} \rightarrow_1^* C\}$.

Ranked trees. Let $\mathbb{N}$ be the set of non-negative integers, and let $\mathbb{N}_{>0}$ be the set of strictly positive integers. A *node* is an element $u \in \mathbb{N}_{>0}^*$. A node u is a *child* of a node v if $u = v \cdot i$ for some $i \in \mathbb{N}_{>0}$. A *tree domain* is a non-empty prefix-closed set of nodes $D \subseteq \mathbb{N}_{>0}^*$ s.t., if $u \cdot (i+1) \in D$, then $u \cdot i \in D$ for every $i \in \mathbb{N}_{>0}$. A *leaf* is a node u in D without children. A *ranked alphabet* is a pair (Σ, rank) of a set of symbols Σ together with a ranking function $\mathsf{rank} : \Sigma \to \mathbb{N}$. A Σ-*tree* is a function $t : D \to \Sigma$, where D is a tree domain, s.t., for every node u in D labelled with a symbol $t(u)$ of rank k, u has precisely k children. For a Σ-tree $t : D \to \Sigma$ and a label $a \in \Sigma$, let $t^{-1}(a) = \{u \in D \mid t(u) = a\}$ be the set of nodes labelled with a. For a tree t and a node u therein, the *subtree* of t at u is defined as expected. We denote by $\mathcal{T}(\Sigma)$ the set of Σ-trees.

Order of a tree. In this paper we will give a restriction on a tree rewriting system guaranteeing that $\mathrm{Pre}^*(C)$ is regular for every regular set C. This restriction will use the notion of an order of a tree. The order of a tree is simply determined by the order of the symbol in the root. Therefore, we suppose that our alphabet Σ comes with a function $\mathsf{ord} : \Sigma \to \mathbb{N}$. The *order* of a tree t is $\mathsf{ord}(t) := \mathsf{ord}(t(\varepsilon))$.

Rewriting. Let $\mathcal{V}_0, \mathcal{V}_1, \ldots$ be pairwise disjoint infinite sets of variables; and let $\mathcal{V} = \bigcup_n \mathcal{V}_n$. We consider the extended alphabet $\Sigma \cup \mathcal{V}$ where a variable $\mathsf{x} \in \mathcal{V}_n$ has rank 0 and order n. We will work with the set $\mathcal{T}(\Sigma, \mathcal{V})$ of $(\Sigma \cup \mathcal{V})$-trees. For such a tree t, let $\mathcal{V}(t)$ be the set

of variables appearing in it. We say that t is *linear* if each variable in $\mathcal{V}(t)$ appears exactly once in t. For some $(\Sigma \cup \mathcal{V})$-tree u, t is *u-ground* if $\mathcal{V}(t) \cap \mathcal{V}(u) = \varnothing$. A *substitution* is a finite partial mapping $\sigma : \mathcal{V} \to \mathcal{T}(\Sigma \cup \mathcal{V})$ respecting orders, i.e., $\mathsf{ord}(\sigma(\mathsf{x})) = \mathsf{ord}(\mathsf{x})$. Given a $(\Sigma \cup \mathcal{V})$-tree t and a substitution σ, $t\sigma$ is the $(\Sigma \cup \mathcal{V})$-tree obtained by replacing each variable x in t in the domain of σ with $\sigma(\mathsf{x})$. A *rewrite rule* over Σ is a pair $l \to r$ of $(\Sigma \cup \mathcal{V})$-trees l and r s.t. $\mathcal{V}(r) \subseteq \mathcal{V}(l)$ and l is linear.[2]

Alternating tree automata. An *alternating tree automaton* (or just *tree automaton*) is a tuple $\mathcal{A} = \langle \Sigma, Q, \Delta \rangle$ where Σ is a finite ranked alphabet, Q is a finite set of states, and $\Delta \subseteq Q \times \Sigma \times (2^Q)^*$ is a set of alternating transitions of the form $p \xrightarrow{a} P_1 \cdots P_n$, with a of rank n. We say that $\mathcal{A}$ is *non-deterministic* if, for every transition as above, all P_j's are singletons, and we omit the braces in this case. An automaton is *ordered* if, for every state p and symbols a, b s.t. $p \xrightarrow{a} \cdots$ and $p \xrightarrow{b} \cdots$, we have $\mathsf{ord}(a) = \mathsf{ord}(b)$. We assume w.l.o.g. that automata are ordered, and we denote by $\mathsf{ord}(p)$ the order of state p. The transition relation is extended to a set of states $P \subseteq Q$ by defining $P \xrightarrow{a} P_1 \cdots P_n$ iff, for every $p \in P$, there exists a transition $p \xrightarrow{a} P_1^p \cdots P_n^p$, and $P_j = \bigcup_{p \in P} P_j^p$ for every $j \in \{1, \ldots, n\}$. It will be useful later in the definition of the saturation procedure to define run trees not just on ground trees, but also on trees possibly containing variables. A variable of order k is treated like a leaf symbol which is accepted by all states of the same order. Let $P \subseteq Q$ be a set of states, and let $t : D \to (\Sigma \cup \mathcal{V})$ be an input tree. A *run tree from P on t* is a 2^Q-tree[3] $s : D \to 2^Q$ over the same tree domain D s.t. $s(\varepsilon) = P$, and: i) if $t(u) = a$ is not a variable and of rank n, then $s(u) \xrightarrow{a} s(u \cdot 1) \cdots s(u \cdot n)$, and ii) if $t(u) = \mathsf{x}$ then $\forall p \in s(u), \mathsf{ord}(p) = \mathsf{ord}(\mathsf{x})$. The *language* recognized by a set of states $P \subseteq Q$, denoted by $\mathcal{L}(P)$, is the set of Σ-trees t s.t. there exists a run tree from P on t.

3 Ordered tree-pushdown systems

We introduce a generalization of pushdown systems, where the pushdown is a tree instead of a word. An *alternating ordered tree-pushdown system* (AOTPS) of order $n \in \mathbb{N}_{>0}$ is a tuple $\mathcal{S} = \langle n, \Sigma, P, \mathcal{R} \rangle$ where Σ is an ordered alphabet containing symbols of order at most n, P is a finite set of *control locations*, and $\mathcal{R}$ is a set of rules of the form $p, l \to S, r$ s.t. $p \in P$ and $S \subseteq P$. Moreover, $l \to r$ is a rewrite rule over Σ of one of the two forms:

$$\textit{(shallow)}\colon\ a(u_1, \ldots, u_m) \to r \quad \text{or} \quad \textit{(deep)}\colon\ a(u_1, \ldots, u_k, b(v_1, \ldots, v_{m'}), u_{k+1}, \ldots, u_m) \to r$$

where each u_i, v_j is either r-ground or a variable, and for the second form we require

$$\textit{(ordering condition)}\colon\ \text{if } \mathsf{ord}(u_i) \leqslant \mathsf{ord}(b), \text{ then } u_i \text{ is } r\text{-ground; for } i = 1, \ldots, m.$$

The rules in $\mathcal{R}$ where $l \to r$ is of the first form are called *shallow*, the others are *deep*. The tree $b(v_1, \ldots, v_{m'})$ in a deep rule is called the *lookahead subtree* of l. A rule $l \to r$ is *flat* if each u_i, v_j is just a variable. Let $\mathcal{R}_{\mathsf{ord}(b)}$ be the set of deep rules, where the lookahead symbol b is of order $\mathsf{ord}(b)$. For example, $a(\mathsf{x}, \mathsf{y}) \to c(a(\mathsf{x}, \mathsf{y}), \mathsf{x})$ is shallow and flat, but $a(b(\mathsf{x}), \mathsf{y}) \to c(\mathsf{x}, \mathsf{y})$ is deep (and flat); here necessarily $\mathsf{ord}(\mathsf{y}) > \mathsf{ord}(b)$. Finally, $a(c, d, \mathsf{x}) \to b(\mathsf{x})$ is not flat since c

[2] Notice that we require that all the variables appearing on the r.h.s. r also appear on the l.h.s. l. All our results carry over even by allowing some variables on the r.h.s. r not to appear on the l.h.s. l, but we forbid this for simplicity of presentation.

[3] Strictly speaking 2^Q does not have a rank/order. It is easy to duplicate each subset at every rank/order to obtain an ordered alphabet, which we avoid for simplicity.

and d are not variables. In Sec. 4 we provide more examples of such rewrite rules by encoding many popular formalisms. While l must be linear, r may be non-linear, thus sub-trees can be duplicated. The *size* of $\mathcal{S}$ is $|\mathcal{S}| := |\Sigma| + |P| + |\mathcal{R}|$, where $|\mathcal{R}| := \sum_{(p,l\to S,r)\in\mathcal{R}}(1+|l|+|S|+|r|)$.

Rewrite rules induce an alternating transition system $\langle \mathcal{C}_\mathcal{S}, \to_\mathcal{S}\rangle$ by root rewriting. The set of configurations $\mathcal{C}_\mathcal{S}$ consists of pairs (p,t) with $p \in P$ and $t \in \mathcal{T}(\Sigma)$, and, for every configuration (p,t), set of control locations $S \subseteq P$, and tree u, $(p,t) \to_\mathcal{S} S \times \{u\}$ if there exists a rule $((p,l) \to (S,r)) \in \mathcal{R}$ and a substitution σ s.t. $t = l\sigma$ and $u = r\sigma$.

Let $\mathcal{A} = \langle \Sigma, Q, \Delta\rangle$ be a tree automaton s.t. $P \subseteq Q$. The *language of configurations* recognized by $\mathcal{A}$ from P is $\mathcal{L}(\mathcal{A}, P) := \{(p,t) \in \mathcal{C} \mid p \in P \text{ and } t \in \mathcal{L}(p)\}$. Given an initial configuration $(p_0, t_0) \in \mathcal{C}$ and a tree automaton $\mathcal{A}$ recognizing a regular set of target configurations $\mathcal{L}(\mathcal{A}, P) \subseteq \mathcal{C}$, the *reachability problem* for $\mathcal{S}$ amounts to determining whether $(p_0, t_0) \in \mathrm{Pre}^*(\mathcal{L}(\mathcal{A}, P))$.

3.1 Reachability analysis

We present a saturation-based procedure to decide reachability in AOTPSs. This also shows that backward reachability relation preserves regularity.

▸ **Theorem 1** (Preservation of recognizability). *Let $\mathcal{S}$ be an order-n AOTPS and let C be regular set of configurations. Then, $\mathrm{Pre}^*(C)$ is effectively regular, and an automaton recognizing it can be built in n-fold exponential time.*

Let $\mathcal{S} = \langle n, \Sigma, P, \mathcal{R}\rangle$ be an AOTPS. The target set C is given as a tree automaton $\mathcal{A} = \langle \Sigma, Q, \Delta\rangle$ s.t. $\mathcal{L}(\mathcal{A}, P) = C$. W.l.o.g. we assume that in $\mathcal{A}$ initial states (states in P) have no incoming transitions. Classical saturation algorithms for pushdown automata proceed by adding transitions to the original automaton $\mathcal{A}$, until no more new transitions can be added. Here, due to the lookahead of the l.h.s. of deep rules, we need to also add new states to the automaton. However, the total number of new states is bounded once the order of the AOTPS is fixed, which guarantees termination. We construct a tree automaton $\mathcal{B} = \langle \Sigma, Q', \Delta'\rangle$ recognizing $\mathrm{Pre}^*(\mathcal{L}(\mathcal{A}, P))$, where Q' is obtained by adding states to Q, and Δ' by adding transitions to Δ, according to a saturation procedure described below.

For every rule $(p, l \to S, r) \in \mathcal{R}$ and for every subtree v of l we create a new state p^v of the same order as v recognizing all Σ-trees that can be obtained by replacing variables in v by arbitrary trees, i.e., $\mathcal{L}(p^v) = \{v\sigma \mid \sigma : \mathcal{V} \to \mathcal{T}(\Sigma),\ v\sigma \in \mathcal{T}(\Sigma)\}$; recall that the substitution should respect the order. Let Q_0 be the set of such p^v's, and let Δ_0 contain the required transitions. Notice that $|Q_0|, |\Delta_0| \leqslant |\mathcal{R}|$.

In order to deal with deep rules we add new states in the following stratified way. Let $Q'_{n+1} = Q \cup Q_0$. We define sets $Q'_n, \ldots, Q'_1$ inductively starting with Q'_n. Assume that Q'_{i+1} is already defined. We make Q'_i contain Q'_{i+1}. Then we add to Q'_i states for every deep rule $g \in \mathcal{R}_i$ of the form $p, a(u_1, \ldots, u_k, b(\ldots), u_{k+1}, \ldots, u_m) \to S, r$, with $\mathsf{ord}(b) = i$. For simplicity of notation, let us suppose that $u_1, \ldots, u_k$ are of order at most $\mathsf{ord}(b)$, and that $u_{k+1}, \ldots, u_m$ are of order strictly greater than $\mathsf{ord}(b)$[4]. We add to Q'_i states:

$$(g, P_{k+1}, \ldots, P_m) \in Q'_i \qquad \text{for all } P_{k+1}, \ldots, P_m \subseteq Q'_{i+1}.$$

In particular, to Q_n we add states of the form (g) since n is the maximal order. We define the set of states in $\mathcal{B}$ to be $Q' := Q'_1$.

[4] This assumption is w.l.o.g. since one can always add shallow rules to reorder subtrees and put them in the required form.

We add transitions to $\mathcal{B}$ in an iterative process until no more transitions can be added. During the saturation process, we maintain the following invariant: *For $1 \leqslant i \leqslant n$, states in $Q'_i \backslash Q'_{i+1}$ recognize only trees of order i.* Therefore, $\mathcal{B}$ is also an ordered tree automaton. Formally, Δ' is the least set containing $\Delta \cup \Delta_0$ and closed under adding transitions according to the following procedure. Take a deep rule

$$g = (p, a(u_1, \ldots, u_k, b(v_1, \ldots, v_{m'}), u_{k+1}, \ldots, u_m) \to S, r) \in \mathcal{R}_{\mathsf{ord}(b)}$$

and assume as before that the order of u_j is at most $\mathsf{ord}(b)$ for $j \leqslant k$, and strictly bigger than $\mathsf{ord}(b)$ otherwise. We consider a run tree t from S on r in $\mathcal{B}$. For every $j = 1, \ldots, m$ we set: $P^t_j = \{p^{u_j}\}$ if u_j is r-ground, and $P^t_j = \bigcup t(r^{-1}(\mathsf{x}))$ if $u_j = \mathsf{x}$ is a variable appearing in r. The set $\bigcup t(r^{-1}(\mathsf{x}))$ collects all states of $\mathcal{B}$ from which the subtree for which x can be replaced must be accepted. Moreover, for the lookahead subtree $b(v_1, \ldots, v_{m'})$, we let $P^t_b = \{(g, P^t_{k+1}, \ldots, P^t_m)\}$. Analogously, we define $S^t_1, \ldots, S^t_{m'}$ considering $v_1, \ldots, v_{m'}$ instead of $u_1, \ldots, u_m$. Then, we add two transitions:

$$p \xrightarrow{a} P^t_1 \cdots P^t_k P^t_b P^t_{k+1} \cdots P^t_m \qquad \text{and} \qquad (g, P^t_{k+1}, \ldots, P^t_m) \xrightarrow{b} S^t_1 \ldots S^t_{m'} \,. \tag{1}$$

Thanks to the ordering condition, $P^t_{k+1}, \ldots, P^t_m \subseteq Q'_{\mathsf{ord}(b)+1}$, so $(g, P^t_{k+1}, \ldots, P^t_m)$ is indeed a state in $Q'_{\mathsf{ord}(b)}$. For a shallow rule g the procedure is the same but ignoring the part about the $b(v_1, \ldots, v_{m'})$ component; so only one rule is added in this case.

