The Complexity of Model Checking (Collapsible) Higher-Order Pushdown Systems

Matthew Hague[1] and Anthony Widjaja To[2]

1,2 Oxford University Computing Laboratory
Wolfson Building, Parks Road
Oxford, OX1 3QD

Abstract

We study (collapsible) higher-order pushdown systems — theoretically robust and well-studied models of higher-order programs — along with their natural subclass called (collapsible) higher-order basic process algebras. We provide a comprehensive analysis of the model checking complexity of a range of both branching-time and linear-time temporal logics. We obtain tight bounds on data, expression, and combined-complexity for both (collapsible) higher-order pushdown systems and (collapsible) higher-order basic process algebra. At order-k, results range from polynomial to $(k+1)$-exponential time. Finally, we study (collapsible) higher-order basic process algebras as graph generators and show that they are almost as powerful as (collapsible) higher-order pushdown systems up to MSO interpretations.

1998 ACM Subject Classification D.2.4

Keywords and phrases Higher-Order, Collapsible, Pushdown Systems, Temporal Logics, Complexity, Model Checking

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2010.228

1 Introduction

Recently, there has been a burgeoning interest in collapsible higher-order pushdown systems (CPDSs), both as generators of structures and as models of higher-order computation. Whereas an order-1 pushdown system augments a finite-state automaton with an unbounded stack memory, a higher-order pushdown system (HOPDS) provides a nested "stack-of-stacks" structure. CPDSs allow a further backtracking operation called *collapse*.

Higher-order pushdown automata (HOPDA) were introduced by Maslov [24]. *Higher-order pushdown systems* (HOPDS) are HOPDA viewed as generators of infinite trees or graphs. Recently these models have been generalised to *collapsible pushdown systems* (CPDS) [17, 19]. In terms of expressivity, order-k CPDSs generate the same class of ranked trees as deterministic order-k recursion schemes [17]. The analogous result holds for *safe* recursion schemes and HOPDSs [18]. These systems provide a natural model for higher-order programs with (unbounded) recursive function calls and are therefore useful in software verification. Further results show an intimate connection with the Caucal hierarchy [10, 11]. For verification, reachability properties — which ask whether a given set of control states can be reached from the initial configuration — are complete for $(k-1)$-ExpTime [5, see appendix], whilst μ-calculus properties are k-ExpTime-complete [7, 26, 17]. Despite these high complexities, Kobayashi has verified resource usage properties of higher-order programs [20] using a novel approach based on intersection types [21, 23].

Hitherto, there has been little work addressing the precise complexity of model checking higher-order programs with respect to the common temporal logics. In most cases, there


© Matthew Hague and Anthony Widjaja To;
licensed under Creative Commons License NC-ND
IARCS Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2010).
Editors: Kamal Lodaya, Meena Mahajan; pp. 228–239

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

	(Collapsible) HOPDS		(Collapsible) HOBPA	
	Data	Expression & Combined	Data	Expression & Combined
μLTL / LTL	$(k-1)$-ExpTime	k-ExpTime	P-time	k-ExpTime
LTL(F, X)	$(k-1)$-ExpTime	k-ExpTime	P-time	k-ExpTime
LTL(U)	$(k-1)$-ExpTime	k-ExpTime	P-time	k-ExpTime
CTL	k-ExpTime	k-ExpTime	P-time	k-ExpTime
CTL+	k-ExpTime	$(k+1)$-ExpTime	P-time	$(k+1)$-ExpTime
CTL*	k-ExpTime	$(k+1)$-ExpTime	P-time	$(k+1)$-ExpTime
EF	$(k-1)$-ExpSpace-hard	$(k-1)$-ExpSpace-hard	P-time	$(k-1)$-ExpSpace-hard

Figure 1 The complexity of model checking order-k higher-order systems. Unless stated, all results are complete.

is currently a single or double exponential gap in the best known upper and lower bounds (derived usually from μ-calculus and reachability respectively). One main contribution of this paper is a nearly complete picture of the model checking complexities against temporal logics. In particular, we consider data complexity (formulas are fixed), expression complexity (systems are fixed), and combined complexity (both formulas and systems are input parameters). Table 1 (left column) summarises our results. In all cases, our lower bounds hold without the collapse operation, whilst our upper bounds allow collapse.

Basic process algebras (BPAs) are a natural and well-studied subclass of order-1 PDSs (cf. [6]), which are suitable abstractions for modelling the control-flow of sequential programs (cf. [2, 14]). We propose higher-order extensions of BPAs, called *(collapsible) higher-order basic process algebras* (HOBPAs), that form a natural subclass of (collapsible) HOPDSs. This differs from the single-state HOPDSs introduced by Bouajjani and Meyer [3]. As graph generators, (collapsible) HOBPAs are almost as powerful as (collapsible) HOPDSs in the following sense: (1) like CPDS, there exists a collapsible order-2 BPA whose graph has an undecidable monadic second-order logic (MSO) theory, and (2) the class of graphs generated by order-k BPAs coincide with those generated by order-k PDSs up to MSO interpretations. In this paper, we provide an almost complete picture for the model checking complexities of standard temporal logics over (collapsible) HOBPAs. See Table 1 (right column). We show that the restriction to HOBPA does not, in most cases, simplify the model checking problem; a notable exception is for data complexity, where the problem becomes polynomial time. Again, our lower bounds hold without collapse, whilst our upper bounds allow collapse.

