# Ordered Tree-Pushdown Systems

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

**1,2 University of Warsaw**
**Warsaw, Poland**
**3,4 CNRS, Université de Bordeaux, INRIA**
**Bordeaux, 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, 14] (cf. also the recent survey [10]). Pushdown systems have been generalized in several directions. One of them are *tree-pushdown systems* [15], 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.

* 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.

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 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 [23, 20]. *Collapsible pushdown systems* [21, 17] 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 [19]) provide a simplification of collapsible pushdown systems by replacing collapse links with arbitrary pushdown annotations[1]. The *Krivine machine* [24] is a related model which evaluates terms in simply-typed $\lambda Y$-calculus. Reachability in all these models is $(\mathsf{n}-1)$-EXPTIMEc [7, 24] (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* [16] 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 [13] 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 [16] 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 $\lambda Y$-calculus, so our formalism

[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.

can be used to obtain a saturation method for schemes.

Ordered tree-pushdown systems proposed in the present paper unify these approaches. 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 [17, 26], 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 [16] 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 [12].

**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 $\mathcal{S} = \langle \mathcal{C}, \to \rangle$, where $\mathcal{C}$ is the set of configurations and $\to \subseteq \mathcal{C} \times 2^{\mathcal{C}}$ is the alternating transition relation. For two sets of configurations $A, B \subseteq \mathcal{C}$ we define $A \to_1 B$ iff, for every $c \in A$, either $c \in B$, or there exists $C \subseteq B$ s.t. $c \to C$, and we denote by $\to_1^*$ its reflexive and transitive closure. The set of *predecessors* of a set of configurations $C \subseteq \mathcal{C}$ is $\mathrm{Pre}^*(C) = \{c \mid \{c\} \to_1^* C\}$.

**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 $(\Sigma, \mathsf{rank})$ of a set of symbols $\Sigma$ together with a ranking function $\mathsf{rank} : \Sigma \to \mathbb{N}$. A $\Sigma$-*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 $\Sigma$-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 $\Sigma$-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 $\Sigma$ 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 $\sigma$, $t\sigma$ is the $(\Sigma \cup \mathcal{V})$-tree obtained by replacing each variable $\mathsf{x}$ in $t$ in the domain of $\sigma$ with $\sigma(\mathsf{x})$. A *rewrite rule* over $\Sigma$ 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 $\Sigma$ 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 $\Sigma$-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 $\Sigma$ 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 $\Sigma$ of one of the two forms:

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

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

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

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

[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.

$u_i, v_j$ 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 $c$ 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 $\sigma$ 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 3.1** (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 $\Delta'$ by adding transitions to $\Delta$, 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 $\Sigma$-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 $\Delta_0$ 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:

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

[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.

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 $Q' := Q'_1$.

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, $\Delta'$ 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 $\mathsf{x}$ 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 3.2** (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 of the technical report [11]. 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 3.1. 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 $\Sigma$. Using the notation from the previous subsection, we have that $|Q'_{n+1}| \leqslant |Q| + |\mathcal{R}|$, $|Q'_n| \leqslant |Q'_{n+1}| + |\mathcal{R}|$, and for every $k \in \{1, \ldots, n-1\}$, $|Q'_k| \leqslant |Q'_{k+1}| + |\mathcal{R}| \cdot 2^{(m-1)\cdot|Q'_{k+1}|} \leqslant O\left(|\mathcal{R}| \cdot 2^{(m-1)\cdot|Q'_{k+1}|}\right)$, 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 3.3.** *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 of the technical report [11].

▸ **Theorem 3.4.** *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 3.5.** *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 3.6.** *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 3.7.** *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 of the technical report [11].