▸ **Lemma 2** (Correctness of saturation). *For $\mathcal{A}$ and $\mathcal{B}$ be as above, $\mathcal{L}(\mathcal{B}, P) = \mathrm{Pre}^*(\mathcal{L}(\mathcal{A}, P))$.*

The correctness proof, even though short, is presented in App. A. The right-in-left inclusion is by straightforward induction on the number of rewrite steps to reach $\mathcal{L}(\mathcal{A}, P)$. The left-in-right inclusion is more subtle, but with an appropriate invariant of the saturation process it also follows by a direct inspection.

3.2 Complexity

The reachability problem for AOTPSs can be solved using the saturation procedure from Theorem 1. For an initial configuration $(p_0, t_0) \in \mathcal{C}$ and an automaton $\mathcal{A}$ recognizing a regular set of target configurations $\mathcal{L}(\mathcal{A}, P)$, we construct $\mathcal{B}$ as in the previous section, and then test $(p_0, t_0) \in \mathcal{L}(\mathcal{B}, P)$. In this section we will analyze the complexity of this procedure in several relevant cases. All lower-bounds follow from the reductions presented in Sec. 4.

Let $m > 1$ be the maximal rank of any symbol in Σ. Using the notation from the previous subsection, we have that $\left|Q'_{n+1}\right| \leqslant |Q| + |\mathcal{R}|$, $|Q'_n| \leqslant \left|Q'_{n+1}\right| + |\mathcal{R}|$, and for every $k \in \{1, \ldots, n-1\}$, $|Q'_k| \leqslant \left|Q'_{k+1}\right| + |\mathcal{R}| \cdot 2^{(m-1)\cdot\left|Q'_{k+1}\right|} \leqslant O\left(|\mathcal{R}| \cdot 2^{(m-1)\cdot\left|Q'_{k+1}\right|}\right)$, and thus $|Q'| \leqslant \mathsf{exp}_{n-1}(O((m-1)\cdot(|Q| + |\mathcal{R}|)))$, where $\mathsf{exp}_0(x) = x$ and, for $i \geqslant 0$, $\mathsf{exp}_{i+1}(x) = 2^{\mathsf{exp}_i(x)}$. The size of the transition relation is at most one exponential more than the number of states, thus $|\Delta'| \leqslant \mathsf{exp}_n(O((m-1)\cdot(|Q| + |\mathcal{R}|)))$. This implies:

▸ **Theorem 3.** *Reachability in order-n AOTPSs is* n-EXPTIMEc.

We identify four subclasses of AOTPSs, for which the reachability problem is of progressively decreasing complexity. First, we can save one exponential if we consider control-state reachability for the class of *non-deterministic, flat* AOTPSs. A system is *non-deterministic* when for every rule $p, l \to S, r$, the set S is a singleton. A system is *flat* when its rules $p, l \to S, r$ are flat (defined on page 4). *Control-state reachability* of a given set of locations $T \subseteq P$ means that the language of final configurations is $T \times \mathcal{T}(\Sigma)$. A proof of the theorem below is presented in App. B.

▸ **Theorem 4.** *Control-state reachability in order-n non-deterministic flat AOTPSs is* $(\mathsf{n}-1)$-EXPTIMEc*, where* $n \geqslant 2$*.*

Second, we consider the class of *linear* non-deterministic systems. Suppose that we consider *non-deterministic reachability*, i.e., that $\mathcal{A}$ is non-deterministic. When $\mathcal{S}$ is linear, i.e., variables in the r.h.s. of rules in $\mathcal{R}$ appear exactly once, then all P_i^t's and S_i^t's in (1) are singletons, and thus $\mathcal{B}$ is also non-deterministic. Consequently, the only states from $Q'_i \backslash Q'_{i+1}$ that are used by rewriting rules have the form $(g, \{p_{k+1}\}, \ldots, \{p_m\})$ for $p_{k+1}, \ldots, p_m \in Q'_{i+1}$. Therefore, there are at most $O((|Q| + |\mathcal{R}|)^{(m-1)^n})$ states and $O(|\mathcal{R}| \cdot |Q'|^m)$ transitions, and $\mathcal{B}$ is thus doubly exponential in n.

▸ **Theorem 5.** *Non-deterministic reachability in linear non-deterministic AOTPSs is* 2-EXPTIMEc*.*

The next simplification is when the system is *shallow* in the sense that it does not have deep rules. In this case we do not need to add states recursively ($Q' := Q \cup Q_0$), and we thus avoid the multiple exponential blow-up. Similarly, when the system is *unary*, i.e., the maximal rank is $m = 1$, only polynomially many states are added.

▸ **Theorem 6.** *Reachability in shallow as well as in unary AOTPSs is* EXPTIMEc*.*

If moreover the system is non-deterministic, then we get PTIME complexity, provided the rank of the letters in the alphabet is bounded.

▸ **Theorem 7.** *Non-deterministic reachability in unary non-deterministic AOTPSs and in shallow non-deterministic AOTPSs of fixed rank is in* PTIME*.*

3.3 Expressiveness

In the next section we give a number of examples of systems that can be directly encoded in AOTPSs. Before that, we would like to underline that AOTPSs can themselves be encoded into collapsible pushdown systems. We formally formulate this equivalence in terms of Krivine machines with states, which are defined later in Sec. 4.3. The details of this reduction are presented in App. E.

▸ **Theorem 8.** *Every AOTPS of order n can be encoded in a Krivine machine with states of the same level s.t. every rewriting step of the AOTPS corresponds to a number of reduction steps of the Krivine machine.*

Since parity games over the configuration graph of the Krivine machine with states are known to be decidable [24], this equivalence yields decidability of parity games over AOTPSs. However, in this paper we concentrate on reachability properties of AOTPSs, which are decidable thanks to our simple saturation algorithm from Sec. 3.1. No such saturation algorithm was previously known for the Krivine machine with states.

4 Applications

In this section, we give several examples of systems that can be encoded as AOTPSs. Ordinary alternating pushdown systems (and even prefix-rewrite systems) can be easily encoded as *unary* AOTPSs by viewing a word as a linear tree; the ordering condition is trivial since symbols have rank $\leqslant 1$. Moreover, *tree-pushdown systems* [14] can be seen as *shallow* AOTPSs. By Theorem 6, reachability is in EXPTIME for both classes, and, by Theorem 7, it reduces to PTIME for the non-alternating variant (for fixed maximal rank).

In the rest of the section, we show how to encode four more sophisticated classes of systems, namely *ordered multi-pushdown systems* (Sec. 4.1), *annotated higher-order pushdown systems* (Sec. 4.2), the *Krivine machine with states* (Sec. 4.3), and *ordered annotated multi-pushdown systems* (Sec. 4.4), and we show that reachability for these models (except the last one) can be decided with tight complexity bounds using our conceptually simple saturation procedure.

4.1 Ordered multi-pushdown systems

In an ordered multi-pushdown system there are n pushdowns. Symbols can be pushed on any pushdown, but only the first non-empty pushdown can be popped [6, 3, 2]. This is equivalent to saying that to pop a symbol from the k-th pushdown, the contents of the previous pushdowns $1, \dots, k-1$ should be discarded. Formally, an *alternating ordered multi-pushdown system* is a tuple $\mathcal{O} = \langle n, \Gamma, Q, \Delta \rangle$, where $n \in \mathbb{N}_{>0}$ is the *order* of the system (i.e., the number of pushdowns), Γ is a finite pushdown alphabet, Q is a finite set of control locations, and $\Delta \subseteq Q \times O_n \times 2^Q$ is a set of rules of the form (p, o, P) with $p \in Q$, $P \subseteq Q$, and o a pushdown operation in $O_n := \{\mathsf{push}_k(a), \mathsf{pop}_k(a) \mid 1 \leqslant k \leqslant n, a \in \Gamma\}$. We say that $\mathcal{O}$ is *non-deterministic* when P is a singleton for every rule. A multi-pushdown system induces an alternating transition system $\langle \mathcal{C}_\mathcal{O}, \rightarrow_\mathcal{O} \rangle$ where the set of configurations is $\mathcal{C}_\mathcal{O} = Q \times (\Gamma^*)^n$, and the transitions are defined as follows: for every $(p, \mathsf{push}_k(a), P) \in \Delta$ there exists a transition $(p, w_1, \dots, w_n) \rightarrow_\mathcal{O} P \times \{(w_1, \dots, a \cdot w_k, \dots, w_n)\}$, and for every $(p, \mathsf{pop}_k(a), P) \in \Delta$ there exists a transition $(p, w_1, \dots, a \cdot w_k, \dots, w_n) \rightarrow_\mathcal{O} P \times \{(\varepsilon, \dots, \varepsilon, w_k, \cdots, w_n)\}$. For $c \in \mathcal{C}_\mathcal{O}$ and $T \subseteq Q$, the *(control-state) reachability problem* for $\mathcal{O}$ asks whether $c \in \mathrm{Pre}^*(T \times (\Gamma^*)^n)$.

Encoding. We show that an ordered multi-pushdown system can be simulated by an AOTPS. The idea is to encode the k-th pushdown as a linear tree of order k, and to encode a multi-pushdown as a tree of linear pushdowns. Let $\perp$ and $\bullet$ be two new symbols not in Γ, let $\Gamma_\perp = \Gamma \cup \{\perp\}$, and let $\Sigma = (\Gamma_\perp \times \{1, \dots, n\}) \cup \{\bullet\}$ be an ordered alphabet, where a symbol $(a, i) \in \Gamma_\perp \times \{i\}$ has order i, rank 1 if $a \in \Gamma$ and rank 0 if $a = \perp$. Moreover, $\bullet$ has rank n and order 1. For simplicity, we write a^i instead of (a, i). A multi-pushdown $w_1, \dots, w_n$, where each $w_j = a_{j,1} \dots a_{j,n_j}$ is encoded as the tree $\mathsf{enc}(w_1, \dots, w_n) := \bullet(a^1_{1,1}(a^1_{1,2}(\dots \perp^1)), \dots, a^n_{n,1}(a^n_{n,2}(\dots \perp^n)))$. For an ordered multi-pushdown system $\mathcal{O} = \langle n, \Gamma, Q, \Delta \rangle$ we define an equivalent AOTPS $\mathcal{S} = \langle n, \Sigma, Q, \mathcal{R} \rangle$ with Σ defined as above, and set of rules $\mathcal{R}$ defined as follows (we use the convention that variable x_k has order k): For every push rule $(p, \mathsf{push}_k(a), P) \in \Delta$, we have a rule $(p, \bullet(\mathsf{x}_1, \dots, \mathsf{x}_n) \rightarrow P, \bullet(\mathsf{x}_1, \dots, a^k(\mathsf{x}_k), \dots, \mathsf{x}_n)) \in \mathcal{R}$, and for every pop rule $(p, \mathsf{pop}_k(a), P) \in \Delta$, we have $(p, \bullet(\mathsf{x}_1, \dots, a^k(\mathsf{x}_k), \dots, \mathsf{x}_n) \rightarrow P, \bullet(\perp^1, \dots, \perp^{k-1}, \mathsf{x}_k, \mathsf{x}_{k+1}, \dots, \mathsf{x}_n)) \in \mathcal{R}$. Both kinds of rules above are linear, and the latter one satisfies the ordering condition since lower-order variables $\mathsf{x}_1, \dots, \mathsf{x}_{k-1}$ are discarded. It is easy to see that $(p, w_1, \dots, w_n) \rightarrow^*_\mathcal{O} P \times \{(w'_1, \dots, w'_n)\}$ if, and only if, $(p, \mathsf{enc}(w_1, \dots, w_n)) \rightarrow^*_\mathcal{S} P \times \{\mathsf{enc}(w'_1, \dots, w'_n)\}$. Thus, the encoding preserves reachability properties. By Theorem 3, we obtain an n-EXPTIME upper-bound for reachability in alternating multi-pushdown systems of order n. Moreover, since $\mathcal{S}$ is linear, and since $\mathcal{S}$ is non-deterministic when $\mathcal{O}$ is non-deterministic, by Theorem 5 we recover the optimal 2-EXPTIMEc complexity proved by [3] (cf. also [2]).

▸ **Theorem 9** ([3]). *Reachability in alternating ordered multi-pushdown systems is in* n-EXPTIME. *Reachability in non-deterministic ordered multi-pushdown systems is* 2-EXPTIMEc.

Reachability for the alternating variant of the model (in n-EXPTIME) was not previously known.

4.2 Annotated higher-order pushdown systems

Let Γ be a finite pushdown alphabet. In the following, we fix an order $n \geqslant 1$, and we let $1 \leqslant k \leqslant n$ range over orders. For our purpose, it is convenient to expose the topmost pushdown at every order recursively.[5] We define Γ_k, the set of *annotated higher-order pushdowns (stacks) of order k*, simultaneously for all $k \in \{1, \dots, n\}$, as the least set containing the empty pushdown $\langle\rangle$, and, whenever $u_1 \in \Gamma_1, \dots, u_k \in \Gamma_k$, $v_j \in \Gamma_j$ for some $j \in \{1, \dots, n\}$, then $\langle a^{v_j}, u_1, \dots, u_k\rangle \in \Gamma_k$. Similarly, if we do not consider stack annotations v_j's, we obtain the set of *higher-order pushdowns of order k*. Operations on annotated pushdowns are as follows. The operation push_k^b pushes a symbol $b \in \Gamma$ on the top of the topmost order-1 stack and annotates it with the topmost order-k stack, push_k duplicates the topmost order-$(k-1)$ stack, pop_k removes the topmost order-$(k-1)$ stack, and $\mathsf{collapse}_k$ replaces the topmost order-k stack with the order-k stack annotating the topmost symbol:

$$\mathsf{push}_k^b(\langle a^u, u_1, \dots, u_n\rangle) = \langle b^{\langle a^u, u_1, \dots, u_k\rangle}, \langle a^u, u_1\rangle, u_2, \dots, u_n\rangle,$$
$$\mathsf{push}_k(\langle a^u, u_1, \dots, u_n\rangle) = \langle a^u, u_1, \dots, u_{k-1}, \langle a^u, u_1, \dots, u_k\rangle, u_{k+1}, \dots, u_n\rangle,$$
$$\mathsf{pop}_k(\langle a^u, v_1, \dots, v_{k-1}, \langle b^v, u_1, \dots, u_k\rangle, u_{k+1}, \dots, u_n\rangle) = \langle b^v, u_1, \dots, u_n\rangle,$$
$$\mathsf{collapse}_k(\langle a^{\langle b^v, v_1, \dots, v_k\rangle}, u_1, \dots, u_n\rangle) = \langle b^v, v_1, \dots, v_k, u_{k+1}, \dots, u_n\rangle.$$

Let $O_n = \bigcup_{k=1}^n \{\mathsf{push}_k^b, \mathsf{push}_k, \mathsf{pop}_k, \mathsf{collapse}_k \mid b \in \Gamma\}$ be the set of stack operations. Similarly, one can define operations push^b and pop_k on stacks without annotations (but not $\mathsf{collapse}_k$, or push_k^b). An *alternating order-n annotated pushdown system* is a tuple $\mathcal{P} = \langle n, \Gamma, Q, \Delta\rangle$, where Γ is a finite stack alphabet, Q is a finite set of control locations, and $\Delta \subseteq Q \times \Gamma \times O_n \times 2^Q$ is a set of rules. An *alternating order-n pushdown system* (i.e., without annotations) is as $\mathcal{P}$ above, except that we consider non-annotated stack and operations on non-annotated stacks. An annotated pushdown system induces a transition system $\langle \mathcal{C}_\mathcal{P}, \to_\mathcal{P}\rangle$, where $\mathcal{C}_\mathcal{P} = Q \times \Gamma_n$, and the transition relation is defined as $(p, w) \to_\mathcal{P} P \times \{w'\}$ whenever $(p, a, o, P) \in \Delta$ with $w = \langle a^u, \cdots\rangle$ and $w' = o(w)$. Thus, a rule (p, a, o, P) first checks that the topmost stack symbol is a, and then applies the transformation provided by the stack operation o to the current stack (which may, or may not, change the topmost stack symbol a). Given $c \in \mathcal{C}_\mathcal{P}$ and $T \subseteq Q$, the *(control-state) reachability problem* for $\mathcal{P}$ asks whether $c \in \mathrm{Pre}^*(T \times \Gamma_n)$.