Similar analyses appear across a number of papers for the special case of order-1 pushdown systems [1, 6, 30, 31, 4]. In all cases we generalise the resulting picture in a natural manner. That is, a 1-ExpTime-complete complexity becomes k-ExpTime-complete, and so on. Our upper bound results concern the data complexity of Collapsible HOBPAs and the data and combined complexities for LTL over CPDSs. Previous work studied reachability, LTL and the alternation-free μ-calculus [16, 27, 13] over HOPDS without collapse. However, we believe the LTL algorithm contains an error, and provide a new algorithm. Furthermore, the alternation-free μ-calculus algorithm in [16] is not optimal. Our remaining results concern lower bounds. We begin with two techniques from in the literature: (1) Engelfriet's characterization of complexity classes k-ExpTime by extensions of HOPDAs (e.g. with space-bounded worktape) [13], and (2) Cachat and Walukiewicz's more "direct" approach via encodings of large numbers using HOPDSs [8]. We employ Technique (1) to prove the lower bounds for LTL (and its fragments), CTL, CTL+, and CTL*. This does not mean that the proofs of the results are immediate: it was left as an open problem in [8] whether the two techniques can be used to derive these lower bounds. Since Technique (1) seems only suited to deriving k-ExpTime lower bounds (for some k), we give two variations of

Technique (2) to derive $(k-1)$-ExpSpace lower bounds for EF model checking over HOPDSs and HOBPAs (the latter proof is substantially more involved). The lower bound proofs in this paper suggest that Technique (1) yields simpler proofs, while Technique (2) offers more flexibility.

The preliminaries are given in §2. We begin in §3 with the results for fixed formulas over collapsible HOBPA. In §4 we discuss branching-time logics, and linear-time in §5. Finally, we conclude this paper with future work in §6. Due to the length and intricate nature of the proofs, we relegate the full details into the full version.

2 Preliminaries

We define (collapsible) higher-order pushdown systems and basic process algebra and give a result of Engelfriet used in some proofs. Note, after defining higher-order and collapsible stores, we only define higher-order systems. For the collapsible version, simply replace the higher-order store with a collapsible one, expanding the stack operations accordingly. Also, the definitions generalise from non-deterministic to alternating in the standard way.

Higher-Order Collapsible Pushdown Stores

We begin by defining a higher-order pushdown store. Collapse links will be introduced afterwards. Intuitively, a higher-order store is a stack of lower order stacks.

▶ **Definition 1** (k-Stores). Let C_0^Σ be a finite alphabet Σ with $[,] \notin \Sigma$. For $k \geq 1$, the set of *k-stores* C_k^Σ contains all $[\gamma_1 \dots \gamma_m]$ with $m \geq 1$ and $\gamma_i \in C_{k-1}^\Sigma$ for all $1 \leq i \leq m$.

There are two operations defined over 1-stores (for all $w \in \Sigma^*$)

$$push_w[a_1 \dots a_m] = [wa_2 \dots a_m] \quad \text{and} \quad top_1[a_1 \dots a_m] = a_1 \ .$$

We define $pop_1 = push_\varepsilon$. Let $\mathcal{O}_1 = \{\ push_w \mid w \in \Sigma^*\ \}$. When $k > 1$, a push operation creates a copy of the topmost stack, while a pop removes it. We assume w.l.o.g. that $\Sigma \cap \mathbb{N} = \emptyset$, where $\mathbb{N}$ is the set of natural numbers. Finally, let $[\gamma_1 \dots \gamma_m] \in C_k^\Sigma$ for some k.

$$\begin{array}{rcll}
push_w[\gamma_1 \dots \gamma_m] & = & [push_w(\gamma_1)\gamma_2 \dots \gamma_m] & \\
push_l[\gamma_1 \dots \gamma_m] & = & [push_l(\gamma_1)\gamma_2 \dots \gamma_m] & \text{if } 2 \leq l < k \\
push_k[\gamma_1 \dots \gamma_m] & = & [\gamma_1\gamma_1\gamma_2 \dots \gamma_m] & \\
pop_l[\gamma_1 \dots \gamma_m] & = & [pop_l(\gamma_1)\gamma_2 \dots \gamma_m] & \text{if } 1 \leq l < k \\
pop_k[\gamma_1 \dots \gamma_m] & = & [\gamma_2 \dots \gamma_m] & \text{if } m > 1 \\
top_l[\gamma_1 \dots \gamma_m] & = & top_l(\gamma_1) & \text{if } 1 \leq l < k \\
top_k[\gamma_1 \dots \gamma_m] & = & \gamma_1 &
\end{array}$$

Note, when $m = 1$, pop_k is undefined. Let $\mathcal{O}_k = \{\ push_w \mid w \in \Sigma^*\ \} \cup \{\ push_l, pop_l \mid 1 < l \leq k\ \}$. We designate $\perp$ to be a *bottom of stack* symbol that is neither pushed nor popped. Let $[w]_1 = [w]$ and $[w]_k = [[w]_{k-1}]$.

For collapse, the order-1 push operation $push_w$ is replaced with $push_{a_1^{i_1} \dots a_m^{i_m} b}$ for $1 \leq i_z \leq k$ and $a_z, b \in \Sigma$ where $1 \leq z \leq m$. A $push_{a_1^{i_1} \dots a_m^{i_m} b}$ on some stack with top_1 character a is equivalent to $push_{a_1 \dots a_m b}$ except each a_z is augmented with a pair $(i_z, 1)$. That is, the top of stack character $a^{(i,j)}$ is replaced by $a_1^{(i_1,1)} \dots a_m^{(i_m,1)} b^{(i,j)}$. The collapse operation from a character $a^{(i,j)}$ is equivalent to j applications of pop_i. The second component j is incremented at every $push_i$. Hence, (i, j) is a link to the order-$(i-1)$ stack beneath the character when it was first pushed.

