Unsafe Grammars and Panic Automata

Teodor Knapik[1], Damian Niwiński[2,⋆],
Paweł Urzyczyn[2,⋆⋆], and Igor Walukiewicz[3,⋆⋆⋆]

[1] Université de la Nouvelle Calédonie
knapik@univ--nc.nc
[2] Institute of Informatics, Warsaw University
{niwinski, urzy}@mimuw.edu.pl
[3] CNRS LaBRI, Université Bordeaux-1
igw@labri.fr

Abstract. We show that the problem of checking if an infinite tree generated by a higher-order grammar of level 2 (hyperalgebraic) satisfies a given μ-calculus formula (or, equivalently, if it is accepted by an alternating parity automaton) is decidable, actually 2-EXPTIME-complete. Consequently, the monadic second-order theory of any hyperalgebraic tree is decidable, so that the safety restriction can be removed from our previous decidability result. The last result has been independently obtained by Aehlig, de Miranda and Ong. Our proof goes *via* a characterization of possibly unsafe second-order grammars by a new variant of higher-order pushdown automata, which we call *panic automata*. In addition to the standard pop_1 and pop_2 operations, these automata have an option of a destructive move called *panic*. The model-checking problem is then reduced to the problem of deciding the winner in a parity game over a suitable 2nd order pushdown system.

1 Introduction

Context-free tree grammars constitute the basic level in an infinite hierarchy of higher-order grammars introduced by W. Damm [8] (built on the earlier ideas of [11]). Courcelle [6] proved decidability of the monadic second-order (MSO) theory of any tree generated by an *algebraic* (context-free) tree grammar. Later Knapik *et al* [13, 14] attempted to extend this decidability result to all levels of the Damm hierarchy. This has been achieved partially, namely with an additional syntactic restriction imposed on the grammars, called *safety*: the MSO theory of any tree generated by a safe grammar of level n is decidable.

Higher-order grammars can be seen as program schemes, where functions can take higher-order arguments. The tree generated by such a grammar describes completely the semantics of the program scheme. Thus decidability of

⋆ Partly supported by KBN Grant 4 T11C 042 25.
⋆⋆ Partly supported by KBN Grant 3 T11C 002 27.
⋆⋆⋆ The 2nd and the 4th author were also supported by the EC Research Training Network *Games*.

L. Caires et al. (Eds.): ICALP 2005, LNCS 3580, pp. 1450–1461, 2005.

the MSO theory of such a tree implies decidability for a large class of properties of behaviours of higher-order program schemes.

The safety requirement, roughly speaking, prevents the use of individual parameters in the functional arguments of higher-order functions. The concept of safe (tree) grammars has been further justified by a characterization in terms of higher-order pushdown automata originally introduced by Maslov [16]: the trees generated by safe higher-order grammars of level n coincide with the trees recognized (in suitable sense) by pushdown automata of level n [14]. See [5, 7] for another characterization of this hierarchy.

Monadic second-order logic is extremely succinct, which yields the non-elementary complexity of the MSO theories, even for regular trees. However, over trees, the MSO logic has the same expressive power as the *μ-calculus*, which is much better tractable algorithmically. Cachat [3] showed that the model-checking problem for the μ-calculus and the trees recognized by the pushdown automata of level n is in n-EXPTIME, actually n-EXPTIME-complete [4].

In this paper we show that for the level 2 (hyperalgebraic), the model checking problem remains decidable, actually 2-EXPTIME-complete, also for grammars *without* safety restriction.

To this end, we first find an automata-theoretic counterpart of unsafe hyperalgebraic grammars, which is an extension of second-order pushdown automata by a new destructive operation that we call *panic*. This operation allows us to simulate the change of environment needed to evaluate parameters of unsafe productions. We further introduce 2nd order pushdown systems with panic (equipped with alternation and ranks), which can be viewed as a generalization of pushdown systems studied in [20, 3]. The model-checking problem then reduces to the problem of deciding the winner in a parity game over such a system. The key step is a reduction of this game to a game over a 2nd order pushdown system *without* panic, for which a 2-EXPTIMEprocedure is already known [3].