Encoding. We represent annotated pushdowns as trees. Let Σ be the ordered alphabet containing, for each $k \in \{1, \dots, n\}$, an end-of-stack symbol $\bot^k \in \Sigma$ of rank 0 and order k. Moreover, for each $a \in \Gamma$ and order $k \in \{1, \dots, n\}$, there is a symbol $\langle a, k\rangle \in \Sigma$ of order k and rank $k+1$ representing the root of a tree encoding a stack of order k. An order-k stack is encoded as a tree recursively by $\mathsf{enc}_k(\langle\rangle) = \bot^k$ and $\mathsf{enc}_k(\langle a^u, u_1, \dots, u_k\rangle) = \langle a, k\rangle(\mathsf{enc}_i(u), \mathsf{enc}_1(u_1), \dots, \mathsf{enc}_k(u_k))$, where i is the order of u. Let $\mathcal{P} = \langle n, \Gamma, Q, \Delta\rangle$ be an annotated pushdown system. We define an equivalent AOTPS $\mathcal{S} = \langle n, \Sigma, Q, \mathcal{R}\rangle$, where Σ is as defined above, and $\mathcal{R}$ contains a rule $p, l \to P, r$ for each rule in $(p, a, o, P) \in \Delta$ and orders m, m_1, where $l \to r$ is as follows (cf. also Fig. 1 in the appendix for a pictorial representation). We use the convention that a variable subscripted by i has order i, and we write $\mathsf{x}_{i..j}$ for $(\mathsf{x}_i, \dots, \mathsf{x}_j)$, and similarly for $\mathsf{z}_{i..j}$:

[5] Our definition is equivalent to [7].

$$\begin{aligned}
&\langle a,n\rangle(\mathsf{y}_m,\mathsf{x}_{1..n}) \to \langle b,n\rangle(\langle a,k\rangle(\mathsf{y}_m,\mathsf{x}_{1..k}),\langle a,1\rangle(\mathsf{y}_m,\mathsf{x}_1),\mathsf{x}_{2..n}) && \text{if } o=\mathsf{push}^b_k,\\
&\langle a,n\rangle(\mathsf{y}_m,\mathsf{x}_{1..n}) \to \langle a,n\rangle(\mathsf{y}_m,\mathsf{x}_{1..k-1},\langle a,k\rangle(\mathsf{y}_m,\mathsf{x}_{1..k}),\mathsf{x}_{k+1..n}) && \text{if } o=\mathsf{push}_k,\\
&\langle a,n\rangle(\mathsf{z}'_{m_1},\mathsf{z}_{1..k-1},\langle b,k\rangle(\mathsf{y}_m,\mathsf{x}_{1..k}),\mathsf{x}_{k+1..n}) \to \langle b,n\rangle(\mathsf{y}_m,\mathsf{x}_{1..n}) && \text{if } o=\mathsf{pop}_k,\\
&\langle a,n\rangle(\langle b,k\rangle(\mathsf{y}_m,\mathsf{x}_{1..k}),\mathsf{z}_{1..k},\mathsf{x}_{k+1..n}) \to \langle b,n\rangle(\mathsf{y}_m,\mathsf{x}_{1..n}) && \text{if } o=\mathsf{collapse}_k.
\end{aligned}$$

The last two rules satisfy the ordering condition of AOTPSs since only higher-order variables $\mathsf{x}_{k+1},\dots,\mathsf{x}_n$ are not discarded. It is easy to see that $(p,w) \to^*_{\mathcal{P}} P\times\{w'\}$ if, and only if, $(p,\mathsf{enc}_n(w)) \to^*_{\mathcal{S}} P\times\{\mathsf{enc}_n(w')\}$. Consequently, the encoding preserves reachability properties. Since an annotated pushdown system of order n is simulated by a flat AOTPS of the same order, the following complexity result is an immediate consequence of Theorems 3 and 4.

▸ **Theorem 10** ([7]). *Reachability in alternating annotated pushdown systems of order n and in non-deterministic annotated pushdown systems of order $n+1$ is* n-EXPTIMEc.

4.3 Krivine machine with states

We show that the Krivine machine evaluating simply-typed λY-terms can be encoded as an AOTPS. Essentially, this encoding was already given in the presentation of the Krivine machine operating on λY-terms from [23], though not explicitly given as tree pushdowns. In this sense, this provides the first saturation algorithm for the Krivine machine, thus yielding an optimal reachability procedure. Moreover, in App. E we present also a converse reduction (as announced earlier in Theorem 8), thus showing that the two models are in fact equivalent.

A *type* is either the basic type 0 or $\alpha\to\beta$ for types α,β. The *level* of a type is $\mathsf{level}(0)=0$ and $\mathsf{level}(\alpha\to\beta)=\max(\mathsf{level}(\alpha)+1,\mathsf{level}(\beta))$. We abbreviate $\alpha\to\cdots\to\alpha\to\beta$ as $\alpha^k\to\beta$. Let $\mathcal{V}=\{x_1^{\alpha_1},x_2^{\alpha_2},\dots\}$ be a countably infinite set of typed variables, and let Γ be a ranked alphabet. A *term* is either (i) a constant $a^{0^k\to 0}\in\Gamma$, (ii) a variable $x^\alpha\in\mathcal{V}$, (iii) an abstraction $(\lambda x^\alpha.M^\beta)^{\alpha\to\beta}$, (iv) an application $(M^{\alpha\to\beta}N^\alpha)^\beta$, or (v) a fixpoint $(YM^{\alpha\to\alpha})^\alpha$. We sometimes omit the type annotation from the superscript, in order to simplify the notation. For a given term M, its set of *free variables* is defined as usual. A term M is *closed* if it does not have any free variable. We denote by $\Lambda(M)$ be the set of *sub-terms* of M. An *environment* ρ is a finite type-preserving function assigning closures to variables, and a *closure* C^α is a pair consisting of a term of type α and an environment, as expressed by the following mutually recursive grammar: $\rho ::= \varnothing \mid \rho[x^\alpha\mapsto C^\alpha]$ and $C^\alpha ::= (M^\alpha,\rho)$. We say that a closure (M,ρ) is *valid* if ρ binds all variables which are free in M (and no others), and moreover $\rho(x^\alpha)$ is itself a valid closure for each free variable x^α in M. Sometimes, we need to restrict an environment ρ by discarding some bindings in order to turn a closure (M,ρ) into a valid one. Given a term M and an environment ρ, the *restriction* of ρ to M, denoted $\rho\big|_M$, is obtained by removing from ρ all bindings for variables which are not free in M. In this way, if (M,ρ) is a closure where ρ assigns valid closures to at least all variables which are free in M, then $(M,\rho\big|_M)$ is a valid closure. In a closure (M,ρ), M is called the *skeleton*, and it determines the type and level of the closure. Let $Cl^\alpha(M)$ be the set of valid closures of type α with skeleton in $\Lambda(M)$. An *alternating Krivine machine*[6] *with states* of level $l\in\mathbb{N}_{>0}$ is a tuple $\mathcal{M}=\langle l,\Gamma,Q,K^0,\Delta\rangle$, where $\langle\Gamma,Q,\Delta\rangle$ is an alternating tree automaton (in which a constant $a^{0^k\to 0}\in\Gamma$ is seen as a letter a of rank k), and K^0 is a closed term of type 0 s.t. the level of any sub-term in $\Lambda(K^0)$ is at most l. In the following, let $\alpha=\alpha_1\to\cdots\to\alpha_k\to 0$. The Krivine machine $\mathcal{M}$ induces a transition system $\langle\mathcal{C}_\mathcal{M},\to_\mathcal{M}\rangle$, where in a configuration $(p,C^\alpha,C_1^{\alpha_1},\dots,C_k^{\alpha_k})\in\mathcal{C}_\mathcal{M}$,

[6] Cf. also [21] for a definition of the Krivine machine in a different context.

$p \in Q$, $C^\alpha \in Cl^\alpha(K^0)$ is the *head closure*, and $C_1^{\alpha_1} \in Cl^{\alpha_1}(K^0), \dots, C_k^{\alpha_k} \in Cl^{\alpha_k}(K^0)$ are the *argument closures*. The transition relation $\to_{\mathcal{M}}$ depends on the structure of the skeleton of the head closure. It is deterministic except when the head is a constant in Γ, in which case the transitions in Δ control how the state changes (cf. also Fig. 2 in the appendix for a pictorial representation):

$$\begin{aligned}
(p, (x^\alpha, \rho), C_1^{\alpha_1}, \dots, C_k^{\alpha_k}) &\to_{\mathcal{M}} \{(p, \rho(x^\alpha), C_1^{\alpha_1}, \dots, C_k^{\alpha_k})\},\\
(p, (M^\alpha N^{\alpha_1}, \rho), C_2^{\alpha_2}, \dots, C_k^{\alpha_k}) &\to_{\mathcal{M}} \{(p, (M^\alpha, \rho|_{M^\alpha}), (N^{\alpha_1}, \rho|_{N^{\alpha_1}}), C_2^{\alpha_2}, \dots, C_k^{\alpha_k})\},\\
(p, (YM^{\alpha\to\alpha}, \rho), C_1^{\alpha_1}, \dots, C_k^{\alpha_k}) &\to_{\mathcal{M}} \{(p, (M^{\alpha\to\alpha}, \rho), ((YM)^\alpha, \rho), C_1^{\alpha_1}, \dots, C_k^{\alpha_k})\},\\
(p, (\lambda x^{\alpha_0}.M^\alpha, \rho), C_0^{\alpha_0}, \dots, C_k^{\alpha_k}) &\to_{\mathcal{M}} \{(p, (M^\alpha, \rho[x^{\alpha_0} \mapsto C_0^{\alpha_0}]), C_1^{\alpha_1}, \dots, C_k^{\alpha_k})\},\\
(p, (a^{0^k\to 0}, \rho), C_1^0, \dots, C_k^0) &\to_{\mathcal{M}} (P_1 \times \{C_1^0\}) \cup \dots \cup (P_k \times \{C_k^0\})\\
&\qquad \text{for every } p \xrightarrow{a} P_1 \cdots P_k \in \Delta.
\end{aligned}$$

We say that $\mathcal{M}$ is *non-deterministic* if $\langle \Gamma, Q, \Delta\rangle$ is non-deterministic and all letters in Γ have rank at most 1. Given $c \in \mathcal{C}_{\mathcal{M}}$ and $T \subseteq Q$, the *(control-state) reachability problem* for $\mathcal{M}$ asks whether $c \in \mathrm{Pre}^*(T \times (\bigcup_{\alpha=\alpha_1\to\dots\to\alpha_k\to 0} Cl^\alpha(K^0) \times Cl^{\alpha_1}(K^0) \times \dots \times Cl^{\alpha_k}(K^0)))$.

Encoding. Following [23], we encode valid closures and configurations of the Krivine machine as ranked trees. Fix a Krivine machine $\mathcal{M} = \langle l, \Gamma, Q, K^0, \Delta\rangle$ of level l. We assume a total order on all variables $\langle x_1^{\beta_1}, \dots, x_n^{\beta_n}\rangle$ appearing in K^0. For a type α, we define $\mathsf{ord}(\alpha) = l - \mathsf{level}(\alpha)$. We construct an AOTPS $\mathcal{S} = \langle l, \Sigma, Q', \mathcal{R}\rangle$ of order l as follows. The ordered alphabet is

$$\Sigma = \{N^\alpha \mid N^\alpha \in \Lambda(K^0) \wedge \mathsf{level}(\alpha) < l\} \cup \{[N^\alpha] \mid N^\alpha \in \Lambda(K^0)\} \cup \{\bot_i \mid i \in \{1, \dots, n\}\}.$$

Here, N^α is a symbol of $\mathsf{rank}(N^\alpha) = n$ and $\mathsf{ord}(N^\alpha) = \mathsf{ord}(\alpha)$. Moreover, if $\alpha = \alpha_1 \to \dots \to \alpha_k \to 0$ for some $k \geqslant 0$, then $[N^\alpha]$ is a symbol of $\mathsf{rank}([N^\alpha]) = n + k$ and $\mathsf{ord}([N^\alpha]) = l$ (in fact, $\mathsf{ord}([N^\alpha])$ is irrelevant, as $[N^\alpha]$ is used only in the root). Finally, $\bot_i$ is a leaf of order i. The set of control locations is $Q' = Q \cup \bigcup_{(p\xrightarrow{a} P_1\cdots P_k)\in\Delta} \{(1, P_1), \dots, (k, P_k)\}$. A closure (N^α, ρ) is encoded recursively as $\mathsf{enc}(N^\alpha, \rho) = N^\alpha(t_1, \dots, t_n)$, where, for every $i \in \{1, \dots, n\}$, i) if $x_i \in \mathsf{FV}(N^\alpha)$ then $t_i = \mathsf{enc}(\rho(x_i))$, and ii) $t_i = \bot_{\mathsf{ord}(\beta_i)}$ otherwise (recall that β_i is the type of x_i). A configuration $c = (p, (N^\alpha, \rho), C_1^{\alpha_1}, \dots, C_k^{\alpha_k})$ is encoded as the tree $\mathsf{enc}(c) = [N^\alpha](t_1, \dots, t_n, \mathsf{enc}(C_1^{\alpha_1}), \dots, \mathsf{enc}(C_k^{\alpha_k}))$, where the first n subtrees encode the closure (N^α, ρ), i.e., $\mathsf{enc}(N^\alpha, \rho) = N^\alpha(t_1, \dots, t_n)$. The encoding is extended point-wise to sets of configurations. Notice that K^0 uses only variables of level at most $l - 1$ (the subterm $\lambda x^\alpha.N$ introducing x^α is of level higher by one), so all skeletons in an environment are of order at most $l - 1$. Similarly, skeletons in argument closures are of level at most $l - 1$; only the head closure may have a skeleton of level l. Thus we do not need symbols N^α for $\mathsf{level}(\alpha) = l$.

Below, we assume that $\alpha = \alpha_1 \to \dots \to \alpha_k \to 0$, that variable y_j has order $\mathsf{ord}(\alpha_j)$ for every $j \in \{0, \dots, k\}$, and that variables x_i and z_i have order $\mathsf{ord}(\beta_i)$ for every $i \in \{1, \dots, n\}$. Notice that $\mathsf{ord}(\alpha) < \mathsf{ord}(\alpha_1), \dots, \mathsf{ord}(\alpha_k)$. Moreover, we write $\mathbf{x} = \langle \mathsf{x}_1, \dots, \mathsf{x}_n\rangle$, $\mathbf{z} = \langle \mathsf{z}_1, \dots, \mathsf{z}_n\rangle$, and $\mathbf{y} = \langle \mathsf{y}_1, \dots, \mathsf{y}_k\rangle$. Finally, by $\mathbf{x}|_M$ we mean the tuple which is the same as $\mathbf{x}$, except that positions corresponding to variables not free in M are replaced by the

symbol $\bot_{\mathsf{ord}(\beta_i)}$. $\mathcal{R}$ contains the following rules:

$$p, [x_i^\alpha](\mathsf{z}_1, \dots, \mathsf{z}_{i-1}, M^\alpha(\mathbf{x}), \mathsf{z}_{i+1}, \dots, \mathsf{z}_n, \mathbf{y}) \to \{p\}, [M^\alpha](\mathbf{x}, \mathbf{y}),$$
$$p, [M^\alpha N^{\alpha_1}](\mathbf{x}, \mathsf{y}_2, \dots, \mathsf{y}_k) \to \{p\}, [M^\alpha](\mathbf{x}|_{M^\alpha}, N^{\alpha_1}(\mathbf{x}|_{N^{\alpha_1}}), \mathsf{y}_2, \dots, \mathsf{y}_k),$$
$$p, [Y M^{\alpha\to\alpha}](\mathbf{x}, \mathbf{y}) \to \{p\}, [M^{\alpha\to\alpha}](\mathbf{x}, Y M^{\alpha\to\alpha}(\mathbf{x}), \mathbf{y}),$$
$$p, [\lambda x_i^{\alpha_0}.M^\alpha](\mathbf{x}, \mathsf{y}_0, \mathbf{y}) \to \{p\}, [M^\alpha](\mathsf{x}_1, \dots, \mathsf{x}_{i-1}, y_0, \mathsf{x}_{i+1}, \dots, \mathsf{x}_n, \mathbf{y}),$$
$$p, [a^{0^k\to 0}](\mathbf{x}, \mathbf{y}) \to \{(1, P_1), \dots, (k, P_k)\}, [a^{0^k\to 0}](\mathbf{x}, \mathbf{y}) \qquad \forall (p \xrightarrow{a} P_1 \cdots P_k) \in \Delta,$$
$$(i, P_i), [a^{0^k\to 0}](\mathbf{z}, \mathsf{y}_1, \dots, \mathsf{y}_{i-1}, M_i^0(\mathbf{x}), \mathsf{y}_{i+1}, \dots, \mathsf{y}_k) \to P_i, [M_i^0](\mathbf{x}).$$

The first rule satisfies the ordering condition since the shared variables y_i are of order strictly higher than $\mathsf{ord}(M^\alpha)$. A direct inspection of the rules shows that, for a configuration c and a set of configurations D, we have $c \to^*_{\mathcal{M}} D$ if, and only if, $\mathsf{enc}(c) \to^*_{\mathcal{S}} \mathsf{enc}(D)$. Therefore, the encoding preserves reachability properties. Since a Krivine machine of level n is simulated by a flat AOTPS of order n, the following is an immediate consequence of Theorems 3 and 4.