Consider $[[[\bot]\,[\bot]]]$. Applying $push_{a^2\bot}$ gives $\left[\left[\left[a^{(2,1)}\ \bot\right]\,[\bot]\right]\right]$. The pair $(2,1)$ points to $[\bot]$. A $push_2$ leads to $\left[\left[\left[a^{(2,2)}\ \bot\right]\,\left[a^{(2,1)}\ \bot\right]\,[\bot]\right]\right]$. Note the second component is incremented in the copy of a, and, thus, $(2,2)$ also points to $[\bot]$. A subtlety occurs after a $push_3$. We obtain the stack below on the left, where the copies of a now refer to the copies of $[\bot]$ within the order-2 stack they occupy. After a collapse, we obtain the stack on the right.

$$\left[\begin{array}{l}\left[\left[a^{(2,2)}\ \bot\right]\,\left[a^{(2,1)}\ \bot\right]\,[\bot]\right]\\ \left[\left[a^{(2,2)}\ \bot\right]\,\left[a^{(2,1)}\ \bot\right]\,[\bot]\right]\end{array}\right] \qquad \left[\begin{array}{c}[[\bot]]\\ \left[\left[a^{(2,2)}\ \bot\right]\,\left[a^{(2,1)}\ \bot\right]\,[\bot]\right]\end{array}\right]$$

Formally, we define order-k stores with links in terms of order-k stores over the infinite alphabet $\Sigma = \{\ a^{(i,j)}\ \mid i,j \in \mathbb{N}\ \}$. The set of operations over an order-k store with links is

$$\begin{aligned}\mathcal{O}_k^c \quad = \quad & \left\{\ push_{a_1^{i_1}\ldots a_m^{i_m} b}\ \mid \forall 1 \le z \le m . 1 \le i_z \le k \wedge a_z \in \Sigma\ \right\}\\ & \cup \{\ push_l, pop_l, collapse\ \mid 1 < l \le k\ \}\ .\end{aligned}$$

Note that this set of operations is slightly different from the original definition [17]. We show, in the full version, that the definitions are equivalent. The semantics of the operations are given below, in terms of the standard order-k pushdown operators, and an order-k stack $\gamma = [\gamma_1 \ldots \gamma_m]$. Let $\gamma^{<k>}$ be the stack γ where each superscript (i,j) with $i \ge k$ is replaced with $(i, j+1)$.

$$\begin{array}{rcll} push_{a_1^{i_1}\ldots a_m^{i_m} b}(\gamma) & = & push_{a_1^{(i_1,1)}\ldots a_m^{(i_m,1)} b_m^{(i',j')}}(\gamma) & \text{where } top_1(\gamma) = b^{(i',j')}\\ collapse(\gamma) & = & pop_i^j(\gamma) & \text{where } top_1(\gamma) = b^{(i,j)}\\ push_k[\gamma_1 \ldots \gamma_m] & = & [\gamma_1^{<k>}\gamma_1 \ldots \gamma_m] & \\ push_l[\gamma_1 \ldots \gamma_m] & = & [push_l(\gamma_1)\gamma_2 \ldots \gamma_m] & \text{where } l < k \end{array}$$

Higher-Order Pushdown Systems

A HOPDS is a finite-state system with a higher-order store. The finite-state component is the control state. At each step, the applicable transitions are determined by the control state and the top_1 character of the stack. Each transition updates the control state and the stack.

▶ **Definition 2.** An *order-k PDS* is a tuple $(P, \mathcal{R}, \Sigma, p_0, \bot)$ where P is a finite set of control states, $\mathcal{R} \subseteq P \times \Sigma \times \mathcal{O}_k \times P$ is a finite set of rules, Σ is a finite stack alphabet, $p_0 \in P$ is an initial control state and $\bot \in \Sigma$ is a bottom of stack symbol.

A configuration of a higher-order PDS is a pair $\langle p, \gamma\rangle$ where $p \in P$ and γ is a k-store. We have a transition $\langle p, \gamma\rangle \hookrightarrow \langle p', \gamma'\rangle$ iff we have $(p, a, o, p') \in \mathcal{R}$, $top_1(\gamma) = a$ and $\gamma' = o(\gamma)$. The initial configuration is $\langle p_0, [\bot]_k\rangle$.

Higher-Order Basic Process Algebra

An order-1 BPA is an order-1 PDS with a single control state. By applying the same restriction, Bouajjani and Meyer have obtained one definition of higher-order BPA [3]. However, consider $\langle p, [[a\ \bot]]\rangle$ and the rule (omitting the control) $(a, push_2)$. We obtain $\langle p, [[a\ \bot]\,[a\ \bot]]\rangle$ and the same rule can be applied, ad infinitum. However, at order-1 we may use $(a, push_{bc})$ to rewrite the top character before adding a new top character. Hence, at order-j, it is natural to be able to rewrite the top order-$(j-1)$ stack, before adding a new one. Consequently, we introduce $(a, push_j, b) = push_a; push_j; push_b$ for all $2 \le j \le k$. E.g., such rules can simulate $push_2; push_3; pop_2$. Let $\mathcal{O}'_k = \{\ push_w \mid w \in \Sigma^*\ \} \cup \{\ (a, push_j, b), pop_j \mid 1 < j \le k\ \}$.

▶ **Definition 3.** An *order-k BPA* is a tuple $(\mathcal{R}, \Sigma, \bot)$ where $\mathcal{R} \subseteq \Sigma \times \mathcal{O}'_k$ is a finite set of rules, Σ is a finite stack alphabet, and $\bot \in \Sigma$ is the bottom of stack symbol.

We mention two results on the expressive power of HOBPAs as graph generators. Familiarity with monadic second-order logic (MSO) is assumed (cf. [28]). As graph generators, (collapsible) HOBPAs are as powerful as (collapsible) HOPDAs up to monadic second-order logic (MSO) interpretation in the following sense. First, it is known that there exists an order-2 CPDA generating a graph with an undecidable MSO theory [17]. In contrast, over HOPDAs, MSO is decidable. This CPDA is not a collapsible HOBPA. On the other hand, using the ideas from [17], it is not difficult to come up with an order-2 collapsible HOBPA generating a graph with an undecidable MSO theory.

▶ **Proposition** 1. There exists a fixed collapsible order-2 BPA which generates a graph with an undecidable MSO theory.