An immediate consequence of our result is that the model checking problem for the hyperalgebraic grammars is decidable also for the monadic second-order logic, consequently the MSO theory of any hyperalgebraic tree is decidable. This result has been recently independently obtained by Aehlig, de Miranda and Ong [2], by a different proof, based on transformations of infinite lambda terms. Compared to [2], the present paper has two new elements: (1) it gives a characterization of unsafe grammars by panic automata, (2) the actual proof yields an optimal decision procedure for the μ-calculus model checking.

At present we do not know if the safety requirement really restricts the generating power of the tree grammars. Recently, Aehlig, de Miranda and Ong have studied the safety restriction [1]. They have shown that it is inessential for the *word* grammars of level 2, where a grammar, as usual, can generate a set of words. To simulate a non-safe grammar by a safe one, they use nondeterminism in an essential way, hence their result is not directly applicable for trees.

Due to space limitations, many arguments are omitted or sketchy; they will appear in the full paper (see [15] for a preliminary version).

2 Trees

Types, terms, and trees. We fix the set of simple types τ constructed from a unique *basic* type $\mathbf{0}$, by the rules $\tau ::= \mathbf{0} \mid (\tau_1 \to \tau_2)$. The level of a type is defined by $\ell(\mathbf{0}) = 0$, and $\ell(\tau_1 \to \tau_2) = \max(1 + \ell(\tau_1), \ell(\tau_2))$. Thus $\mathbf{0}$ is the only type of level 0 and each type of level 1 is of the form $\mathbf{0}^n \to \mathbf{0}$ for some $n > 0$ (which abbreviates $(\mathbf{0} \to (\mathbf{0} \to (\cdots(\mathbf{0} \to \mathbf{0})\ldots)))$, with $n+1$ occurrences of $\mathbf{0}$).

A *typed alphabet* is a set Γ of symbols, each γ in Γ given with its type, $\gamma : \tau$. We inductively extend Γ to the set $T(\Gamma)$ of *(applicative) terms* over Γ; if $t : \tau_1 \to \tau_2$ and $s : \tau_1$ then $(ts) : \tau_2$. As usual we abbreviate $(\cdots((t_0 t_1)t_2)\cdots)t_n$ by $t_0 t_1 \ldots t_n$. For terms $t, t_1, \ldots, t_m$, and symbols $z_1, \ldots, z_m$, of appropriate types, term $t[z_1{:=}t_1, \ldots, z_k{:=}t_k]$ results from simultaneous replacement in t of z_i by t_i.

The set of natural numbers is denoted by ω. A *tree* (over a set X) is any nonempty prefix-closed subset T of the free monoid X^*, with ε as the *root*. If $u \in T$, $x \in X$, and $ux \in T$, then ux is a *successor* of u in T.

Now let Σ be a *signature*, i.e., a typed alphabet of symbols of level ≤ 1. A *Σ-tree* is a mapping $t\colon \mathit{dom}\ t \to \Sigma$, where $\mathit{dom}\ t \subseteq \omega^*$ is a tree, and if $t(w)$ is a symbol of type $\mathbf{0}^k \to \mathbf{0}$ then w has exactly k successors, $w1, \ldots, wk$ (hence w is a leaf if $t(w) : \mathbf{0}$). The set of Σ-trees is written $T^\infty(\Sigma)$.

A *limit* of a sequence $t_0, t_1, \ldots$ of Σ-trees is defined, provided that for any k, there is $m = m(k)$, such that $t_n{\restriction}k$ and $t_{n'}{\restriction}k$ coincide, for all $n, n' \geq m(k)$ (where, in general, $t{\restriction}k$ is restriction of t to the set $\{w \in \mathit{dom}\ t : |w| \leq k\}$). Then $\lim t_n \in T^\infty(\Sigma)$ is just the set-theoretical union of the functions $t_n{\restriction}m(n)$.