▸ **Theorem 3.8.** *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 [25], 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* [15] can be seen as *shallow* AOTPSs. By Theorem 3.6, reachability is in EXPTIME for both classes, and, by Theorem 3.7, 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 $1, \ldots, k-1$ 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), $\Gamma$ 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 $(p, w_1, \ldots, w_n) \rightarrow_\mathcal{O} P \times \{(w_1, \ldots, a \cdot w_k, \ldots, w_n)\}$, and for every $(p, \mathsf{pop}_k(a), P) \in \Delta$ there exists a transition $(p, w_1, \ldots, a \cdot w_k, \ldots, w_n) \rightarrow_\mathcal{O} P \times \{(\varepsilon, \ldots, \varepsilon, w_k, \cdots, w_n)\}$. 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 $\Gamma$, let $\Gamma_\perp = \Gamma \cup \{\perp\}$, and let $\Sigma = (\Gamma_\perp \times \{1, \ldots, n\}) \cup \{\bullet\}$ 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 $w_1, \ldots, w_n$, where each $w_j = a_{j,1} \ldots a_{j,n_j}$ is encoded as the tree $\mathsf{enc}(w_1, \ldots, w_n) := \bullet(a^1_{1,1}(a^1_{1,2}(\ldots \perp^1)), \ldots, a^n_{n,1}(a^n_{n,2}(\ldots \perp^n)))$. 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 $\Sigma$ defined as above, and set of rules $\mathcal{R}$ defined as follows (we use the convention that variable $\mathsf{x}_k$ has order $k$): For every push rule $(p, \mathsf{push}_k(a), P) \in \Delta$, we have a rule $(p, \bullet(\mathsf{x}_1, \ldots, \mathsf{x}_n) \rightarrow P, \bullet(\mathsf{x}_1, \ldots, a^k(\mathsf{x}_k), \ldots, \mathsf{x}_n)) \in \mathcal{R}$, and for every pop rule $(p, \mathsf{pop}_k(a), P) \in \Delta$, we have $(p, \bullet(\mathsf{x}_1, \ldots, a^k(\mathsf{x}_k), \ldots, \mathsf{x}_n) \rightarrow P, \bullet(\perp^1, \ldots, \perp^{k-1}, \mathsf{x}_k, \mathsf{x}_{k+1}, \ldots, \mathsf{x}_n)) \in \mathcal{R}$. Both kinds of rules above are linear, and the latter one satisfies the ordering condition since lower-order variables $\mathsf{x}_1, \ldots, \mathsf{x}_{k-1}$ are discarded. It is easy to see that $(p, w_1, \ldots, w_n) \rightarrow^*_\mathcal{O} P \times \{(w'_1, \ldots, w'_n)\}$ if, and only if, $(p, \mathsf{enc}(w_1, \ldots, w_n)) \rightarrow^*_\mathcal{S} P \times \{\mathsf{enc}(w'_1, \ldots, w'_n)\}$. Thus, the encoding preserves reachability properties. By Theorem 3.3, 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 3.5 we recover the optimal 2-EXPTIMEc complexity proved by [3] (cf. also [2]).

▸ **Theorem 4.1** ([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 $\Gamma$ 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 $\Gamma_k$, the set of *annotated higher-order pushdowns (stacks) of order* $k$, simultaneously for all $k \in \{1, \ldots, n\}$, as the least set containing the empty pushdown $\langle\rangle$, and, whenever $u_1 \in \Gamma_1, \ldots, u_k \in \Gamma_k$, $v_j \in \Gamma_j$ for some $j \in \{1, \ldots, n\}$, then $\langle a^{v_j}, u_1, \ldots, u_k\rangle \in \Gamma_k$. 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 $\mathsf{push}_k^b$ pushes a symbol $b \in \Gamma$ on the top of the topmost order-1 stack and annotates it with the topmost order-$k$ stack, $\mathsf{push}_k$ duplicates the topmost order-$(k-1)$ stack, $\mathsf{pop}_k$ 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:

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

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 $\mathsf{push}^b$ and $\mathsf{pop}_k$ on stacks without annotations (but not $\mathsf{collapse}_k$, or $\mathsf{push}_k^b$). An *alternating order-n annotated pushdown system* is a tuple $\mathcal{P} = \langle n, \Gamma, Q, \Delta\rangle$, where $\Gamma$ 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 $\langle \mathcal{C}_\mathcal{P}, \rightarrow_\mathcal{P}\rangle$, where $\mathcal{C}_\mathcal{P} = Q \times \Gamma_n$, and the transition relation is defined as $(p, w) \rightarrow_\mathcal{P} P \times \{w'\}$ 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 $\Sigma$ be the ordered alphabet containing, for each $k \in \{1, \ldots, n\}$, an end-of-stack symbol $\perp^k \in \Sigma$ of rank 0 and order $k$. Moreover, for each $a \in \Gamma$ and order $k \in \{1, \ldots, n\}$, 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 $\mathsf{enc}_k(\langle\rangle) = \perp^k$ and $\mathsf{enc}_k(\langle a^u, u_1, \ldots, u_k\rangle) = \langle a, k\rangle(\mathsf{enc}_i(u), \mathsf{enc}_1(u_1), \ldots, \mathsf{enc}_k(u_k))$, 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 $\Sigma$ is as defined above, and $\mathcal{R}$ contains a rule $p, l \rightarrow P, r$ for each rule in $(p, a, o, P) \in \Delta$ and orders $m, m_1$, where $l \rightarrow r$ is as follows (cf. also Fig. 1 in the appendix of the technical report [11]

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

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 $\mathsf{z}_{i..j}$:

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

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 3.3 and 3.4.

▸ **Theorem 4.2** ([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 $\lambda Y$-terms can be encoded as an AOTPS. Essentially, this encoding was already given in the presentation of the Krivine machine operating on $\lambda Y$-terms from [24], 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 of the technical report [11] we present also a converse reduction (as announced earlier in Theorem 3.8), thus showing that the two models are in fact equivalent.

A *type* is either the basic type 0 or $\alpha \to \beta$ for types $\alpha, \beta$. 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 $\Gamma$ 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 $(Y M^{\alpha \to \alpha})^\alpha$. 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* $\rho$ is a finite type-preserving function assigning closures to variables, and a *closure* $C^\alpha$ is a pair consisting of a term of type $\alpha$ 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 $(M, \rho)$ is *valid* if $\rho$ 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 $x^\alpha$ in $M$. Sometimes, we need to restrict an environment $\rho$ by discarding some bindings in order to turn a closure $(M, \rho)$ into a valid one. Given a term $M$ and an environment $\rho$, the *restriction* of $\rho$ to $M$, denoted $\rho|_M$, is obtained by removing from $\rho$ all bindings for variables which are not free in $M$. In this way, if $(M, \rho)$ is a closure where $\rho$ assigns valid closures to at least all variables which are free in $M$, then $(M, \rho|_M)$ is a valid closure. In a closure $(M, \rho)$, $M$ 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 $\alpha$ 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

[6] Cf. also [22] for a definition of the Krivine machine in a different context.

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 $(p, C^\alpha, C_1^{\alpha_1}, \ldots, C_k^{\alpha_k}) \in \mathcal{C}_{\mathcal{M}}$, $p \in Q$, $C^\alpha \in Cl^\alpha(K^0)$ is the *head closure*, and $C_1^{\alpha_1} \in Cl^{\alpha_1}(K^0), \ldots, C_k^{\alpha_k} \in Cl^{\alpha_k}(K^0)$ 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 $\Gamma$, in which case the transitions in $\Delta$ control how the state changes (cf. also Fig. 2 in the appendix of the technical report [11] for a pictorial representation):

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

We say that $\mathcal{M}$ is *non-deterministic* if $\langle \Gamma, Q, \Delta \rangle$ is non-deterministic and all letters in $\Gamma$ 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 $c \in \mathrm{Pre}^*(T \times (\bigcup_{\alpha=\alpha_1\to\cdots\to\alpha_k\to 0} Cl^\alpha(K^0) \times Cl^{\alpha_1}(K^0) \times \cdots \times Cl^{\alpha_k}(K^0)))$.