We sketch the proof of this proposition in the full version. Secondly, we discuss the expressive power of HOBPAs without collapse. Carayol and Wöhrle [9, 10] gave a fixed graph Δ_2^k, for each integer $k > 0$, such that the class of graphs that are MSO-interpretable in the graphs generated by order-k PDSs coincide with the class of graphs that are MSO-interpretable in Δ_2^k. It is easy to check that Δ_2^k can be generated by a fixed order-k BPAs (e.g. see [9]), which implies the following proposition.

▶ **Proposition** 2. The class of graphs that are MSO-interpretable in the graphs generated by order-k BPAs coincide with the class of graphs MSO-interpretable in the graphs generated by order-k PDSs.

Higher-Order Pushdown Automata with an Auxiliary Work Tape

For the lower bound proofs we use HOPDA with a space-bounded work tape. That is, in addition to the control state and the stack, the machine has a bounded, two-way work tape. This tape operates identically to the tape in a Turing machine.

▶ **Definition 4.** An *order-k PDA with an $s(n)$-space work tape* is a tuple $(P, \mathcal{R}, \Sigma, \Gamma \cup \{\varepsilon\}, \Delta, p_0, \bot, \square, \mathcal{F})$ where P is a finite set of control states, $\mathcal{R} \subseteq (P \times \Gamma \cup \{\varepsilon\} \times \Sigma \times \Delta) \times \mathcal{O}_k \times (\Delta \times \{l, r\} \times P)$ is a finite set of rules, Σ is a finite stack alphabet, Γ is a finite input alphabet, Δ is a finite tape alphabet, $p_0 \in P$ is an initial control state, $\bot \in \Sigma$ is the bottom of stack symbol, $\square \in \Delta$ denotes a blank tape cell and $\mathcal{F} \subseteq P$ is a set of accepting control states.

Given an input word of length n, a configuration of a HOPDA with $s(n)$ bounded work tape is a tuple $\langle p, \gamma, t, j\rangle$ where $p \in P$, γ is a k-store, t (the tape contents) is a word in $\Delta^{s(n)}$ and $1 \leq j \leq s(n)$ indicates the position of the read/write head on the tape.

A rule $(p, \alpha, a, x, o, y, d, p') \in \mathcal{R}$ can be applied when the current control state is p, the input character is α, the top-of-stack character is a, and the tape contents at position j are x. The control state is then updated to p', the command o is applied to the stack, and y is written to the tape. The tape head moves accordingly for $d = l$ (left) or $d = r$ (right).

More formally, we have a transition $\langle p, \gamma, t, j\rangle \overset{\alpha}{\hookrightarrow} \langle p', \gamma', t', j'\rangle$ iff we have $(p, \alpha, a, x, o, y, l, p') \in \mathcal{R}$, $j > 1$, $top_1(\gamma) = a$, $t(j) = x$, $\gamma' = o(\gamma)$, $t'(j) = y$, $t'(h) = t(h)$ for all $h \neq j$ and $j' = j - 1$ or we have $(p, \alpha, a, x, o, y, r, p') \in \mathcal{R}$, $j < s(n)$, $top_1(\gamma) = a$, $t(j) = x$, $\gamma' = o(\gamma)$, $t'(j) = y$, $t'(h) = t(h)$ for all $h \neq j$ and $j' = j - 1$. For $\alpha \neq \varepsilon$, we write $c \overset{\alpha}{\hookrightarrow}_\varepsilon c'$ whenever there is a sequence of ε-transitions from c to some c_1, an α-transition from c_1 to c_2 and a sequence of ε-transitions to c'. A word $\alpha_1, \ldots, \alpha_n$ is accepted by the automaton iff $c_n = \langle p, \gamma\rangle$ and $p \in \mathcal{F}$ and $c_0 \overset{\alpha_1}{\hookrightarrow}_\varepsilon \cdots \overset{\alpha_n}{\hookrightarrow}_\varepsilon c_n$ where $c_0 = \langle p_0, [\bot]_k, \square^{s(n)}, 1\rangle$.

Temporal Logics

We will assume familiarity with the temporal logics discussed, remarking only that μLTL is LTL extended with fixed point operators. Full definitions can be found in the literature [12, 29]. We assume, for all logics, the valuations of atomic propositions depend only on the control state and current top-of-stack character, referred to as a *head*. That is, $\Lambda : P \times \Sigma \to 2^{Prop}$ is an assignment of satisfied atomic propositions from the set $Prop$ to each head in $P \times \Sigma$. We say a system satisfies a formula if it holds at the initial state of the system.

Engelfriet's Results

We use the following theorem of Engelfriet [13] in some proofs. Let NSPACE($s(n)$)-P^k denote the class of languages accepted by a non-deterministic order-k PDA with an $s(n)$-space-bounded work tape, where n is the length of the input word. Similarly, ASPACE($s(n)$)-P^k denotes the class of languages accepted by an alternating order-k PDA with an $s(n)$-space-bounded work tape. Finally $\bigcup_{d>0}$ DTIME($exp_k(ds(n))$) is the class of languages accepted by a time-bounded Turing machine, where $exp_0(x) = x$ and $exp_k(x) = 2^{exp_{k-1}(x)}$.

▶ **Theorem 5** ([13], Thm. 2.5). *For any $k \geq 1$ and $s(n) \geq log(n)$, we have NSPACE($s(n)$)-P^k = ASPACE($s(n)$)-P^{k-1} = $\bigcup_{d>0}$ DTIME($exp_k(ds(n))$).*

That is, a non-deterministic order-k PDA with a polynomially-bounded work tape exists for every k-ExpTime language, and an alternating order-k PDA with a polynomially-bounded work tape exists for every $(k+1)$-ExpTime language.

3 Model Checking Collapsible HOBPA Against Fixed Formulas

We begin with a P-time algorithm for model checking collapsible HOBPA against fixed formulas. Hardness follows from the P-time-hardness of context-free language emptiness [15].