Grammars. We fix an infinite typed alphabet of *variables* (or *parameters*), $\mathcal{X}$. A *grammar* is a tuple $\mathcal{G} = (\Sigma, N, \mathcal{S}, E)$, where Σ is a *signature*, N is a finite typed alphabet of *nonterminals*[1], $\mathcal{S} \in N$ is a *start symbol* of type $\mathbf{0}$, and E is a set of productions of the form

$$\mathcal{F} z_1 \ldots z_m \Rightarrow w$$

where $\mathcal{F} : \tau_1 \to \tau_2 \cdots \to \tau_m \to \mathbf{0}$ is a nonterminal, z_i a variable of type τ_i, and w an applicative term in $T(\Sigma \cup N \cup \{z_1 \ldots z_m\})$ of type $\mathbf{0}$. The *level* of a grammar is the highest level of its nonterminals. Since we are interested in grammars as generators of (single) Σ-trees, we assume that for each $\mathcal{F}$, there is exactly one production with $\mathcal{F}$ on the left-hand side.

The single-step reduction relation $\to_{\mathcal{G}}$ between terms over $\Sigma \cup N$ is defined inductively by the following clauses.

1. $\mathcal{F} t_1 \ldots t_k \to_{\mathcal{G}} t[z_1{:=}t_1, \ldots, z_k{:=}t_k]$ if there is a production $\mathcal{F} z_1 \ldots z_k \Rightarrow t$ with $z_i : \rho_i$, and $t_i \in T(\Sigma \cup N)$, where $t_i : \rho_i$, for $i = 1, \ldots, k$.
2. If $t \to_{\mathcal{G}} t'$ then $(st) \to_{\mathcal{G}} (st')$ and $(tq) \to_{\mathcal{G}} (t'q)$, whenever the expressions in question are applicative terms.

[1] Without loss of generality, we assume that the types of nonterminals are *homogeneous*, i.e., $\mathbf{0}$ or $\tau_1 \to \cdots \to \tau_n \to \mathbf{0}$, where if each τ_i is homogeneous and $\ell(\tau_1) \geq \ell(\tau_2) \geq \ldots \geq \ell(\tau_n)$.

That is, $t \to_{\mathcal{G}} t'$ whenever t' is obtained from t by replacing some occurrence of a nonterminal F by the right-hand side of the appropriate production in which all parameters are in turn replaced by the actual arguments of F.

In order to define the result of an infinite derivation, we extend Σ to $\Sigma^\perp = \Sigma \cup \{\perp\}$, with $\perp : \mathbf{0}$. With any term t over $\Sigma \cup N$, we inductively associate an expression $t^\perp$ over $\Sigma^\perp$, by setting $f^\perp = f$ (for $f \in \Sigma$), $X^\perp = \perp$ (for $X \in N$), and $(st)^\perp = (s^\perp r^\perp)$ whenever $s^\perp \neq \perp$, otherwise $(st)^\perp = \perp$. Then, we define relation $t \twoheadrightarrow^\infty_{\mathcal{G}} t'$, where t is a term in $T(\Sigma \cup N)$ and t' a tree in $T^\infty(\Sigma^\perp)$, by

- t' is finite, and there is reduction sequence $t = t_0 \to_{\mathcal{G}} \ldots \to_{\mathcal{G}} t_n = t'$, or
- t' is infinite, and there is an infinite reduction sequence $t = t_0 \to_{\mathcal{G}} t_1 \to_{\mathcal{G}} \ldots$ such that $t' = \lim t_n^\perp$.