▸ **Theorem 11** ([1]). *Reachability in alternating Krivine machines with states of level n and in non-deterministic Krivine machines with states of level $n+1$ is* n-EXPTIMEc.

4.4 Ordered annotated multi-pushdown systems

Ordered annotated multi-pushdown systems are the common generalization of ordered multi-pushdown systems and annotated pushdown systems [15]. Such a system is comprised of $m > 0$ annotated higher-order pushdowns arranged from left to right, where each pushdown is of order $n > 0$. While push operations are unrestricted, pop and collapse operations implicitly destroy all pushdowns to the left of the pushdown being manipulated, in the spirit of [6, 3, 2]. [15] has shown that reachability in this model can be decided in mn-fold exponential time, by using a saturation-based construction leveraging on the previous analysis for the first-order case [6, 3, 2]. In App. F, we provide a simple encoding of an annotated multi-pushdown system with parameters (m, n) into an AOTPS of order mn. It is essentially obtained by taking together our previous encodings of ordered (cf. Sec. 4.1) and annotated systems (cf. Sec. 4.2). As a consequence of this encoding, by using the fact that an AOTPS of order mn can be encoded by a Krivine machine of the same level (by Theorem. 8), and by recalling the known fact that the latter can be encoded by a 1-stack annotated multi-pushdown system of order mn [25], we deduce that the concurrent behavior of an ordered m-stack annotated multi-pushdown system of order n can be *sequentialized* into a 1-stack annotated pushdown system of order mn (thus at the expense of an increase in order). The following complexity result is a direct consequence of Theorem 3.

▸ **Theorem 12** ([15]). *Reachability in alternating ordered annotated multi-pushdown systems of parameters (m, n) is in* (mn)-EXPTIME.

We remark that our result is for alternating systems, while in [15] they consider non-deterministic systems and obtain $(\mathsf{m}(\mathsf{n}-1))$-EXPTIME complexity. It seems that their method can be extended to alternating systems, and then the complexity becomes (mn)-EXPTIME as well.

5 Safety

The notion of safety has been made explicit by Knapik, Niwiński, and Urzyczyn [19] who identified the class of *safe recursive schemes*. They have shown that this class defines the

same set of infinite trees as higher-order pushdown systems, i.e., the systems from Sec. 4.2 but without annotations. Blum and Ong [4] have extended the notion of safety to the simply-typed λ-calculus in a clear way. Then [25] adapted it to λY-calculus, and have shown that safe λY-terms correspond to higher-order pushdown automata without annotation.

There is a simple notion of safety for AOTPSs that actually corresponds to safety for pushdown systems and terms. We say that a $(\Sigma \cup \mathcal{V})$-tree is *safe* when looking from the root to the leafs the order does never increase. Formally, a tree u is *safe* if every subtree t thereof has order $\mathsf{ord}(t) \leqslant \mathsf{ord}(u)$ and it is itself safe. A rewrite rule $l \to r$ is *safe* if both l and r are safe. We say that $\mathcal{S}$ is *safe* if all its rules are safe.

As a first example, let us look at the encoding of annotated higher-order pushdown systems from Sec. 4.2. If we drop annotation then higher-order pushdowns are represented by safe trees, and all the rules are safe in the sense above. The case of Krivine machines is more difficult to explain, because it would need the definition of safety from [25]. In particular, one would have to partition variables into *lambda-variables* and *Y-variables*, which we avoid in the current presentation for simplicity. In the full version of the paper we will show that safe terms are encoded by safe trees, and that all the rules of the encoding of the Krivine machine preserve safety. Finally, we remark that the translation from AOTPSs to the Krivine machine with states previously announced in Theorem 8 can be adapted to produce a safe Krivine machine with states from a safe AOTPS.

6 Conclusions

We have introduced a novel extension of pushdown automata which is able to capture several sophisticated models thanks to a simple ordering condition on the tree-pushdown. While ordered tree-pushdown systems are not more expressive than annotated higher-order pushdown systems, or than Krivine machines, they offer some conceptual advantages. Compared to Krivine machines, they have states, and typing is replaced by a lighter mechanism of ordering; for example, the translation from our model back to the Krivine machine is much more cumbersome. Compared to annotated pushdown automata, the tree-pushdown is more versatile than a higher-order stack; for example, one can compare the encoding of the Krivine machine into our model to its encoding to annotated pushdown automata. We hope that ordered tree-pushdown systems will help to establish more connections with other models, as we have done in this paper with multi-pushdown systems.

There exist restrictions of multi-pushdown systems that we do not cover in this paper. Reachability games are decidable for phase-bounded multi-pushdown systems [26]. We can encode the phase-bounded restriction directly in our tree-pushdown systems, but we do not know how to deal with the scope-bounded restriction. Encoding the scope-bounded restriction would give an algorithm for reachability games over such systems, but we do not know if the problem is decidable.

Our general saturation algorithm can be used to verify reachability properties. We plan to extend it to the more general *parity properties*, in the spirit of [17]. We leave as future work implementing our saturation algorithm, leveraging on subsumption techniques to keep the search space as small as possible.

Acknowledgments. We kindly acknowledge stimulating discussions with Irène Durand, Géraud Sénizergues, and Jean-Marc Talbot, and the anonymous reviewers for their helpful comments.

References

1 K. Aehlig. A finite semantics of simply-typed lambda terms for infinite runs of automata. *Log. Methods Comput. Sci.*, 3(1):1–23, 2007.

2 M. F. Atig. Model-checking of ordered multi-pushdown automata. *Log. Methods Comput. Sci.*, 8(3), 09 2012.

3 M. F. Atig, B. Bollig, and P. Habermehl. Emptiness of multi-pushdown automata is 2ETIME-complete. In *Proc. of DLT'08*, pages 121–133. Springer, 2008.

4 W. Blum and C.-H. L. Ong. The safe lambda calculus. *Log. Methods Comput. Sci.*, 5(1), 2009.

5 A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: Application to model-checking and saturation method. In *Proc. of CONCUR'97*, volume 1243 of *LNCS*, pages 135–150, 1997.

6 L. Breveglieri, A. Cherubini, C. Citrini, and S. Crespi-Reghizzi. Multi-push-down languages and grammars. *Int. J. Found. Comput. Sci.*, 7(3):253–292, 1996.

7 C. Broadbent, A. Carayol, M. Hague, and O. Serre. A saturation method for collapsible pushdown systems. In *Proc. of ICALP'12*, volume 7392 of *LNCS*, pages 165–176, 2012.

8 C. Broadbent, A. Carayol, M. Hague, and O. Serre. C-SHORe: A collapsible approach to higher-order verification. In *Proc. of ICFP '13*, pages 13–24. ACM, 2013.

9 C. Broadbent and N. Kobayashi. Saturation-based model checking of higher-order recursion schemes. In *In Proc. of CSL'13*, pages 129–148, 2013.

10 A. Carayol and M. Hague. Saturation algorithms for model-checking pushdown systems. In *Proc. of AFL'14*, volume 151 of *EPTCS*, pages 1–24, 5 2014.

11 A. Cyriac, P. Gastin, and K. N. Kumar. MSO decidability of multi-pushdown systems via split-width. In *In Proc. of CONCUR'12*, pages 547–561, 2012.

12 J. Esparza and S. Schwoon. A BDD-based model checker for recursive programs. In *Proc. of CAV'01*, pages 324–336. Springer-Verlag, 2001.

13 A. Finkel, B. Willems, and P. Wolper. A direct symbolic approach to model checking pushdown systems. In *Proc. of INFINITY'97*, volume 9, pages 27–37, 1997.

14 I. Guessarian. Pushdown tree automata. *Theor. Comp. Sys.*, 16:237–263, 1983.

15 M. Hague. Saturation of concurrent collapsible pushdown systems. In *Proc. of FSTTCS'13*, volume 24 of *LIPIcs*, pages 313–325, Dagstuhl, Germany, 2013.

16 M. Hague, A. Murawski, C.-H. L. Ong, and O. Serre. Collapsible pushdown automata and recursion schemes. In *Proc. of LICS'08*, pages 452–461, 2008.

17 M. Hague and C.-H. Ong. A saturation method for the modal mu-calculus over pushdown systems. *Inform. and Comput.*, 209(5):799 – 821, May 2011.

18 A. Kartzow and P. Parys. Strictness of the collapsible pushdown hierarchy. In *Proc. of MFCS'12*, pages 566–577. Springer, 2012.

19 T. Knapik, D. Niwiński, and P. Urzyczyn. Higher-order pushdown trees are easy. In *Proc. of FOSSACS'02*, volume 2303 of *LNCS*, pages 205–222, 2002.

20 T. Knapik, D. Niwiński, P. Urzyczyn, and I. Walukiewicz. Unsafe grammars and panic automata. In *Proc. of ICALP'05*, pages 1450–1461. Springer-Verlag, 2005.

21 J.-L. Krivine. A call-by-name lambda-calculus machine. *Higher Order Symbol. Comput.*, 20:199–207, September 2007.

22 A. N. Maslov. Multilevel stack automata. *Probl. Peredachi Inf.*, 12(1):55–62, 1976.

23 S. Salvati and I. Walukiewicz. Krivine machines and higher-order schemes. In *Proc. of ICALP'11*, volume 6756 of *LNCS*, pages 162–173. Springer-Verlag, 2011.

24 S. Salvati and I. Walukiewicz. Krivine machines and higher-order schemes. *Inf. Comput.*, 239(0):340–355, 2014.

25 S. Salvati and I. Walukiewicz. Simply typed fixpoint calculus and collapsible pushdown automata. *Math. Struct. in Comp. Science*, pages 1–47, 5 2015.

26 A. Seth. Global reachability in bounded phase multi-stack pushdown systems. In T. Touili, B. Cook, and P. Jackson, editors, *Proc. of CAV'10*, volume 6174 of *LNCS*, pages 615–628. Springer, 2010.

A Proof of Lemma 2

Let $\mathcal{A}$ be the automaton recognizing the target set of configurations, and let $\mathcal{B}$ be the automaton obtained at the end of the saturation procedure (cf. page 5).

▸ **Lemma 2** (Correctness of saturation). *For $\mathcal{A}$ and $\mathcal{B}$ be as above, $\mathcal{L}(\mathcal{B}, P) = \mathrm{Pre}^*(\mathcal{L}(\mathcal{A}, P))$.*

We prove the two inclusions of the lemma separately.

▸ **Lemma 13** (Completeness). *For $\mathcal{A}$ and $\mathcal{B}$ as above, $\mathrm{Pre}^*(\mathcal{L}(\mathcal{A}, P)) \subseteq \mathcal{L}(\mathcal{B}, P)$.*

Proof. Let (p, t) be a configuration in $\mathrm{Pre}^*(\mathcal{L}(\mathcal{A}, P))$. We show $(p, t) \in \mathcal{L}(\mathcal{B}, P)$ by induction on the length $d \geqslant 0$ of the shortest sequence of rewrite steps from (p, t) to $\mathcal{L}(\mathcal{A}, P)$. If $d = 0$, then $(p, t) \in \mathcal{L}(\mathcal{A}, P)$. Since the saturation procedure only adds states and transitions to $\mathcal{A}$, we directly have $(p, t) \in \mathcal{L}(\mathcal{B}, P)$. Inductively, assume that the property holds for all configurations reaching $\mathcal{L}(\mathcal{A}, P)$ in at most $d \geqslant 0$ steps, and let configuration (p, t) be at distance $d + 1 > 0$ from $\mathcal{L}(\mathcal{A}, P)$. There exists a rule $(p, l \to S, r) \in \mathcal{R}$ and a substitution σ s.t. $t = l\sigma$ and from every configuration in $S \times \{r\sigma\}$ we can reach $\mathcal{L}(\mathcal{A}, P)$ in at most d steps. Let $l = a(u_1, \ldots, u_m)$ and $t = a(t_1, \ldots, t_m)$. By induction hypothesis, $S \times \{r\sigma\} \subseteq \mathcal{L}(\mathcal{B}, P)$, thus $\mathcal{B}$ has a run tree β from S on $r\sigma$. Its part, also denoted β, is a run tree from S on r. Suppose first that our rule is shallow, and consider the transition $p \xrightarrow{a} P_1 \cdots P_m$ added to Δ' in the saturation procedure because of this rule $p, l \to S, r$ and this run tree β. This transition can be used in the root of t, so it suffices to show that $t_1 \in \mathcal{L}(P_1), \ldots, t_m \in \mathcal{L}(P_m)$. If u_i is r-ground, then $P_i = \{p^{u_i}\}$ and $t_i \in \mathcal{L}(p^{u_i})$ by construction. If $u_i = \mathsf{x}$ is a variable appearing in r, then by definition $P_i = \bigcup \beta(r^{-1}(\mathsf{x}))$. Since β is a run tree on the whole $r\sigma$, we have $t_i = \sigma(\mathsf{x}) \in \mathcal{L}(P_i)$. The case of a deep rule is similar. Let $u_k = b(v_1, \ldots, v_{m'})$ be the lookahead subtree of l that is neither r-ground nor a variable, and let $t_k = b(s_1, \ldots, s_{m'})$. Then in the root of t_k we use the second added transition $(g, P_{k+1}, \ldots, P_m) \xrightarrow{b} S_1 \cdots S_{m'}$. It suffices to show $s_1 \in \mathcal{L}(S_1), \ldots, s_{m'} \in \mathcal{L}(S_{m'})$, which is done as above. ◂

▸ **Lemma 14** (Soundness). *For $\mathcal{A}$ and $\mathcal{B}$ as above, $\mathcal{L}(\mathcal{B}, P) \subseteq \mathrm{Pre}^*(\mathcal{L}(\mathcal{A}, P))$.*

The soundness proof requires several steps. First, we assign a semantics $[\![p]\!] \subseteq \mathcal{T}(\Sigma)$ to all states p in $\mathcal{B}$. For a set of states $S \subseteq Q'$, $[\![S]\!] := \bigcap_{p \in S} [\![p]\!]$. For $p \in Q'_{n+1}$ we take

$$[\![p]\!] := \begin{cases} \{t \mid (p, t) \in \mathrm{Pre}^*(\mathcal{L}(\mathcal{A}, P))\} & \text{if } p \in P, \\ \mathcal{L}_{\mathcal{A}}(p) & \text{if } p \in Q \backslash P, \\ \mathcal{L}_{\mathcal{B}}(p) & \text{if } p = p^v \in Q_0. \end{cases}$$

Then by induction on $n - i$ we define $[\![p]\!]$ for $p = ((q, l \to S, r), P_{k+1}, \ldots, P_m) \in Q'_i \backslash Q'_{i+1}$. Let $l = a(u_1, \ldots, u_m)$, where u_k is the lookahead subtree. As $[\![p]\!]$ we take the set of trees $t_k \in [\![p^{u_k}]\!]$ s.t. for all $t_1 \in [\![p^{u_1}]\!], \ldots, t_{k-1} \in [\![p^{u_{k-1}}]\!]$ and all $t_{k+1} \in [\![P_{k+1}]\!], \ldots, t_m \in [\![P_m]\!]$ it holds $(q, a(t_1, \ldots, t_m)) \in \mathrm{Pre}^*(\mathcal{L}(\mathcal{A}, P))$. Notice that $P_{k+1}, \ldots, P_m \subseteq Q'_{i+1}$, so $[\![P_{k+1}]\!], \ldots, [\![P_m]\!]$ as well as $[\![p^{u_1}]\!], \ldots, [\![p^{u_k}]\!]$ are already defined. Second, we define sound transitions as those respecting the semantics. Formally, a transition $p \xrightarrow{a} P_1 \cdots P_m$ is *sound* iff $\forall (t_1 \in [\![P_1]\!], \ldots, t_m \in [\![P_m]\!])$, $a(t_1, \ldots, t_m) \in [\![p]\!]$.