▶ **Theorem 6.** *For any logic that can be translated into μ-calculus, model checking collapsible HOBPA against a fixed formula is in P-time.*

Proof. As argued in the full version, any collapsible HOBPA can be simulated by a CPDS with a fixed number of control states. Therefrom, and since the formula is fixed, we construct a CPDS parity game with a fixed number of control states. At order-k, the winner of these games can be determined in k-ExpTime in the number of control states, and polynomial in the alphabet [17]. Hence, the algorithm runs in P-time. ◀

4 Branching Time

We begin by observing, for CPDS, the upper bounds for CTL, CTL+ and CTL* can be obtained by translating into μ-calculus, which has a k-ExpTime model checking problem. For CTL, the translation is polynomial. For CTL+ and CTL* it is exponential, giving $(k+1)$-ExpTime, and k-ExpTime when the formula is fixed. For the lower bound results, we discuss EF, CTL and then CTL+.

▶ **Theorem 7.** *For a fixed formula, and a given order-k CPDS, model checking CTL, CTL+ and CTL* is in k-ExpTime. For a non-fixed system and non-fixed formula, CTL is k-ExpTime, and CTL+ and CTL* are in $(k+1)$-ExpTime.*

4.1 Lower Bounds for EF

In most cases, we are able to derive optimal lower bounds using Theorem 5. However, Theorem 5 is not immediately applicable for (e.g.) $(k-1)$-ExpSpace problems. In the case of EF-logic, the model checking problem over order-1 PDSs and BPAs is PSpace-complete [25, 31]. We now give $(k-1)$-ExpSpace lower bounds for data complexity of order-k PDSs and the expression complexity of order-k BPAs (and thus of order-k PDSs) using the technique of [8] of encoding large numbers. We conjecture that these lower bounds are tight (currently, the best upper bound is k-ExpTime, which is inherited from μ-calculus).

▶ **Theorem 8.** *Model checking EF over order-k PDS without collapse is $(k-1)$-ExpSpace-hard, even for a fixed formula.*

Proof. (sketch) We reduce membership for a given $(k-1)$-ExpSpace Turing machine M using $exp_{k-1}(p(n))$ space on an input word of length n, for some polynomial function p. Fix a number $m \in \mathbb{Z}_{>0}$, which we will later define as $p(n)$ once n is set. The proof combines the technique of [1] for proving that EF-logic over PDS is PSpace-hard and the technique of [8] for encoding and checking large numbers (i.e. k-towers of exponentials) using operations in $\mathcal{O}_k$.

We shall start by briefly recalling the encoding techniques of large numbers from [8]. For each $i \in \mathbb{Z}_{>0}$, we define $\Sigma_i := \{a_i, b_i\}$ and $\Sigma_{\leq i} := \bigcup_{j=1}^{i} \Sigma_i$. We now define the notion of *i-counters* by induction. A *1-counter* (of length m) is a word $\sigma_{m-1} \dots \sigma_0 \in (\Sigma_1)^m$. Such a word naturally represents the number $\sum_{i=0}^{m-1} \sigma_i 2^i$ where a_1 represents 0 and b_1 represents 1. Assuming that the notion of i-counter has been defined, an *$(i+1)$-counter* is simply a word $\sigma_r l_r \dots \sigma_0 l_0$ over $\Sigma_{\leq i+1}$, where $r = exp_i(m) - 1$, $\sigma_j \in \Sigma_{i+1}$, and l_j is an i-counter representing the number j. This $(i+1)$-counter represents the number $\sum_{j=0}^{r} \sigma_j 2^j$, where (as before) a_{i+1} and b_{i+1} are used to (respectively) represent 0 and 1.

Cachat and Walukiewicz [8] showed that a polynomial-size order-k pushdown game arena $\mathcal{P}$ with a reachability objective could be defined (depending only on m) with the following control states and properties: $counter_k$ — from configuration $(counter_k, \gamma)$ of $\mathcal{P}$, Player 0 wins iff γ ends with a k-counter; $first_k$ (resp. $last_k$) — from configuration $(first_k, \gamma)$ (resp. $(last_k, \gamma)$), Player 0 wins iff γ ends with a k-counter representing 0 (resp. $exp_{k-1}(m)$); $equal_k$ — from $(equal_k, \gamma)$, Player 0 wins iff γ ends with two k-counters representing equal values; $succ_k$ — from $(succ_k, \gamma)$, Player 0 wins iff γ ends with two k-counters representing successive values. We observe that the game element of $\mathcal{P}$ can easily be translated into fixed EF formulas (i.e. not depending on m) satisfying the same properties, the main reason being that the game arena $\mathcal{P}$ has a fixed number of rounds.

The rest of the proof uses the idea of [1]. Using an EF operator, we will first guess a word in $\Sigma_{\leq k+1}$ representing an accepting computation of M on the given input word $w = \alpha_1 \dots \alpha_n$. We then need to check that the guess is valid. That is, it represents a sequence of configurations, the initial configuration is the right form, the final configuration is reached, and consecutive configurations respect the transition relation. All these can be done by means of a fixed formula, thanks to the result above for encoding large numbers. ◀

▶ **Theorem 9.** *For a fixed order-k HOBPA without collapse, model checking EF is $(k-1)$-ExpSpace-hard.*