To define a unique tree produced by the grammar, we recall a standard *approximation ordering* on $T^\infty(\Sigma^\perp)$: $t' \sqsubseteq t$ if $dom\ t' \subseteq dom\ t$ and, for each $w \in dom\ t'$, $t'(w) = t(w)$ or $t'(w) = \perp$. Then the tree generated by $\mathcal{G}$ is defined by

$$[\![\mathcal{G}]\!] = \sup\{t \in T^\infty(\Sigma^\perp) : S \twoheadrightarrow^\infty_{\mathcal{G}} t\}$$

It is easy to see that, by the Church-Rosser property of our grammar, the above set is directed, and hence $[\![\mathcal{G}]\!]$ is well defined since $T^\infty(\Sigma^\perp)$ with the approximation ordering is a cpo.

In this paper we only study grammars of level 2, which we call *hyperalgebraic*, as they constitute the next level above the algebraic (context-free) grammars.

Parity games. A *parity game* is a perfect information game of possibly infinite duration played by two players, say Eve and Adam. We present it as a tuple $(V_\exists, V_\forall, E, p_1, \Omega)$, where $V_\exists$ and $V_\forall$ are (disjoint) sets of positions of Eve and Adam, respectively, $E \subseteq V \times V$ is the relation of possible moves, with $V = V_\exists \cup V_\forall$, $p_1 \in V$ is a designated initial position, and $\Omega : V \to \omega$ is the ranking function.

The players start a play in the position p_1 and then move the token according to relation E (always to a successor of the current position), thus forming a path in the graph (V, E). The move is selected by Eve or Adam, depending on who is the owner of the current position. If a player cannot move, she/he looses. Otherwise, the result of the play is an infinite path in the graph, $v_0, v_1, v_2, \ldots$ Eve wins the play if $\limsup_{n\to\infty} \Omega(v_n)$, is even, otherwise Adam wins.

Parity games, introduced by Emerson and Jutla [9, 17], have been recognized as a combinatorial essence of many model checking problems. A crucial property is the *positional determinacy*: any position is winning for one of the players, and moreover a winning strategy of player θ can be made *positional*, i.e., represented by a (partial) function $\sigma : V_\theta \to V$.[2] We say simply that Eve *wins* the game $\mathcal{G}$ if

[2] Positional strategy σ is *winning* for θ if every play $p_1 = q_1, q_2, \ldots$ is won by θ, provided that $q_\ell \in V_\exists$ follows $q_{\ell+1} = \sigma(q_\ell)$.

she has a winning strategy, the similar for Adam. We refer the reader to [19, 12] for an introduction to parity games.

Model checking. The model checking problem in our consideration is to verify if the tree generated by a given grammar $\mathcal{G}$ satisfies a property φ expressed in some logical language, in symbols $[\![\mathcal{G}]\!] \models \varphi$. The most expressive logics considered in literature are the *monadic second-order logic* (MSO) or the *μ-calculus*; both have the same expressive power over Σ-trees [18]. In this paper we avoid a logical machinery by using an equivalent formalism of *alternating parity automata* (defined below). A polynomial-time translation from the μ-calculus to alternating automata, $\varphi \mapsto A_\varphi$, is known [9], such that, for any tree t, $t \models \varphi$ iff t is recognized by A_φ.

An alternating parity tree automaton over signature Σ can be presented as a tuple

$$\mathcal{B} = \langle \Sigma, Q_\exists, Q_\forall, q_1, \delta, \Omega \rangle$$

where $Q_\exists \cup Q_\forall = Q$ is a finite set of states with the initial state q_1, $\Omega : Q \to \omega$ is a ranking function, and δ is a set of transitions of the form $q \to f(q_1, \ldots, q_k)$, where $q, q_1, \ldots, q_k \in Q$ and $f \in \Sigma$ with type $f : \mathbf{0}^k \to \mathbf{0}$.