▸ Proposition 1. If all transitions are sound, then $\mathcal{L}(p) \subseteq [\![p]\!]$ for every $p \in Q'$.

Proof. Let $t \in \mathcal{L}(p)$. We proceed by complete induction on the height of t. Let $t = a(t_1, \ldots, t_m)$ (possibly $m = 0$ if $t = a$ is a leaf). There exists a sound transition $p \xrightarrow{a} P_1 \cdots P_m$ s.t. $t_1 \in \mathcal{L}(P_1), \ldots, t_m \in \mathcal{L}(P_m)$. By induction hypothesis, $t_1 \in [\![P_1]\!], \ldots, t_m \in [\![P_m]\!]$, and thus by the definition of sound transition, $a(t_1, \ldots, t_m) \in [\![p]\!]$. ◂

▸ Proposition 2. Transitions in $\Delta \cup \Delta_0$ are sound.

Proof. Let $(p \xrightarrow{a} P_1 \cdots P_m) \in \Delta$, and let $t_1 \in [\![P_1]\!], \ldots, t_m \in [\![P_m]\!]$. Since we assume that there are no transitions back to the initial states in P, we have $P_1, \ldots, P_m \subseteq Q \backslash P$, and thus $t_1 \in \mathcal{L}_{\mathcal{A}}(P_1), \ldots, t_m \in \mathcal{L}_{\mathcal{A}}(P_m)$ by the definition of the semantics. Consequently, $t := a(t_1, \ldots, t_m) \in \mathcal{L}_{\mathcal{A}}(p)$. If $p \notin P$ we are done, since $[\![p]\!] = \mathcal{L}_{\mathcal{A}}(p)$ in this case. Otherwise, if $p \in P$ then $(p, t) \in \mathcal{L}(\mathcal{A}, P)$, which is included in $\mathrm{Pre}^*(\mathcal{L}(\mathcal{A}, P))$, and thus we have $t \in [\![p]\!]$ by definition.

For Δ_0 the situation is even simpler. Let $p \xrightarrow{a} P_1 \cdots P_m \in \Delta_0$, and let $t_1 \in [\![P_1]\!], \ldots, t_m \in [\![P_m]\!]$. By the definition of the semantics we have $t_1 \in \mathcal{L}_{\mathcal{B}}(P_1), \ldots, t_m \in \mathcal{L}_{\mathcal{B}}(P_m)$ which implies that $t := a(t_1, \ldots, t_m) \in \mathcal{L}_{\mathcal{B}}(p) = [\![p]\!]$. ◂

▸ Proposition 3. The saturation procedure adds only sound transitions.

Proof. This is induction on the order in which transitions are added by the procedure. Let $g = (p, l \to S, r)$ with $l = a(u_1, \ldots, u_m)$, and let t be a run tree in $\mathcal{B}$ from S on r. Since all transitions used in t were present in $\mathcal{B}$ earlier, they are sound. We show that the transition $p \xrightarrow{a} P_1 \cdots P_m$ as added by saturation is sound. To this end, let $t_1 \in [\![P_1]\!], \ldots, t_m \in [\![P_m]\!]$, and we show $t' := a(t_1, \ldots, t_m) \in [\![p]\!]$. Since $p \in P$, this amounts to showing that $(p, t') \in \mathrm{Pre}^*(\mathcal{L}(\mathcal{A}, P))$.

First, assume that g is shallow. Observe that $t' = l\sigma$ for some substitution σ (if u_i is r-ground then $P_i = \{p^{u_i}\}$, so $t_i \in [\![P_i]\!]$ means that t_i "matches" to u_i; if $u_i = \mathsf{x}$ is a variable appearing in r then P_i is nonempty and contains states of order $\mathsf{ord}(\mathsf{x})$, so t_i is of the same order as x). Thus the system has a transition from (p, t') to $S \times \{r\sigma\}$, and it thus suffices to show $S \times \{r\sigma\} \subseteq \mathrm{Pre}^*(\mathcal{L}(\mathcal{A}, P))$. Every node of r labeled by a variable x is labelled in the run tree t by a subset of $P_i = \bigcup t(r^{-1}(\mathsf{x}))$ for some i, and simultaneously $\sigma(\mathsf{x}) = t_i$ (recall that all variables of r have to appear in l). Since t uses only sound transitions and $t_1 \in [\![P_1]\!], \ldots, t_m \in [\![P_m]\!]$, by induction on its height we have $r\sigma \in [\![S]\!]$, which implies $S \times \{r\sigma\} \subseteq \mathrm{Pre}^*(\mathcal{L}(\mathcal{A}, P))$ by the definition of the semantics since $S \subseteq P$.

If g is deep, then $P_k = \{(g, P_{k+1}, \ldots, P_m)\}$. Recall our assumption that $u_1, \ldots, u_{k-1}$ have order at most $\mathsf{ord}(u_k)$; due to the ordering condition they are r-ground. It follows that $P_1 = \{p^{u_1}\}, \ldots, P_{k-1} = \{p^{u_{k-1}}\}$, so $t_1 \in [\![p^{u_1}]\!], \ldots, t_{k-1} \in [\![p^{u_{k-1}}]\!]$. Since $t_k \in [\![P_k]\!], \ldots, t_m \in [\![P_m]\!]$, we deduce directly from the definition of $[\![P_k]\!]$ that $(p, t') \in \mathrm{Pre}^*(\mathcal{L}(\mathcal{A}, P))$.

When g is deep, the transition $(g, P_{k+1}, \ldots, P_m) \xrightarrow{b} S_1 \cdots S_{m'}$ is additionally added for this rule, and we have to show that this transition is sound too. Let $w_1 \in [\![S_1]\!], \ldots, w_{m'} \in [\![S_{m'}]\!]$, and we show $t_k := b(w_1, \ldots, w_{m'}) \in [\![(g, P_{k+1}, \ldots, P_m)]\!]$. To this end, let $t_1 \in [\![p^{u_1}]\!], \ldots, t_{k-1} \in [\![p^{u_{k-1}}]\!]$ and $t_{k+1} \in [\![P_{k+1}]\!], \ldots, t_m \in [\![P_m]\!]$, and we show $(p, a(t_1, \ldots, t_m)) \in \mathrm{Pre}^*(\mathcal{L}(\mathcal{A}, P))$. The proof is as for a shallow rule, noticing that a node labeled in r by a variable x is labeled in t either by a subset of P_i for some $i \in \{k+1, \ldots, m\}$ (and then $\sigma(\mathsf{x}) = t_i$), or by a subset of S_j for some $j \in \{1, \ldots, m'\}$ (and then $\sigma(\mathsf{x}) = w_j$); we can again conclude that $r\sigma \in [\![S]\!]$ by induction on the height of t. ◂

Proof of Lemma 14. By Proposition 2, the initial transitions in $\Delta \cup \Delta_0$ are sound, and by Proposition 3, all transitions in Δ' are sound. Let $(p, t) \in \mathcal{L}(\mathcal{B}, P)$. Thus, $t \in \mathcal{L}(p)$. By Proposition 1, $t \in [\![p]\!]$. Since $p \in P$, by the definition of the semantics, $(p, t) \in \mathrm{Pre}^*(\mathcal{L}(\mathcal{A}, P))$. ◂

B Complexity of control-state reachability for flat non-deterministic AOTPSs

While control-state reachability (reachability of a configuration having a particular control location) for order-n annotated pushdown systems is n-EXPTIMEc (cf. Theorem 10)—and similarly for the Krivine machine (cf. Theorem 11)—it is known that if we consider *non-deterministic* annotated pushdown systems of order n, the complexity goes down to $(n-1)$-EXPTIMEc (similarly for an analogous restriction on the Krivine machine). As we will show below, this is also the case for *flat* AOTPSs.

▸ **Theorem 4.** *Control-state reachability in order-n non-deterministic flat AOTPSs is* $(n-1)$-EXPTIMEc*, where* $n \geqslant 2$.

In fact, we prove a stronger statement (cf. Theorem 15 below). Instead of control-state reachability, we consider reachability of target sets defined by a restricted class of alternating tree automata, which we call *non-n-alternating alternating tree automata*. Intuitively, this class of tree automata will be defined in such a way that

- it is preserved by the saturation procedure, and
- allows a faster running time of the procedure by saving one exponential in the number of states (and, consequently, transitions).

Formally, an alternating tree automaton $\mathcal{A} = \langle \Sigma, Q, \Delta \rangle$ is *non-n-alternating*[7] if its state space Q can be partitioned into two sets Q_- and Q_+ so that:

- for every transition $p \xrightarrow{a} P_1 \dots P_m \in \Delta$ with $p \in Q_-$ it holds $P_i \subseteq Q_-$ for all i,
- for every transition $p \xrightarrow{a} P_1 \dots P_m \in \Delta$ with $p \in Q_+$ it holds $\sum_i |P_i \cap Q_+| \leqslant 1$, and
- in Q_- there is exactly one state of order n, call it p^n, and the set of transitions from p^n is $\{p \xrightarrow{a} \varnothing \dots \varnothing \mid \mathsf{ord}(a) = n\}$.

That is, states in Q_- are closed under the transition relation. Moreover, they are either of order $\leqslant n-1$, or trivially accept every order-n tree (i.e, p^n is the unique such state in Q_-). On the other side, a state p in Q_+ can be of order n, but then it can non-trivially accept at most one subtree of order n (by staying in Q_+). When this happens, there is no alternation when doing so, i.e., this unique subtree of order n is accepted by a single state in Q_+.

▸ **Theorem 15.** *Reachability of a target set defined by a non-n-alternating alternating tree automaton in order-n non-deterministic flat AOTPSs is* $(n-1)$-EXPTIMEc*, where* $n \geqslant 2$.

In particular we can realize control-state reachability, since one can build a non-n-alternating automaton that only checks the control-state and accepts every tree. The better complexity bound described by Theorem 15 is realized by almost the same saturation procedure as in the general case; we only perform two small modifications. First, for different variables x of order n, we have separate states p^{x}. All these states do the same: they just check that the order of the node is n. Moreover, we already have such state in $\mathcal{A}$: it is called p^n in the definition of a non-n-alternating automaton. This redundancy should be eliminated: instead of using all these states we glue them together into this single state p^n. Second, we have created states p^v for every subtree v of the l.h.s. l of every rule. But we need such states only for v other than the whole l, and, for a deep rule, than the lookahead subtree of l.

[7] A similar notion for collapsible pushdown systems was proposed in [7].

For convenience, let us keep only such p^v and remove all other. It is easy to see that these modifications do not influence correctness. Thus, we only need to analyze the complexity.

First, we observe that the automaton obtained at any step of the saturation procedure is non-n-alternating, and we will later analyze its size depending on this assumption. We divide the states of $\mathcal{B}$ in Q' into Q'_- and Q'_+ as follows (as required in the definition of a non-n-alternating automaton). The original automaton $\mathcal{A}$ by assumption is non-n-alternating, which by definition gives us a partitioning of states from Q into Q_+ and Q_-. Recalling that we can assume that in $\mathcal{A}$ we do not have transitions leading to initial states (we already made and justified this same assumption when describing the saturation procedure), we assume that all initial states are in Q_+. We inherit this partitioning for states in Q'. The states from Q_0 are all taken to Q'_-. Because the system is flat, none of states p^v is of order n (we have such states only for v being a variable of order $\leqslant n-1$), and thus $\langle \Sigma, Q \cup Q_0, \Delta \cup \Delta_0 \rangle$ is non-n-alternating for this division of states. Next, by induction on $n-i$ we classify states from Q'_i. Consider a state $(g, P_{k+1}, \ldots, P_m) \in Q'_i \backslash Q'_{i+1}$. We put it into Q'_+ if $P_{k+1} \cup \cdots \cup P_m \subseteq Q'_-$; otherwise (some state from some P_i is in Q'_+), we put it into Q'_-. In particular, for $i = n$ the state is always taken to Q'_+, as necessarily $k = m$. Recall that the states in the sets P_j come from Q'_{i+1}, so for them we already know whether they are in Q'_+ or in Q'_-. We have not added any transitions, so $\langle \Sigma, Q'_1, \Delta \cup \Delta_0 \rangle$ is still non-n-alternating for the above division of states.

Now we should see that a single step of the saturation procedure preserves the property that the automaton is non-n-alternating for our division of states. We only need to check that the newly added transitions satisfy the required properties. By induction assumption we know that the automaton before the considered step is non-n-alternating. Moreover the system is non-deterministic, so $|S| = 1$ in the considered rule $p, l \to S, r$. This ensures the following property of the run tree t from S on r: State sets on only one path may contain states from Q'_+, each set at most one such state. In particular at most one leaf is labelled by such state set. Thus, at most one among the sets $\bigcup t(r^{-1}(\mathsf{x}))$ contains a state from Q'_+, and it contains at most one such state; the sets $\{p^{\mathsf{x}}\}$ do not contain states from Q'_+. For a shallow rule this is the end, as all P^t_j's are of this form (with different variables for different j's). For a deep rule we also have the special child with $P^t_b = \{(g, P^t_{k+1}, \ldots, P^t_m)\}$. When none of the P^t_j's for $j \neq k$ has a state from Q'_+, then by definition $(g, P^t_{k+1}, \ldots, P^t_m) \in Q'_+$; the transition from p is fine (recall that the initial state p belongs to Q'_+), and the transition from $(g, P^t_{k+1}, \ldots, P^t_m)$ is also fine since at most one among S^t_j's has a state from Q'_+. Suppose that some P^t_j for $i \neq k$ has a state from Q'_+. This is only possible when the corresponding subtree of l is not r-ground, so it is of order greater than the special deep subtree $b(v_1, \ldots, v_{m'})$ of l thanks to the ordering condition. Thus, this set P^t_j is listed in the state $(g, P^t_{k+1}, \ldots, P^t_m)$, and the latter by definition belongs to Q'_-. Then the transition from p is fine, and the transition from $(g, P^t_{k+1}, \ldots, P^t_m)$ is also fine since none of the S^t_j's has a state from Q'_+.