Proof. (sketch) The proof uses some general ideas from the previous proof, but, without control states to encode tests for large numbers, we need an entirely different construction. We briefly explain the order-2 case. Our HOBPA $\mathcal{P}$ will guess an accepting run of a fixed exponential space Turing machine M accepting an ExpSpace-complete language, obtaining a

stack of the form $[w]_2$. For the checking stage, our HOBPA $\mathcal{P}$ now tries to find some location inside the stack that is invalid. In doing so, we need to ensure that all of the information on top of this location is not destroyed. To this end, we will build a stair-like structure from $[w]_2$ by performing operations of the form $[push_1(a'); push_2; push_1(prime)]$ or of the form $[push_1(a''); push_2; push_1(dprime)]$ when seeing a topmost stack symbol a. Here, *prime* and *dprime* are simply intermediate symbols to help signify the action that was previous executed, i.e., we could simply only allow pop_1 operation when *prime* or *dprime* is seen as topmost symbol. The double prime marking is used to "remember" the *starting* point of (sub)configuration that we suspect is invalid. That is, we will have to make sure that it is put precisely once. At some point, $\mathcal{P}$ simply applies rules of the form $push_1(a')$ when a is seen without applying $push_2$, which marks the *end* point of a (sub)configuration that we suspect is invalid. We then only allow rules pop_2 when primed or double primed symbols are seen. To make sure that we see precisely one separator symbol (i.e. $a_3 \in \Sigma_3$), we can use an EF formula saying facts about the location of the double primed symbol a_3''. Such a stair-like structure will allow us to define EF formulas that play the roles of $counter_i$, $first_i$, $last_i$, $equal_i$, and $succ_i$ and their associated EF formulas in the previous proof. ◀

4.2 Lower Bounds for CTL

Data Complexity We know, for a fixed formula, model checking CTL, CTL+ and CTL* against HOPDSs is in k-ExpTime. Here, we show the lower bound.

▶ **Theorem 10.** *For a fixed formula, model checking CTL over a given order-k HOPDS without collapse is k-ExpTime-hard.*

Proof. (sketch) From Theorem 5 we take a language that is k-ExpTime-hard and fix an equivalent order-$(k-1)$ alternating HOPDA with a polynomially space-bounded work tape $\mathcal{P}$. The reduction is inspired by Bozzelli [4].

We use an order-k stack to navigate a computation tree of the HOPDA. To simulate the work tape, at each step, after an operation on the order-$(k-1)$ stack, a sequence of tape symbols are pushed on to the top order-1 stack. Then, the system can do a check branch to ensure the guessed tape is consistent with the previous, or continue simulating the execution. To continue, an order-k push saves the current state (for backtracking), the work tape is erased, the next rule is announced, and a $push_k$ remembers the rule. This process repeats. Consider the example order-3 stack below.

$$\left[\begin{array}{c}\left[\begin{array}{c}[tw_1]\\ [w_2]\\ \cdots\end{array}\right]\end{array}\right.\quad\left[\begin{array}{c}[r\ldots]\\ \cdots\end{array}\right]\quad\left[\begin{array}{c}[t'w_1']\\ [w_2']\\ \cdots\end{array}\right]\quad\left.\cdots\right]$$

This stack is at a configuration with the tape given by the word t and order-2 stack $[[w_1]\,[w_2]\ldots]$, which can backtrack to a configuration with tape t' and order-2 stack $[[w_1']\,[w_2']\ldots]$. The rule r connects the configurations. When an accepting configuration is seen, or the children of the current node have been fully explored, we backtrack using pop_k, and check untested universal branches. The automaton accepts when the (marked) initial stack is reached. That is, all paths have been explored, and found to be accepting.

The check branches have further branches for each of the polynomially many positions of the work tape. Each branch uses the control state to find the correct position, and then, using the control state, compares it with the corresponding positions in the previous work tape, which is recovered via pop_k operations.

The CTL formula $E((op \wedge AX(check \rightarrow AFgood))Ufin)$ asserts a path encoding an accepting tree exists, and checking branches all accept. The proposition *op* indicates the current path is simulating a tree, and *check* indicates a checking branch. Finally, *good* indicates that the check has been passed, and fin denotes the (successful) completion of the run. ◀

Expression Complexity The following theorem takes care of all cases.

▶ **Theorem 11.** *For a fixed order-k HOBPA without collapse, model checking CTL is k-ExpTime-hard.*

Proof. (sketch) The proof is in stages. First, we adapt Theorem 10, unfixing the formula to fix the HOPDS. A HOBPA is obtained using a more complex formula. There are two main assertions we move from the HOPDS to the formula. First, the check branch becomes a straight-line sequence of pops and the formula uses sequences of EX to compare positions. Secondly, the word position being read has to be guessed and added to the work tape information, then checked by the formula. Hence, we have the result for a fixed HOPDS.

To obtain a HOBPA the main difficulty is that control states were used to separate the check, backtrack and simulation phases of the model. Here we use the $(a, push_j, b)$ rules so that, when pushing, the automaton can read a character a, and mark it $\underline{a}$ in the next applied rule. Hence the system knows when it is moving up or down the stack, and when it has simulated a stack action. Also the automaton announces the intended phases. For example, the check branch announces "check", removes the work tape, announces "pop_k", pops, announces "check" again and removes the work tape. The formula can then use EX^j to look into the first tape, and $E(tapeU(pop_k \wedge EX(check \wedge EX^j\varphi)))$ to look j steps into the next tape, where *tape* indicates that a tape character is seen. ◀

4.3 Lower Bounds for CTL+

For data complexity, the CTL lower bound transfers to CTL+ and CTL*. For the expression complexity, the following theorem suffices.

▶ **Theorem 12.** *For a fixed order-k HOBPA without collapse, model checking CTL+ is* $(k+1)$*-ExpTime-hard.*

Proof. (sketch) First we adapt the proof of Theorem 10 to show, CTL* is $(k+1)$-ExpTime-hard. We then replace the CTL* formula with a CTL+ formula. Then we show how to fix the system, and restrict ourselves to HOBPA.

For a non-fixed formula and system, our CTL* proof adapts Bozzelli's order-1 proof [4]. Fix a language that is hard for $(k+1)$-ExpTime and an equivalent order-$(k-1)$ alternating HOPDA with an *exponentially* space-bounded work tape. The system proceeds as before, but guesses the length of the work tape and uses a word $bin_n(0)c_0bin_n(1)c_1 \cdots bin_n(2^n-1)c_{2^n-1}$ to represent it, where $bin_n(i)$ is the n-digit binary representation of i, and c_j are cell contents. The check phase has one branch to check the cell counters are sequential, and the others, instead of just popping down the stack, mark a position in each tape. The formula asserts, when markings are sensible, the tape contents are locally consistent. This can, in fact, be encoded in CTL+ by taking advantage of straight-line parts of the execution and adding extra markings. Obtaining a fixed HOBPA is similar to the CTL case, with some extra tricks. E.g., to ensure each marker is placed once and in the correct order. ◀

5 Linear Time

We consider the linear time logics. We first deal with the upper bound for linear time μ-calculus (μLTL) — and hence LTL — before considering the lower bounds in turn.