The acceptance of a tree $t \in T^\infty(\Sigma)$ by the automaton $\mathcal{B}$ can be presented by a suitable parity game. We first define a the computation tree, $r : dom\, r \to Q$ with $dom\, r \subseteq (\omega \times \omega)^*$, such that, for any $u \in dom\, r$, the projection $u\!\downarrow_1$ on the first component is a node in $dom\, t$. We let $r(\epsilon) = q_1$, and whenever $r(u) = q$ and $t(u\!\downarrow_1) = f$, then, for any transition of the form $q \to f(q_1, \ldots, q_k)$, the node u has a successor $u(i, j)$ with $r(u(i, j)) = q_i$, for some j, and $i = 1, \ldots, k$. Now consider a parity game $Game(\mathcal{B}, t)$ with $V = dom\, r$ partitioned such that $u \in V_\exists$ iff $r(u) \in Q_\exists$, the initial configuration ε, and $\Omega(u) = \Omega(r(u))$. We let $\mathcal{B}$ *accept* the tree t iff Eva wins this game.

The model checking problem addressed in this paper will be the following.

Problem 1. Given a 2nd order grammar $\mathcal{G}$ and an alternating parity tree automaton $\mathcal{B}$. Does $\mathcal{B}$ accept $[\![\mathcal{G}]\!]$?

3 Panic Automata

Classically the content of a pushdown store is just a word over the pushdown alphabet. For our purpose, it is convenient to consider pushdown symbols with "time stamps" (sort of). We let a *level* 1 *pushdown store* (or a 1-*pds* or 1-stack) over an alphabet Γ be a non-empty word $a_1 \ldots a_k$ over $\Gamma \times \omega$. A *level* 2 *pds* (2-*pds*, 2-stack) is a non-empty sequence $s_1 \ldots s_l$ of 1-pds's, which may also be written as $[s_1][s_2] \ldots [s_l]$ or as $s'[s_l]$, where s' stands for $[s_1][s_2] \ldots [s_{l-1}]$. The 1-stack s_i is called the *i-th row* of s. We assume that push-down stores grows to the *right*, so that, for instance, the item (a, m) is on top of the 2-pds $s'[w(a, m)]$. By $top(s)$ we denote the topmost Γ-symbol of s, i.e., $top(s'[w(a, m)]) = a$.

The following operations are possible on level 2 push-down stores.

$$\begin{aligned} push_1\langle a\rangle([s_1][s_2]\dots[s_l][w]) &= [s_1][s_2]\dots[s_l][w(a,l)] \\ pop_1(\alpha[w\xi]) &= \alpha[w] \\ push_2(\alpha[w]) &= \alpha[w][w] \\ pop_2(\alpha[v][w]) &= \alpha[v] \\ panic([s_1][s_2]\dots[s_m]\dots[s_l][w(a,m)]) &= [s_1][s_2]\dots[s_m] \\ skip(s) &= s \end{aligned}$$

The operation pop_2 (resp. pop_1) is undefined on a 2-stack s if it contains only one row (resp. the top row of s has only one element).

Let $\perp$ be a symbol in Γ. It is easy to see that if a 2-pds $[s_1]\dots[s_m]$ is generated from $[(\perp,0)]$ by the above operations, and $s_i = (a_{i,1}, m_{i,1})\dots(a_{i,k_i}, m_{i,k_i})$ then $m_{i,j} \le i-1$, and $j \le j'$ follows $m_{i,j} \le m_{i,j'}$. Intuitively, whenever a new symbol $a \in \Gamma$ is placed on the top of the stack, the second component registers the number of the stack row which is directly below the current top row. Later the symbol can be duplicated several times by subsequent executions of $push_2$, but the second component keeps record of the level when it has first appeared in the stack. The *panic* operation returns to the 2-stack previous to the first appearance of the actual top symbol.

Now let Σ be a signature, and let $\Sigma_r \subseteq \Sigma$ be the set of symbols of type $\mathbf{0}^r \to \mathbf{0}$. A *panic automaton* is defined as a tuple

$$\mathcal{A} = \langle\, \Sigma, Q, \Gamma, q_1, \delta, \perp \,\rangle,$$

where Q is a finite set of *states*, with an *initial state* q_1, Γ is a stack alphabet, with a distinguished *bottom symbol* $\perp$, and $\delta : Q \times \Gamma \to \mathcal{I}$ is a *transition function*, where $\mathcal{I}$ is the set of possible instructions,

$$\mathcal{I} \subseteq Op^{\Gamma} \times Q \cup \bigcup_r \Sigma_r \times Q^r,$$

where $Op^{\Gamma} = \{push_1\langle a\rangle : a \in \Gamma\} \cup \{pop_1, push_2, pop_2, panic, skip\}$.