Let us now analyze the complexity of the algorithm. The original saturation algorithm by definition uses states in $Q'_1, \ldots, Q'_{n+1}$. We show that in fact a subset of those states is actually needed, by constructing a sequence $Q''_1, \ldots, Q''_{n+1}$ which is pointwise included in the former, and such that $Q''_1 \cup \cdots \cup Q''_{n+1}$ is one exponential smaller than $Q'_1 \cup \cdots \cup Q'_{n+1}$. We

define the following sets:

$$Q''_{n+1} = Q'_{n+1}, \qquad Q''_n = Q'_n,$$
$$Q''_{n-1} = Q''_n \cup \{(g, P_{k+1}, \ldots, P_m) \in Q'_{n-1} \mid \forall i. |P_i| \geqslant 1, \ \sum_i |P_i \backslash \{p^n\}| \leqslant 1\},$$
$$Q''_i = Q''_{i+1} \cup \bigcup_{g \in \mathcal{R}_i} \{g\} \times \left(2^{Q''_{i+1}}\right)^{m-k} \qquad \text{for } 1 \leqslant i \leqslant n-2,$$

where $g = (p, a(u_1, \ldots, u_k, b(v_1, \ldots, v_{m'}), u_{k+1}, \ldots, u_m) \to S, r)$ and $\mathsf{ord}(b) = i$. Thus to Q''_{n-1} we only add those states $(g, P_{k+1}, \ldots, P_m)$ where all P_i except one are equal to $\{p^n\}$, and the remaining one is either a singleton or a pair $\{q, p^n\}$ for some state q. We can see that all transitions in Δ' will only use states from Q''_1. To prove this, recall that Δ' is the least set containing $\Delta \cup \Delta_0$ and closed under applying the saturation procedure. The initial set $\Delta \cup \Delta_0$ only uses states from $Q'_{n+1} \subseteq Q''_1$. Thus, suppose that the current set of transitions uses only states from Q''_1; we should prove that transitions added by (one step of) the saturation procedure also use only states from Q''_1. Notice that all states in the considered run tree t on r appear in some transition, so they come from Q''_1. The only "new" state is $(g, P_{k+1}, \ldots, P_m)$ created for a deep rule. Let b be the root of the special subtree of the left side of g. If $\mathsf{ord}(b) \neq n-1$, then we have $(g, P_{k+1}, \ldots, P_m) \in Q''_{\mathsf{ord}(b)} \subseteq Q''_1$, since, as already observed, $P_{k+1}, \ldots, P_m \subseteq Q''_1 \cap Q'_{\mathsf{ord}(b)+1} = Q''_{\mathsf{ord}(b)+1}$. For $\mathsf{ord}(b) = n-1$, we recall that each of the sets $P_{k+1}, \ldots, P_m$ describes a subtree of order n, so it is either of the form $\{p^n\}$ or $\bigcup t(r^{-1}(\mathsf{x}))$ for some variable x of order n. But, as observed previously, at most one of these sets may contain a state from Q'_+. However (as required in an non-n-alternating automaton) all states of order n except p^n are from Q'_+. Thus $\sum_{j=k+1}^{m} |P_j \backslash \{p^n\}| \leqslant 1$, and in consequence $(g, P_{k+1}, \ldots, P_m) \in Q''_{n-1} \subseteq Q''_1$. This finishes the proof that only states from Q''_1 are used.

Notice that $|Q''_{n-1}| \leqslant |Q'_n| + |\mathcal{R}| \cdot 2(m-1) \cdot |Q'_n|$ (while $|Q'_{n-1}|$ is exponential in $|Q'_n|$), where m is the maximal rank of any symbol in Σ. Consequently, $|Q''_1| \leqslant \mathsf{exp}_{n-2}(O((|Q| + |\mathcal{R}|) \cdot m \cdot |\mathcal{R}|))$ and $|\Delta'| \leqslant \mathsf{exp}_{n-1}(O((|Q| + |\mathcal{R}|) \cdot m \cdot |\mathcal{R}|))$.

C The translation for annotated pushdown systems

We present graphically the rewrite rules of the resulting ordered tree transition system. The rules in Figure 1 are the same as in the main text. We hope that the graphical presentation better conveys the intuition behind them.

D The translation for Krivine machines

We present graphically the rewrite rules of the resulting ordered tree transition system. The rules in Figure 2 are the same as in the main text. We hope that the graphical presentation better conveys the intuition behind them.

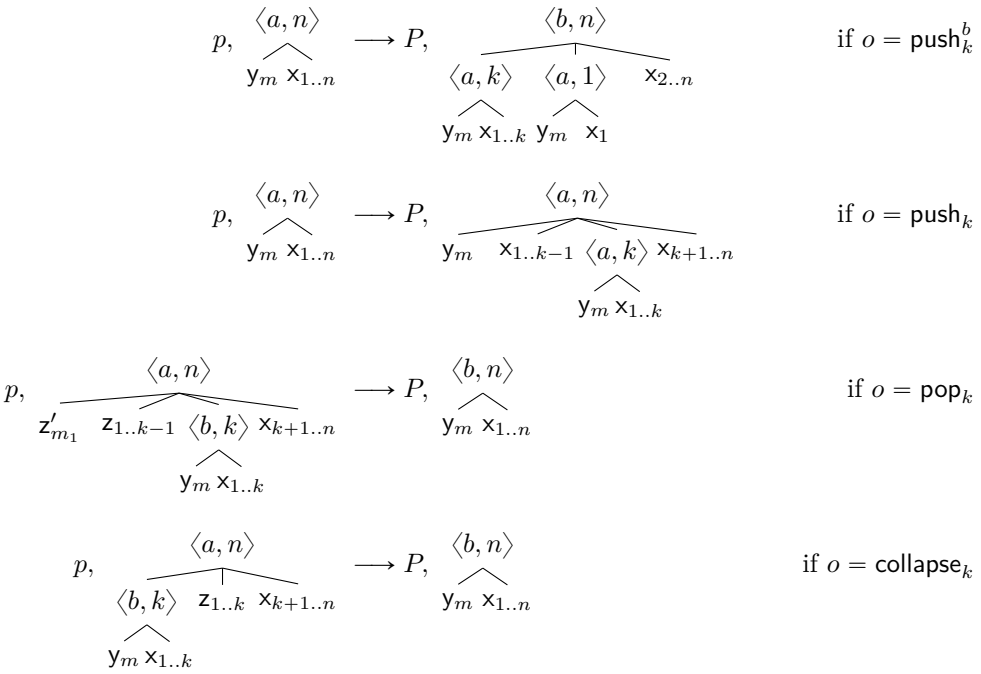


Figure 1 Translation from annotated higher-order pushdown systems to AOTPSs

E Encoding AOTPS into the Krivine machine with states

▸ **Theorem 8.** *Every AOTPS of order n can be encoded in a Krivine machine with states of the same level s.t. every rewriting step of the AOTPS corresponds to a number of reduction steps of the Krivine machine.*

The encoding is performed in four steps.

Step 1: Flattening the AOTPS

In this step we show that the l.h.s. of rewrite rules of AOTPSs can be flattened, in the sense that the only lookahead that the system has is in deep rules, and all other subtrees are just variables. We recall that a rule $p, l \to S, r$ is *flat* if l is either of the form $a(\mathsf{x}_1, \ldots, \mathsf{x}_m)$ (shallow rule) or $a(\mathsf{x}_1, \ldots, \mathsf{x}_{k-1}, b(\mathsf{y}_1, \ldots, \mathsf{y}_{m'}), \mathsf{x}_{k+1}, \ldots, \mathsf{x}_m)$ (deep rule); an AOTPS is *flat* when all its rules are flat.

▸ **Theorem 16.** *Every AOTPS $\mathcal{S}$ can be converted into an equivalent flat AOTPS $\mathcal{S}'$ of exponential size.*

Fix an AOTPS $\mathcal{S} = \langle n, \Sigma, P, \mathcal{R} \rangle$. We create an equivalent flat AOTPS $\mathcal{S}' = \langle n, \Gamma, P, \mathcal{R}' \rangle$ as follows. Let $\mathsf{sub}(\mathcal{R})$ contain all proper (that is, other than the whole tree) subtrees of l for all rules $p, l \to S, r$ in $\mathcal{R}$. The intuition is that the new system will store in each node with k children a tuple consisting of k subsets of $\mathsf{sub}(\mathcal{R})$; the i-th of them will contain these patterns that match to the tree at the i-th child. In this way, when a rule $p, l \to S, r$ is applied, it is sufficient to read in the root whether appropriate subpatterns of l match to subtrees starting in children of the root, instead of testing a longer part of the tree. Thus, as

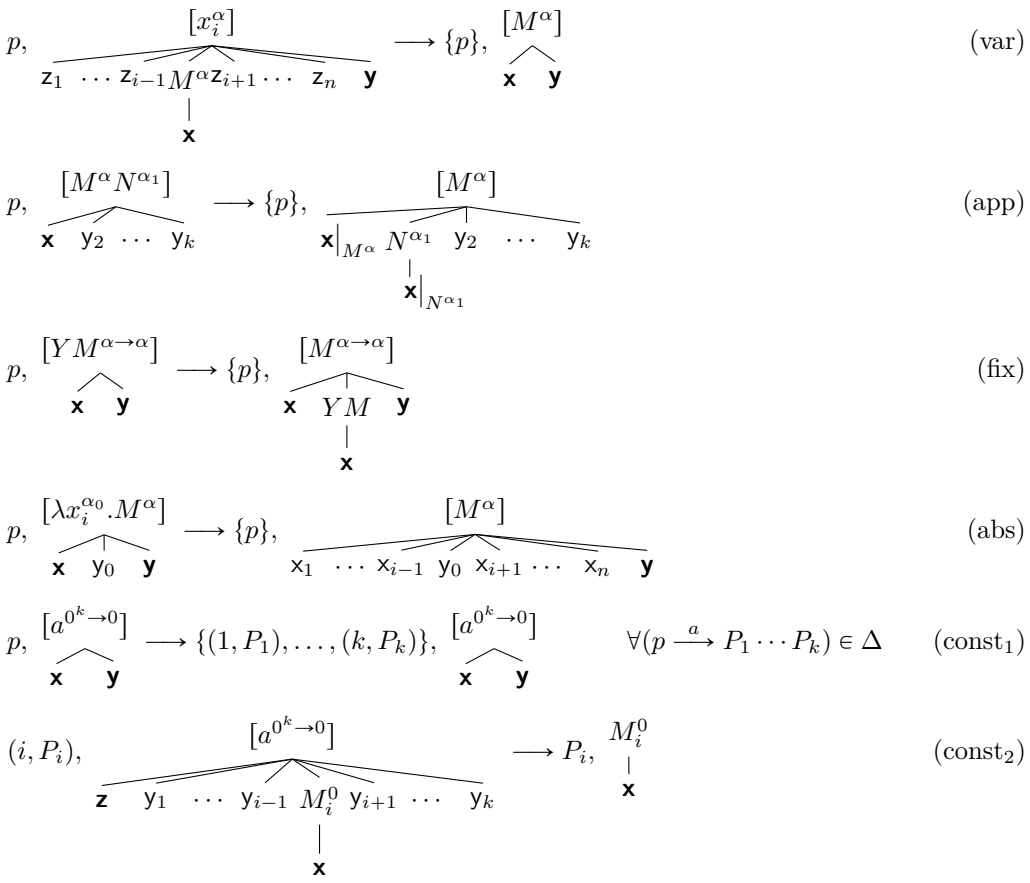


Figure 2 Translation from the Krivine machine to AOTPSs

the new alphabet Γ we take $\bigcup_{a\in\Sigma}\{a\}\times(2^{\mathsf{sub}(\mathcal{R})})^{\mathsf{rank}(a)}$, where the rank and the order of a symbol in Γ is inherited from its Σ coordinate. For a Σ-tree t we obtain the Γ-tree $\mathsf{enc}(t)$ by labelling each node u with the tuple of $\mathsf{rank}(t(u))$ sets where the i-th set contains those trees $l'\in\mathsf{sub}(\mathcal{R})$ that match the subtree rooted at the i-th child of u (that is, for which this subtree equals $l'\sigma$ for some substitution σ).

Consider some rule $p,l\to S,r$. Thanks to the additional labeling of $\mathsf{enc}(t)$, seeing only the label in the root of t we know whether l matches to t. This allows us to replace every r-ground subtree of l by an r-ground variable, and obtain l in one of the two forms allowed in a flat rule. On the other hand, we have to ensure that r creates a tree with the correct labelling. This is possible since our labelling is compositional: the tuple in a node can be computed basing on labels of its children.

More precisely, for each rule $p,l\to S,r$ we create new rules as follows. Let $l=a(u_1,\dots,u_m)$, and, if the rule is deep, let $u_k=b(v_1,\dots,v_{m'})$ be the lookahead subtree of l. For all sets $X_1,\dots,X_m\subseteq\mathsf{sub}(\mathcal{R})$ such that $u_1\in X_1,\dots,u_m\in X_m$, and (in the case of a deep rule) for all sets $Y_1,\dots,Y_{m'}\subseteq\mathsf{sub}(\mathcal{R})$ such that $v_1\in Y_1,\dots,v_{m'}\in Y_{m'}$ there will be one new rule.[8] If $p,l\to S,r$ is shallow, as the new left side we take $(a,X_1,\dots,X_m)(\mathsf{x}_1,\dots,\mathsf{x}_m)$, where we leave $\mathsf{x}_i=u_i$ if u_i is a variable, and we take a fresh variable $\mathsf{x}_i\notin\mathcal{V}(r)$ of order $\mathsf{ord}(u_i)$ otherwise (when u_i is r-ground). If $p,l\to S,r$ is deep, as the new left side we take $(a,X_1,\dots,X_m)(\mathsf{x}_1,\dots,\mathsf{x}_{k-1},(b,Y_1,\dots,Y_{m'})(\mathsf{y}_1,\dots,\mathsf{y}_{m'}),\mathsf{x}_{k+1},\dots,\mathsf{x}_m)$, where again we leave subtrees being variables and we replace other subtrees by fresh variables. The choice of $X_1,\dots,X_m$ and $Y_1,\dots,Y_{m'}$ assigns sets of "matching patterns" to all variables:

$$\mathsf{mp}(\mathsf{x}_i)=X_i,\qquad\qquad \mathsf{mp}(\mathsf{y}_i)=Y_i.$$

Then in a bottom-up manner we can assign such sets to all subtrees of r:

$$\begin{aligned}\mathsf{mp}(c(w_1,\dots,w_j))=\{\mathsf{z}\in\mathsf{sub}(\mathcal{R})\mid\mathsf{ord}(\mathsf{z})=\mathsf{ord}(c)\}\cup\\ \cup\{c(w_1',\dots,w_j')\in\mathsf{sub}(\mathcal{R})\mid w_1'\in\mathsf{mp}(w_1),\dots,w_j'\in\mathsf{mp}(w_k)\}.\end{aligned}$$

The new right side of the rule is $[\![r]\!]$ with $[\![\cdot]\!]$ defined by:

$$\begin{aligned}&[\![w]\!]=(c,\mathsf{mp}(w_1),\dots,\mathsf{mp}(w_j))([\![w_1]\!],\dots,[\![w_j]\!]) && \text{if } w=c(w_1,\dots,w_j),\\ &[\![w]\!]=w && \text{if } w \text{ is a variable.}\end{aligned}$$

We remark that in order to recover correct marking of the right side of a rule it was necessary to mark a node by patterns matching to children of that node instead of patterns matching to the node itself (the latter marking would be insufficient). It is easy to see that each transition of $\langle\mathcal{C}_{\mathcal{S}},\to_{\mathcal{S}}\rangle$ can be faithfully simulated by a transition of $\langle\mathcal{C}_{\mathcal{S}'},\to_{\mathcal{S}'}\rangle$.

Step 2: Eliminating control locations

To ease the presentation in step 3, we now remove control locations from the AOTPS. To allow alternation, we have to extend slightly the definition of an AOTPS. The rules will be now of the form $l\to R$, where R is a set of trees r such that $l\to r$ is a rewrite rule. The resulting alternating transition system $\langle\mathcal{C}_{\mathcal{S}},\to_{\mathcal{S}}\rangle$ has Σ-trees as configurations, and, for every

[8] Some sets in $\mathsf{sub}(\mathcal{R})$ are "inconsistent", as well as some sets $Y_1,\dots,Y_{m'}$ may be "inconsistent" with X_k. New rules using such sets are redundant, but it also does not hurt to add them, as anyway they cannot be applied to any tree of the form $\mathsf{enc}(t)$.

configuration t, and set of configurations U there is a transition $t \to_{\mathcal{S}} U$ if there exists a rule $l \to R$ of $\mathcal{S}$ and a substitution σ s.t. $t = l\sigma$ and $U = \{r\sigma \mid r \in R\}$.