5.1 Upper Bounds for μLTL

Since the linear time μ-calculus (μLTL) does not translate polynomially into μ-calculus, we show the k-ExpTime upper bound of model checking μLTL against CPDS separately. Note that μLTL trivially subsumes all other linear time logics considered in this paper.

▶ **Theorem 13.** *Model checking μLTL against order-k CPDSs is in k-ExpTime for a non-fixed formula, and $(k-1)$-ExpTime for a fixed formula.*

Proof. (sketch) We can translate any μLTL formula φ into a Büchi automaton $\mathcal{B}$ at an exponential cost [29]. From a given CPDS $\mathcal{P}$ we construct a product CPDS $\mathcal{P}_B = \mathcal{P} \times \mathcal{B}$ which has a Büchi acceptance condition such that $\mathcal{P}_B$ accepts iff $\mathcal{P}$ does not satisfy φ.

An order-k Büchi CPDS is a CPDS parity game with two colours and only one player. Hence, non-emptiness can be reduced to determining the winner in a parity game, which takes k-ExpTime in the size of the CPDS [17]. Since the Büchi CPDS is exponential in the size of φ, this complexity is too high. The algorithm for an order-k parity game is by a reduction to an order-$(k-1)$ game of exponential size. Because the Büchi CPDS has one player, we can avoid the exponential blow up, constructing an order-$(k-1)$ game of polynomial size. This can be solved in $(k-1)$-ExpTime in the size of the Büchi CPDS, giving an algorithm in k-ExpTime for a non-fixed formula, and $(k-1)$-ExpTime for a fixed formula. ◀

5.2 Lower Bounds for LTL

We first give a matching lower bound for data complexity (fixed formula) of LTL, which already hold for its fragments LTL(F,X) and LTL(U). Since we have previously shown that order-k HOBPA can be analysed in P-time for fixed formulas, it remains to consider HOPDS.

▶ **Theorem 14.** *Model checking HOPDS without collapse against fixed LTL(F, X) and LTL(U) formulas is $(k-1)$-ExpTime-hard.*

Proof. The non-emptiness problem for HOPDS is $(k-1)$-ExpTime-complete [13]. This problem easily reduces to checking the fixed formula $G(\neg f)$, where f holds at all accepting states. Since this formula is both in LTL(F, X) and LTL(U), we are done. ◀

Next we study the expression complexity (fixed system). This is our main result of this section: already for a fixed order-k HOBPA, both LTL(F, X) and LTL(U) are k-ExpTime-hard.

▶ **Theorem 15.** *Model checking LTL(F, X) and LTL(U) against a fixed HOBPA without collapse is k-ExpTime-hard.*

Proof. (sketch) We take a k-ExpTime-hard language $\mathcal{L}$, and, by Theorem 5, its equivalent HOPDS with $s(n)$-bounded space work tape $\mathcal{P}$, for some polynomial $s(n)$. We shall construct a fixed HOBPA $\mathcal{P}'$ such that the language $\mathcal{L}$ is polynomial-time reducible to the LTL(F,X) model checking problem over $\mathcal{P}'$. We can similarly derive the desired lower bound for LTL(U) by "weakly" simulating the next operators with the until operator in the standard way.

We shall now give an intuition of the construction of $\mathcal{P}'$. Our HOBPA $\mathcal{P}'$ is an "over-approximation" of $\mathcal{P}$ in the sense that $\mathcal{P}'$ can do whatever actions $\mathcal{P}$ can do but also more. We will then use LTL(F,X) formulas to enforce correct simulations. This is of course due to the fact that $\mathcal{P}'$ lacks control states and work tape, and its definition should not depend on the input word to $\mathcal{P}$. We shall now elaborate more on how this can be implemented. Given a word $w = \alpha_1 \dots \alpha_n \in \Gamma^*$, we would like to determine if there is an accepting run of $\mathcal{P}$ on w, i.e., a sequence of configurations of the form $\langle p, \gamma, t, j\rangle$ starting with a starting configuration and ending with a final configuration. Here, p is a control state of $\mathcal{P}$, γ is a k-store, $t \in \Delta^{s(n)}$ is a tape content, and $1 \leq j \leq s(n)$ is the position of the tape head. We will represent each such configuration as the topmost symbols of a contiguous sequence of configurations of $\mathcal{P}'$. For example, suppose that the current configuration of $\mathcal{P}$ is $\langle p, \gamma, t, j\rangle$ where $top_1(\gamma) = a$. The HOBPA $\mathcal{P}'$ will start by having (p, a) as its topmost stack symbol. It will then keep modifing its topmost symbol to reflect the tape content t and the position j. This is done by guessing each individual tape cell content from left to right. At some point, $\mathcal{P}'$ will nondetermistically choose some rule of $\mathcal{P}$ to fire. We simulate this by first executing the stack operation and then guess some new state, which we put on top of the stack (this guess is needed because pop operations will destroy control state information). It will then continue by guessing next tape content in the same manner. This process can be repeated indefinitely, unless $\mathcal{P}'$ decides to go to a final (i.e. sink) state, in which $\mathcal{P}'$ will just loop forever. Given an input word $w = \alpha_1 \dots \alpha_n \in \Gamma^*$, we may force a correct simulation of $\mathcal{P}$ on w by $\mathcal{P}'$ using an LTL(F,X) formula. That is, we give a formula φ_w such that $w \in \mathcal{L}(\mathcal{P})$ iff $\mathcal{P}', c_0 \not\models \varphi_w$, where c_0 is an appropriate initial configuration of $\mathcal{P}'$ reflecting the initial state of $\mathcal{P}$. This can be done by first ensuring that each configuration of $\mathcal{P}$ in the simulation as a contiguous sequence of configurations of $\mathcal{P}'$ is valid. In particular, the guessed tape content (reflected by the topmost symbols in this sequence of configurations of $\mathcal{P}'$) must be of length $s(n)$ and has precisely one tape head, which can be easily expressed in LTL(F,X) using a single operator G and nestings of next operators of depth $s(n)$ (approximately). Recall that $s(n)$ is a polynomial function. Using the same technique, we also express that two representations of consecutive configurations of $\mathcal{P}$ in the simulation respect the transition relation of $\mathcal{P}$. Similarly, we enforce the initial configuration in the simulation and that some final configuration of $\mathcal{P}$ is reached. ◀