**Encoding.** Following [24], 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 $\langle x_1^{\beta_1}, \ldots, x_n^{\beta_n} \rangle$ appearing in $K^0$. For a type $\alpha$, 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

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

Here, $N^\alpha$ is a symbol of $\mathsf{rank}(N^\alpha) = n$ and $\mathsf{ord}(N^\alpha) = \mathsf{ord}(\alpha)$. Moreover, if $\alpha = \alpha_1 \to \cdots \to \alpha_k \to 0$ 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, $\perp_i$ is a leaf of order $i$. The set of control locations is $Q' = Q \cup \bigcup_{(p \xrightarrow{a} P_1 \cdots P_k) \in \Delta} \{(1, P_1), \ldots, (k, P_k)\}$. A closure $(N^\alpha, \rho)$ is encoded recursively as $\mathsf{enc}(N^\alpha, \rho) = N^\alpha(t_1, \ldots, t_n)$, where, for every $i \in \{1, \ldots, n\}$, i) if $x_i \in \mathsf{FV}(N^\alpha)$ then $t_i = \mathsf{enc}(\rho(x_i))$, and ii) $t_i = \perp_{\mathsf{ord}(\beta_i)}$ otherwise (recall that $\beta_i$ is the type of $x_i$). A configuration $c = (p, (N^\alpha, \rho), C_1^{\alpha_1}, \ldots, C_k^{\alpha_k})$ is encoded as the tree $\mathsf{enc}(c) = [N^\alpha](t_1, \ldots, t_n, \mathsf{enc}(C_1^{\alpha_1}), \ldots, \mathsf{enc}(C_k^{\alpha_k}))$, where the first $n$ subtrees encode the closure $(N^\alpha, \rho)$, i.e., $\mathsf{enc}(N^\alpha, \rho) = N^\alpha(t_1, \ldots, t_n)$. 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 $x^\alpha$ 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 $N^\alpha$ for $\mathsf{level}(\alpha) = l$.

Below, we assume that $\alpha = \alpha_1 \to \cdots \to \alpha_k \to 0$, that variable $\mathsf{y}_j$ has order $\mathsf{ord}(\alpha_j)$ for every $j \in \{0, \ldots, k\}$, and that variables $\mathsf{x}_i$ and $\mathsf{z}_i$ have order $\mathsf{ord}(\beta_i)$ for every $i \in \{1, \ldots, n\}$. Notice that $\mathsf{ord}(\alpha) < \mathsf{ord}(\alpha_1), \ldots, \mathsf{ord}(\alpha_k)$. Moreover, we write $\mathbf{x} = \langle \mathsf{x}_1, \ldots, \mathsf{x}_n \rangle$, $\mathbf{z} = \langle \mathsf{z}_1, \ldots, \mathsf{z}_n \rangle$, and $\mathbf{y} = \langle \mathsf{y}_1, \ldots, \mathsf{y}_k \rangle$. 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:

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

The first rule satisfies the ordering condition since the shared variables $\mathsf{y}_i$ 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 3.3 and 3.4.

▶ **Theorem 4.3** ([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 [16]. 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]. [16] 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 of the technical report [11], 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. 3.8), and by recalling the known fact that the latter can be encoded by a 1-stack annotated multi-pushdown system of order $mn$ [26], 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 3.3.

▶ **Theorem 4.4** ([16]). *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 [16] 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 [20] 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 $\lambda$-calculus in a clear way. Then [26] adapted it to $\lambda Y$-calculus, and have shown that safe $\lambda Y$-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 [26]. 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 3.8 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 [27]. 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 [18]. 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 L. Clemente, P. Parys, S. Salvati, and I. Walukiewicz. Ordered tree-pushdown systems. Technical report, University of Warsaw, 2015.

12 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.

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

14 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.

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

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

17 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.

18 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.

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

20 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.

21 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.

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

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

24 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.

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

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

27 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.