Control locations can be encoded in the root symbol of the pushdown tree. The new alphabet is $\Sigma' = \Sigma \cup (\Sigma \times P)$, where new symbols in $\Sigma \times P$ inherit order and rank from the Σ-component. Let $p, l \to S, r$ be a shallow rule of the original system, with $l = a(\mathsf{x}_1, \ldots, \mathsf{x}_m)$ and $r = c(s_1, \ldots, s_k)$ (the case for a deep rule is analogous). Then, the new system has a rule $l' \to R$ (with no control locations), with $l' = (a, p)(u_1, \ldots, u_m)$ and $R = \{(c, q)(s_1, \ldots, s_k) \mid q \in S\}$. There is a problem when r is just a variable x (necessarily occurring in l). In this case, we use a deep rule by guessing the root symbol c at the (unique) position of x in l (below a). That is, for every shallow rule $p, l \to S, \mathsf{x}_k$ of the original system, and for every symbol c of rank h, we introduce a deep rule $l' \to R$, where $l' = (a, p)(\mathsf{x}_1, \ldots, \mathsf{x}_{k-1}, c(\mathsf{y}_1, \ldots, \mathsf{y}_h), \mathsf{x}_{k+1}, \ldots, \mathsf{x}_m)$ and $R = \{(c, q)(\mathsf{y}_1, \ldots, \mathsf{y}_h) \mid q \in S\}$. We can assume that in the original system there are no deep rules of the form $p, l \to S, \mathsf{x}$, as those can be broken down into two rules, a deep one where the r.h.s. is not a variable, and a shallow one where the r.h.s. is a variable. Thus, we do not have consider any other case when removing control locations.

Notice that while starting from a flat AOTPS, the obtained AOTPS without control locations is also flat.

Step 3: From AOTPSs to higher-order recursion schemes

After the first two steps we have a flat AOTPS without control locations (where rules are of the form $l \to R$ with R a set of trees). Our goal is to translate an AOTPS of this form into a Krivine machine, thus proving Theorem 8. In order to obtain a more natural translation, we use recursion schemes, a model quite similar to the Krivine machine.

We first define alternating higher-order recursion schemes with states. We use types as defined in Sec. 4.3. However, instead of λY-terms, we use applicative terms, where moreover we can use typed nonterminals from some set $\mathcal{N}$. An *applicative term* is either (i) a constant $a^{0^{\mathsf{rank}(a)} \to 0} \in \Gamma$, (ii) a variable $x^\alpha \in \mathcal{V}$, (iii) a nonterminal $A^\alpha \in \mathcal{N}$, (iv) an application $(M^{\alpha \to \beta}\, N^\alpha)^\beta$. An *alternating recursion scheme with states* of level $n \in \mathbb{N}_{>0}$ is a tuple $\mathcal{G} = \langle n, \Gamma, Q, \mathcal{N}, \mathcal{R}, \Delta \rangle$, where $\langle \Gamma, Q, \Delta \rangle$ is an alternating tree automaton, $\mathcal{N}$ is a finite set of nonterminals of level at most n, and $\mathcal{R}$ is a function assigning to each nonterminal A in $\mathcal{N}$ of type $\alpha_1 \to \cdots \to \alpha_k \to 0$ a rule of the form $A\, x_1^{\alpha_1} \ldots x_k^{\alpha_k} \to M$, where M is an applicative term of type 0 with free variables in $\{x_1^{\alpha_1}, \ldots, x_k^{\alpha_k}\}$, constants from Γ, and nonterminals from $\mathcal{N}$. A recursion scheme $\mathcal{G}$ induces an alternating transition system $\langle \mathcal{C}_\mathcal{G}, \to_\mathcal{G} \rangle$, where in a configuration $(p, M) \in \mathcal{C}_\mathcal{G}$ we have $p \in Q$ and M is a closed applicative term of type 0 using constants from Γ and nonterminals from $\mathcal{N}$. We have two kinds of transitions. First, for each rule $A\, x_1^{\alpha_1} \ldots x_k^{\alpha_k} \to M$ in $\mathcal{R}$ we have a transition $(p, A\, M_1 \ldots M_k) \to_\mathcal{G} \{(p, M[M_1/x_1, \ldots, M_k/x_k])\}$. Second, for every $(p \xrightarrow{a} P_1 \ldots P_k) \in \Delta$ we have a transition $(p, a\, M_1 \ldots M_k) \to_\mathcal{G} (P_1 \times \{M_1\}) \cup \cdots \cup (P_k \times \{M_k\})$.

Fix a flat AOTPS $\mathcal{S} = \langle n, \Sigma, \mathcal{R} \rangle$ without control locations. An *extended letter* is a pair (a, o) where $a \in \Sigma$ and $o \colon \{1, \ldots, \mathsf{rank}(a)\} \to \{1, \ldots, n\}$. The meaning is that the letter is a and its children have orders $o(1), \ldots, o(\mathsf{rank}(a))$. For each extended letter (a, o) we will have a corresponding nonterminal $A_{a,o}$. A first approximation of the encoding is that a tree $a(u_1, \ldots, u_k)$ will be represented as $A_{a,o}$ to which encodings of $u_1, \ldots, u_k$ are applied as arguments. Then the rule for the nonterminal $A_{a,o}$ can simulate all shallow rules of our system $\mathcal{S}$, constructing any term having $u_1, \ldots, u_k$ as subterms. Notice that rules of $\mathcal{S}$ are flat, so we need not to look inside $u_1, \ldots, u_k$; however we need to know their orders—that is why we assign nonterminals to extended letters not just to letters.

Nevertheless, there are also deep rules. In order to handle them, each tree $a(u_1, \dots, u_k)$ will be represented in multiple ways. One representation will handle shallow rules. Moreover, for each extended letter (b, o') and $\hat{\imath}$ we will have a nonterminal $A^{b,o',\hat{\imath}}_{a,o}$, and $a(u_1, \dots, u_k)$ will be represented as $A^{b,o',\hat{\imath}}_{a,o}$ with applied representations of $u_1, \dots, u_k$. This nonterminal will be still waiting for subtrees of a potential parent of our a, having label b; when they will be applied, the nonterminal will simulate deep rules of $\mathcal{S}$ having $b(\dots, a(\dots), \dots)$ on the left side, where the a is on the $\hat{\imath}$-th position. As we have multiple encodings of a tree, we need to keep them in parallel. Thus $A_{a,o}$ (and similarly $A^{b,o',\hat{\imath}}_{a,o}$) instead of taking one argument for each subtree, takes multiple arguments for each subtree, one for each encoding of the subtree.

Let $\overline{\Sigma}_k$ be the set of all triples $(b, o', \hat{\imath})$, where (b, o') is an extended letter for $b \in \Sigma$, and $o'(\hat{\imath}) = k$. Let us fix some (arbitrary) total order on elements of $\overline{\Sigma}_k$, that will be used to order arguments of our terms. Using the product notation $\alpha \times \beta \to \gamma$ for the type $\alpha \to \beta \to \gamma$, we define types

$$\Psi_k = 0 \times \prod_{(a,o,\hat{\imath}) \in \overline{\Sigma}_k} \alpha_{a,o,>k},$$

where the $(a, o, \hat{\imath})$ are ordered according to our fixed order on $\overline{\Sigma}_k$. We have used here the types

$$\alpha_{a,o,>k} = \Psi_{o(i_1)} \to \dots \to \Psi_{o(i_m)} \to 0,$$

where $i_1 < \dots < i_m$ is the list of all $i \in \{1, \dots, \mathsf{rank}(a)\}$ for which $o(i) > k$. In particular we have $\alpha_{a,o,>n} = 0$. Notice that $\alpha_{a,o,>k}$ and types in Ψ_k are of level at most $n - k$. We are now ready to define the type of our nonterminals: $A_{a,o}$ has type

$$\Psi_{o(1)} \to \dots \to \Psi_{o(\mathsf{rank}(a))} \to 0$$

and $A^{b,o',\hat{\imath}}_{a,o}$ has type

$$\Psi_{o(1)} \to \dots \to \Psi_{o(\mathsf{rank}(a))} \to \alpha_{b,o',>\mathsf{ord}(a)}.$$

Recall that the resulting higher-order recursion scheme has to use constants and an alternating tree automaton to simulate alternation. We will have two kinds of constants in Γ: $\vee_i$ of type $0^i \to 0$, and $\wedge_i$ of type $0^i \to 0$, defined for appropriate numbers $i \in \mathbb{N}$ (we need only finitely many of them). The constant $\vee_i$ simulates nondeterministic choice, and $\wedge_i$ simulates universal choice. The tree automaton $\langle \Gamma, Q, \Delta \rangle$ is defined in the natural way: Q consists of a single state q, and we have transitions

$$q \xrightarrow{\vee_i} \varnothing \dots \varnothing Q \varnothing \dots \varnothing, \qquad q \xrightarrow{\wedge_i} Q \dots Q.$$

In particular there is no transition reading $\vee_0$, and we have the transition $q \xrightarrow{\wedge_0} \varepsilon$.

Next, we define our encoding of trees into applicative terms. In the definition below we use tuples just as a shorthand: $M\,(N_1, \dots, N_k)$ is intended to mean $M\,N_1\,\dots\,N_k$. A tree is encoded as a tuple of applicative terms:

$$\mathsf{Enc}(u) = (\mathsf{enc}(u), \mathsf{enc}^{b_1,o_1,\hat{\imath}_1}(u), \dots, \mathsf{enc}^{b_k,o_k,\hat{\imath}_k}(u))$$

where $(b_1, o_1, \hat{\imath}_1) < \dots < (b_k, o_k, \hat{\imath}_k)$ are all the elements of $\overline{\Sigma}_{\mathsf{ord}(u)}$ ordered according to the fixed order on $\overline{\Sigma}_{\mathsf{ord}(u)}$. We have here one basic encoding:

$$\mathsf{enc}(a(u_1, \dots, u_m)) = A_{a,o}\,\mathsf{Enc}(u_1)\,\dots\,\mathsf{Enc}(u_k) \qquad \text{where } \forall i \cdot o(i) = \mathsf{ord}(u_i).$$

Additionally for each $(b_j, o_j, \hat{\imath}_j) \in \overline{\Sigma}_{\mathsf{ord}(a)}$ we have

$$\mathsf{enc}^{b_j,o_j,\hat{\imath}_j}(a(u_1,\ldots,u_m)) = A_{a,o}^{b_j,o_j,\hat{\imath}_j}\,\mathsf{Enc}(u_1)\ \ldots\ \mathsf{Enc}(u_k) \qquad \text{where } \forall i \cdot o(i) = \mathsf{ord}(u_i).$$

It remains to define rules for the nonterminals from the rules of $\mathcal{S}$. Let us use the convention on variable naming that every left side of a rule in $\mathcal{R}$ is of the form $a(\mathsf{x}_1,\ldots,\mathsf{x}_m)$ or $b(\mathsf{y}_1,\ldots,\mathsf{y}_{\hat{\imath}-1},a(\mathsf{x}_1,\ldots,\mathsf{x}_m),\mathsf{y}_{\hat{\imath}+1},\ldots,\mathsf{x}_k)$. Recall that $\mathcal{S}$ is flat, so every left side is of such form. A right side $R = \{r_1,\ldots,r_l\}$ of a rule is encoded as

$$\mathsf{enc}(R) = \wedge_l\,\mathsf{enc}(r_1)\ \ldots\ \mathsf{enc}(r_l),$$

where $\mathsf{enc}(r_i)$ is defined as for normal trees, with the encoding of variables given by:

$$\mathsf{enc}(\mathsf{x}_i) = x_i, \qquad \mathsf{enc}^{a,o,\hat{\imath}}(\mathsf{x}_i) = x_i^{a,o,\hat{\imath}},$$
$$\mathsf{enc}(\mathsf{y}_i) = y_i, \qquad \mathsf{enc}^{a,o,\hat{\imath}}(\mathsf{y}_i) = y_i^{a,o,\hat{\imath}}.$$

Here, x_i is a variable of the AOTPS, while x_i is a variable in an applicative term of the recursion scheme, and similarly for the other variables. In order to define a rule for a nonterminal $A_{a,o}$ we look at all rules with $a(\mathsf{x}_1,\ldots,\mathsf{x}_m)$ on the left side with $\mathsf{ord}(\mathsf{x}_i) = o(i)$ for all i. Let $R_1,\ldots,R_l$ be the right sides of these rules. Then we take

$$A_{a,o}\,\mathsf{Enc}(\mathsf{x}_1)\ \ldots\ \mathsf{Enc}(\mathsf{x}_m) \to \vee_{l+m}\,\mathsf{enc}(R_1)\ \ldots\ \mathsf{enc}(R_l)\,\mathsf{deep}_1\ \ldots\ \mathsf{deep}_m$$

with

$$\mathsf{deep}_{\hat{\imath}} = x_{\hat{\imath}}^{a,o,\hat{\imath}}\,\mathsf{Enc}(\mathsf{x}_{i_1})\ \ldots\ \mathsf{Enc}(\mathsf{x}_{i_d}),$$

where $i_1 < \cdots < i_d$ are all the $i \in \{1,\ldots,m\}$ for which $o(i) > o(\hat{\imath})$. In order to define a rule for a nonterminal $A_{a,o}^{b,o',\hat{\imath}}$ we look at all rules having on the left side $b(\mathsf{y}_1,\ldots,\mathsf{y}_{\hat{\imath}-1},a(\mathsf{x}_1,\ldots,\mathsf{x}_m),\mathsf{y}_{\hat{\imath}+1},\ldots,\mathsf{x}_k)$ with $\mathsf{ord}(\mathsf{y}_i) = o'(i)$ and $\mathsf{ord}(\mathsf{x}_i) = o(i)$ for all i, and $\mathsf{ord}(a) = o'(\hat{\imath})$ (in particular for nonterminals $A_{a,o}^{b,o',\hat{\imath}}$ with $\mathsf{ord}(a) \neq o'(\hat{\imath})$ there are no such rules). Let $R_1,\ldots,R_l$ be the right sides of these rules (when $\mathsf{ord}(a) \neq o'(\hat{\imath})$, we always have $l = 0$). Then we take

$$A_{a,o}^{b,o',\hat{\imath}}\,\mathsf{Enc}(\mathsf{x}_1)\ \ldots\ \mathsf{Enc}(\mathsf{x}_m)\,\mathsf{Enc}(\mathsf{y}_{i_1})\ \ldots\ \mathsf{Enc}(\mathsf{y}_{i_d}) \to \vee_l\,\mathsf{enc}(R_1)\ \ldots\ \mathsf{enc}(R_l),$$

where $i_1 < \cdots < i_d$ are all the $i \in \{1,\ldots,k\}$ for which $o'(i) > \mathsf{ord}(a)$.