6 Future Work

There are several avenues of future work. E.g., we have no matching upper bound for the complexity of EF model checking. Walukiewicz has shown the problem to be PSpace-complete at order-1 [31]. However, his techniques do not easily extend to HOPDS owing to the subtleties of higher-order stacks. We may also study simpler logics such as LTL(F).

Acknowledgments. We thank Olivier Serre for interesting discussions and the anonymous referees for their helpful remarks. This work was partly supported by EPSRC (EP/F036361 and EP/E005039), and was done while the second author was a student at the School of Informatics, University of Edinburgh.

References

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

2 A. Bouajjani, P. Habermehl, and R. Mayr. Automatic verification of recursive procedures with one integer parameter. *Theor. Comput. Sci.* 295:85–106 (2003)

3 A. Bouajjani and A. Meyer. Symbolic Reachability Analysis of Higher-Order Context-Free Processes. In *FSTTCS*, p. 135–147, 2004.

4 L. Bozzelli. Complexity results on branching-time pushdown model checking. *Theor. Comput. Sci.*, 379(1-2):286–297, 2007.

5 C. Broadbent and L. Ong. On global model checking trees generated by higher-order recursion schemes. In *FOSSACS*, p. 107–121, 2009.

6 O. Burkart, D. Caucal, F. Moller, and B. Steffen. Verification on infinite structures. In *Handbook of Process Algebra*, Elsevier, 1999.

7 T. Cachat. Higher order pushdown automata, the caucal hierarchy of graphs and parity games. In *ICALP*, p. 556–569, 2003.

8 T. Cachat and I. Walukiewicz. The complexity of games on higher order pushdown automata. *CoRR*, abs/0705.0262, 2007.

9 A. Carayol. Regular sets of higher-order pushdown stacks. In *MFCS*, p. 168–179, 2005.

10 A. Carayol and S. Wöhrle. The caucal hierarchy of infinite graphs in terms of logic and higher-order pushdown automata. In *FSTTCS*, p. 112–123, 2003.

11 D. Caucal. On infinite terms having a decidable monadic theory. In *Proc. MFCS*, p. 165–176, 2002.

12 E. A. Emerson. Temporal and modal logic. In *Handbook of Theoretical Computer Science*, p. 995–1072, Elsevier, 1990.

13 J. Engelfriet. Iterated pushdown automata and complexity classes. In *STOC*, p. 365–373, 1983.

14 J. Esparza and J. Knoop. An automata-theoretic approach to interprocedural data-flow analysis. In *FoSSaCS*, p. 14–30, 1999.

15 E. M. Gurari. *An Introduction to the Theory of Computation*. W. H. Freeman & Co., New York, NY, USA, 1989.

16 M. Hague. *Global Model Checking Higher Order Pushdown Systems*. PhD thesis, Oxford University, 2009.

17 M. Hague, A. S. Murawski, C.-H. L. Ong, and O. Serre. Collapsible pushdown automata and recursion schemes. In *LICS*, p. 452–461, 2008.

18 T. Knapik, D. Niwinski, and P. Urzyczyn. Higher-order pushdown trees are easy. In *FoSSaCS*, p. 205–222, 2002.

19 T. Knapik, D. Niwinski, P. Urzyczyn, I. Walukiewicz. Unsafe Grammars and Panic Automata. In *ICALP*, p. 1450-1461, 2005.

20 N. Kobayashi. Model-checking higher-order functions. In *PPDP*, p. 25–36, 2009.

21 N. Kobayashi. Types and higher-order recursion schemes for verification of higher-order programs. In *POPL*, p. 416–428, 2009.

22 N. Kobayashi and C.-H. L. Ong. Complexity of model checking recursion schemes for fragments of the modal mu-calculus. In *ICALP*, p. 223–234, 2009.

23 N. Kobayashi and C.-H. L. Ong. A type system equivalent to the modal mu-calculus model checking of higher-order recursion schemes. In *LICS*, p. 179–188, 2009.

24 A. N. Maslov. Multilevel stack automata. *Probl. Inf. Transm.*, 15:1170–1174, 1976.

25 R. Mayr. Strict lower bounds for model checking BPA. *Electr. Notes Theor. Comput. Sci.*, 18, 1998.

26 C.-H. L. Ong. On model-checking trees generated by higher-order recursion schemes. In *LICS*, p. 81–90, 2006.

27 A. Seth. An Alternative Construction in Symbolic Reachability Analysis of Second Order Pushdown Systems. Int. J. Found. Comput. Sci. 19(4): 983-998, 2008.

28 W. Thomas. Constructing Infinite Graphs with a Decidable MSO-Theory. In *MFCS*, p. 113–124, 2003.

29 M. Vardi. A temporal fixpoint calculus. In *POPL*, p. 250-259, 1988.

30 I. Walukiewicz. Pushdown processes: Games and model checking. In *CAV*, p. 62–74, 1996.

31 I. Walukiewicz. Model checking CTL properties of pushdown systems. In *FSTTCS*, p. 127–138, 2000.