Let us briefly see the correspondence between steps of $\mathcal{S}$ and steps of $\mathcal{G}$. Consider a tree $u = a(u_1,\ldots,u_m)$. Its encoding $\mathsf{enc}(u)$ is $A_{a,o}\,\mathsf{Enc}(u_1)\ \ldots\ \mathsf{Enc}(u_m)$ where $o(i) = \mathsf{ord}(u_i)$ for each i. The first step of $\mathcal{G}$ performed from $\mathsf{enc}(u)$ results in the right side of the rule for $A_{a,o}$ where we substitute $\mathsf{enc}(u_i)$ for x_i and $\mathsf{enc}^{b,o',\hat{\imath}}(u_i)$ for $x_i^{b,o',\hat{\imath}}$ for each $i, b, o', \hat{\imath}$ (denote this substitution as $[\varphi]$). The resulting tree starts with some $\vee_j$, so then $\mathcal{G}$ nondeterministically chooses one of its subtrees. There is a subtree $\mathsf{enc}(R)[\varphi]$ for each shallow rule $a(\mathsf{x}_1,\ldots,\mathsf{x}_m) \to R$ of $\mathcal{S}$ where $\mathsf{ord}(\mathsf{x}_i) = o(i) = \mathsf{ord}(u_i)$ for each i. These are exactly all shallow rules that can be applied to u. By applying this rule to u we obtain the set $\{r\sigma \mid r \in R\}$ where σ maps x_i to u_i for each i. Notice that $\mathsf{enc}(R)[\varphi]$ starts with $\wedge_{|R|}$ and has as children $\mathsf{enc}(r)[\varphi]$ for each $r \in R$. Thus $\mathcal{G}$ in the next step will transit into the set $\{\mathsf{enc}(r)[\varphi] \mid r \in R\}$. Finally, we see that $\mathsf{enc}(r\sigma) = \mathsf{enc}(r)[\varphi]$.

Another possibility for $\mathcal{G}$ in the second step is to transit for some $\hat{\imath} \in \{1,\ldots,m\}$ to

$$\begin{aligned}\mathsf{deep}_{\hat{\imath}}[\varphi] = (x_{\hat{\imath}}^{a,o,\hat{\imath}}\,\mathsf{Enc}(\mathsf{x}_{i_1})\ \ldots\ \mathsf{Enc}(\mathsf{x}_{i_d}))[\varphi] &= \mathsf{enc}^{a,o,\hat{\imath}}(u_{\hat{\imath}})\,\mathsf{Enc}(u_{i_1})\ \ldots\ \mathsf{Enc}(u_{i_d}) = \\ &= A_{b,o'}^{a,o,\hat{\imath}}\ \mathsf{Enc}(v_1)\ \ldots\ \mathsf{Enc}(v_k)\,\mathsf{Enc}(u_{i_1})\ \ldots\ \mathsf{Enc}(u_{i_d}),\end{aligned}$$

where $u_i = b(v_1, \dots, v_k)$, and $o'(i) = \mathsf{ord}(v_i)$ for each i, and $i_1 < \cdots < i_d$ are all $i \in \{1, \dots, m\}$ for which $o(i) > o(\hat{\imath})$. The next three steps of $\mathcal{G}$ simulate exactly all deep rules of $\mathcal{S}$ having on the left side $a(\mathsf{y}_1, \dots, \mathsf{y}_{i-1}, b(\mathsf{x}_1, \dots, \mathsf{x}_k), \mathsf{y}_{i+1}, \dots, \mathsf{y}_m)$ with $\mathsf{ord}(\mathsf{y}_i) = o(i)$ and $\mathsf{ord}(\mathsf{x}_i) = o'(i)$ for all i (in the same way as it was for shallow rules and for the nonterminal $A_{a,o}$). It is important that the right sides of such rules use only those variables y_i for which $\mathsf{ord}(\mathsf{y}_i) = o(i) > o(\hat{\imath}) = \mathsf{ord}(b)$. We conclude that $\langle \mathcal{C}_{\mathcal{S}}, \to_{\mathcal{S}} \rangle$ and $\langle \mathcal{C}_{\mathcal{G}}, \to_{\mathcal{G}} \rangle$ faithfully simulate each other.

Step 4: From higher-order recursion schemes to the Krivine machine

It is well-known how to translate from one formalism to the other; cf. [25].

F Ordered annotated multi-pushdown systems

We encode ordered annotated multi-pushdown systems [15] into AOTPSs. Formally, an *alternating ordered annotated multi-pushdown system* is a tuple $\mathcal{R} = \langle m, n, \Gamma, Q, \Delta \rangle$, where $m \in \mathbb{N}_{>0}$ is the number of higher-order pushdowns, $n \in \mathbb{N}_{>0}$ is the order of each of the m higher-order pushdowns, Γ is a finite pushdown alphabet containing a distinguished initial symbol e, Q is a finite set of control locations, and $\Delta \subseteq Q \times \{1, \dots, m\} \times \Gamma \times O_n \times 2^Q$ is a set of rules, where $O_n = \bigcup_{k=1}^{n} \{\mathsf{push}_k^b, \mathsf{push}_k, \mathsf{pop}_k, \mathsf{collapse}_k \mid b \in \Gamma\}$. Intuitively, a rule (p, l, a, o, P) can be applied when the control location is p and the topmost symbol on the l-th stack is a, and it applies the stack operation specified by o to this stack. Pop and collapse operations are called *consuming*. An alternating ordered annotated multi-pushdown system $\mathcal{R}$ induces an alternating transition system $\langle \mathcal{C}_{\mathcal{R}}, \to_{\mathcal{R}} \rangle$, where $\mathcal{C}_{\mathcal{R}} = Q \times \Gamma_n^m$, and $(p, w_1, \dots, w_m) \to_{\mathcal{R}} P \times \{(w_1', \dots, w_m')\}$ if, and only if, there exists a rule $(p, l, a, o, P) \in \Delta$ s.t.

1) $w_l = \langle a^{u_l}, \cdots \rangle$,
2) if o is consuming, then $w_1' = \cdots = w_{l-1}' = \langle e^{\langle\rangle}, \langle\rangle, \dots, \langle\rangle \rangle$,
3) if o is not consuming, then $w_1' = w_1, \dots, w_{l-1}' = w_{l-1}$,
4) $w_l' = o(w_l)$, and
5) $w_{l+1}' = w_{l+1}, \dots, w_m' = w_m$.

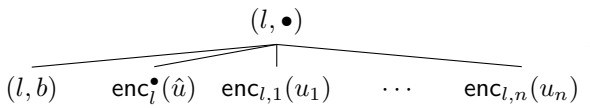
For $c \in \mathcal{C}_{\mathcal{R}}$ and $T \subseteq Q$, the *(control-state) reachability problem* for $\mathcal{R}$ asks whether $c \in \mathrm{Pre}^*(T \times \Gamma_n^m)$.

Encoding. Let Σ be an ordered alphabet containing, for every pushdown index $l \in \{1, \dots, m\}$ and order $k \in \{1, \dots, n\}$, 1) an end-of-stack symbol $(l, k, \bot)$ of order $(l-1) \cdot n + k$ and rank 0, 2) a symbol (l, k) of order $(l-1) \cdot n + k$ and rank $k+2$, 3) a symbol $(l, \bullet)$ of order $(l-1) \cdot n + 1$ and rank $n+2$. Moreover, Σ contains a symbol $\bullet$ of order 1 and rank $m \cdot (n+1)$, and, for every pushdown index $l \in \{1, \dots, m\}$ and every $a \in \Gamma$, a symbol (l, a) of order $(l-1) \cdot n + 1$ and rank 0. Thus Σ has order mn. Notice that the size of Σ is $|\Sigma| = O(m \cdot (n + |\Gamma|))$. Fix a pushdown index l. An empty order-k pushdown is encoded as the tree $\mathsf{enc}_{l,k}(\langle\rangle) = (l, k, \bot)$. A nonempty order-$k$ pushdown $\langle a^{\hat{u}}, u_1, \dots, u_k \rangle$ is encoded as the tree

$$\mathsf{enc}_{l,k}(\langle a^{\hat{u}}, u_1, \dots, u_k \rangle) = \begin{array}{ccccccc} & & & (l,k) & & & \\ & \diagup & & \diagup & & \diagdown & \\ (l,a) & & \mathsf{enc}_l^{\bullet}(\hat{u}) & & \mathsf{enc}_{l,1}(u_1) & \cdots & \mathsf{enc}_{l,k}(u_k) \end{array},$$

and

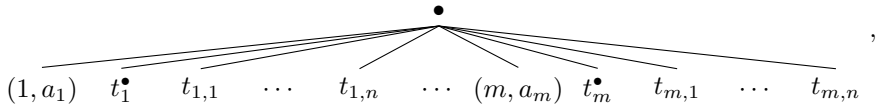
$$\mathsf{enc}^{\bullet}_l(\langle b^{\hat{u}}, u_1, \ldots, u_n\rangle) = \quad .$$



To simplify the encoding, we assume w.l.o.g. that collapse links are always of order n (collapse to a lower order is still allowed though). An m-tuple of nonempty order-n pushdowns

$$w = (\langle a_1^{\hat{u}_1}, u_{1,1}, \ldots, u_{1,n}\rangle, \ldots, \langle a_m^{\hat{u}_m}, u_{m,1}, \ldots, u_{m,n}\rangle)$$

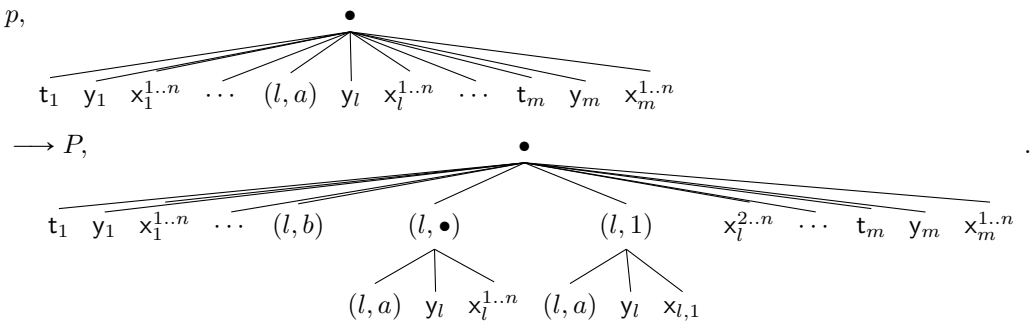
is encoded as the following tree $\mathsf{enc}(w)$



,

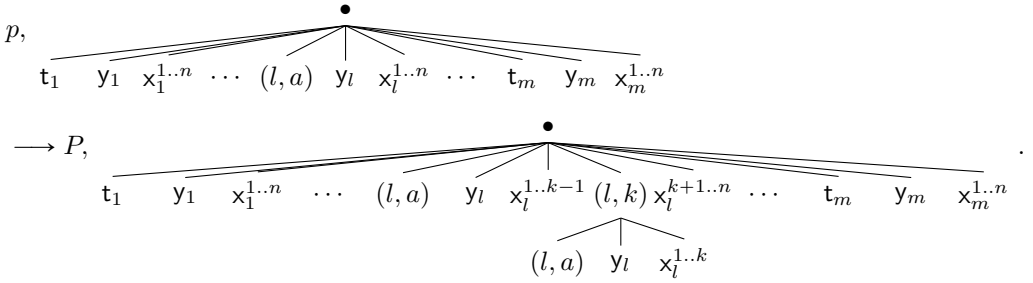
where $t^{\bullet}_l = \mathsf{enc}^{\bullet}_l(\hat{u}_l)$ for every $l = 1, \ldots, m$, and $t_{l,k} = \mathsf{enc}_{l,k}(u_{l,k})$ for every $l = 1, \ldots, m$ and $k = 1, \ldots, n$. We can observe here two differences when comparing to the encoding of annotated higher-order pushdown systems. First, we do not put the topmost stack symbol in the root, but in an additional child of the form (l, a) just below the root. This allows to avoid an alphabet of exponential size containing all m-tuples of letters from Γ (the rest of the stack is encoded in an analogous way, but this is only for uniformity). Second, at the beginning of each collapse link we use a special letter $(l, \bullet)$ instead of (l, k). This letter has a fixed order, and thanks to that we can use a variable of this fixed order to match the subtree encoding a collapse link. Recall that for annotated higher-order pushdown systems we have created separate rules for each possible order of the collapse links, but here such a solution would result in an exponential blowup since we have m collapse links of independent orders.

Let $\langle m, n, \Gamma, Q, \Delta\rangle$ be an ordered annotated multi-pushdown system. We define an equivalent AOTPS $\mathcal{S} = \langle mn, \Sigma, Q, \mathcal{R}_{\mathcal{S}}\rangle$ of order mn, where Σ and Q are as defined above, and $\mathcal{R}_{\mathcal{S}}$ contains a rule for each rule in Δ. We use the convention that variables $\mathsf{x}_{l,k}$, $\mathsf{z}_{l,k}$ have order $(l-1)\cdot n + k$, and variables y_l, y'_l, t_l, are of order $(l-1)\cdot n + 1$. We write $\mathsf{x}_l^{i..j}$ (with $i \leqslant j$) for the tuple of variables $(\mathsf{x}_{l,i}, \ldots, \mathsf{x}_{l,j})$. If $(p, l, a, \mathsf{push}^b_k, P) \in \Delta$, then there is the following shallow rule in $\mathcal{R}_{\mathcal{S}}$:

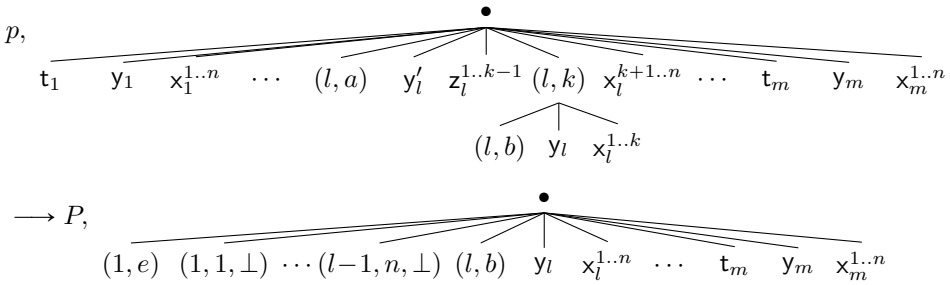


.

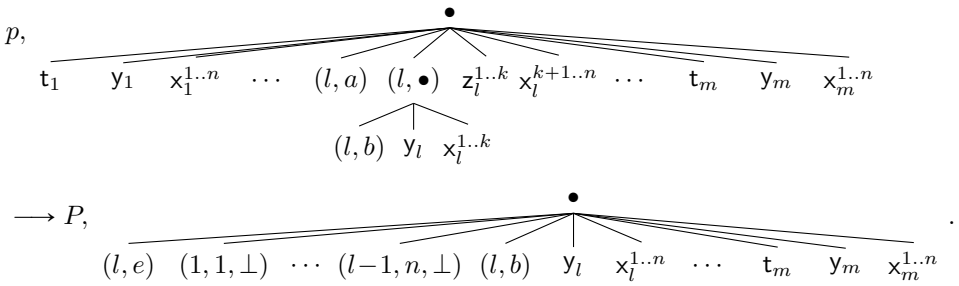
If $(p, l, a, \mathsf{push}_k, P) \in \Delta$, then there is the following shallow rule in $\mathcal{R}_{\mathcal{S}}$:



If $(p, l, a, \mathsf{pop}_k, P) \in \Delta$, then there is the following deep rule in $\mathcal{R}_{\mathcal{S}}$:



This deep rule satisfies the ordering condition since (l, k) has order $(l-1) \cdot n + k$, and all variables $\mathsf{x}_l^{k+1..n}$ that appear also on the r.h.s. have order strictly higher than (l, k). Notice that y'_l has relatively low order, but it does not appear on the r.h.s. Finally, if $(p, l, a, \mathsf{collapse}_k, P) \in \Delta$, then there is the following deep rule in $\mathcal{R}_{\mathcal{S}}$:



The deep rule above satisfies the ordering condition since $(l, \bullet)$ has order $(l-1) \cdot n + 1$, and all variables $\mathsf{x}_l^{k+1..n}$, t_{l+1}, y_{l+1}, $\mathsf{x}_{l+1}^{1..n}$, …, t_m, y_m, $\mathsf{x}_m^{1..n}$ that appear also on the r.h.s. have strictly higher order.

▸ **Lemma 17** (Simulation). *We have that* $(p, w) \to_{\mathcal{R}}^* P \times \{w'\}$ *if, and only if,* $(p, \mathsf{enc}(w)) \to_{\mathcal{S}}^* P \times \{\mathsf{enc}(w')\}$. *Thus, the (control-state) reachability problem for* $\mathcal{R}$ *is equivalent to the reachability problem for* $\mathcal{S}$.