A *configuration* of an automaton $\mathcal{A}$ as above is a pair (q, s), where $q \in Q$, and s is a 2-stack (over Γ). The *initial configuration* is $(q_1, [(\perp, 0)])$. We define the relation $\to_{\mathcal{A}}$ on configurations as follows. Let $s = [s_1]\dots[s_l][w(a,m)]$.

1. If $\delta(q,a) = \langle \alpha, q'\rangle$ with $\alpha \in Op^{\Gamma}$ then $\langle q, s\rangle \to_{\mathcal{A}} \langle q', \alpha(s)\rangle$.
2. If $\delta(q,a) = \langle f, p_1, \dots, p_r\rangle$ then $\langle q, s\rangle \to_{\mathcal{A}} \langle p_i, s\rangle$, for all $i = 1, \dots, r$.

In the first case we write $\to^{\circ}_{\mathcal{A}}$ instead of $\to_{\mathcal{A}}$. The symbol $\twoheadrightarrow^{\circ}_{\mathcal{A}}$ stands for the reflexive and transitive closure of $\to^{\circ}_{\mathcal{A}}$.

Let $t : dom\, t \to \Sigma$ be a Σ-tree. A *partial run* of $\mathcal{A}$ on t is a partial function ϱ from an initial segment of $dom\, t$ to the set of all configurations, such that if $\varrho(w) = \langle q, s\rangle$, for some $w \in dom\, t$, then $\delta(q,a) = (f, p_1, \dots, p_r)$, where $f = t(w)$ and $a = top(s)$. In addition $\langle p_i, s\rangle \twoheadrightarrow^{\circ}_{\mathcal{A}} \varrho(wi)$, for each $i = 1, \dots, r$ when $\varrho(wi)$ is defined.

If a partial run is a total function over $dom\, t$, we call it a *run*. As our automaton is deterministic, there can be at most one tree over which $\mathcal{A}$ has a run. This is the *tree recognized by* $\mathcal{A}$.

4 Automata vs Grammars

Theorem 4.1. *For any panic automaton $\mathcal{A}$, one can construct a hyperalgebraic grammar $\mathcal{G}_\mathcal{A}$, such that if the automaton recognizes a tree t in $T^\infty(\Sigma)$ then $t = [\![\mathcal{G}_\mathcal{A}]\!]$. Conversely, for a hyperalgebraic grammar $\mathcal{G}$, one can construct a panic automaton $\mathcal{A}_\mathcal{G}$ such that if the grammars generates a tree in $T^\infty(\Sigma)$, this is the (unique) tree recognized by $\mathcal{A}_\mathcal{G}$. Both constructions can be realized in polynomial time.*

Here we only sketch the direction $\mathcal{G} \mapsto \mathcal{A}_\mathcal{G}$, which is essential for the upper bound result.

Let $\circ_1, \circ_2, \ldots$ be fresh identifiers of type $\mathbf{0}$, which we call *holes*. The pushdown alphabet Γ of $\mathcal{A}_\mathcal{G}$ consists of subterms of the right-hand sides of the productions of $\mathcal{G}$, possibly applied to some holes, so that the result is of type $\mathbf{0}$. More precisely, let $u = Ft_1 \ldots t_d$ be such a subterm, where F is an operator (variable or nonterminal) of type $\tau_1 \to \cdots \to \tau_d \to \mathbf{0}^k \to \mathbf{0}$. Then $u \circ_1 \ldots \circ_k \in \Gamma$. In particular, if $u : \mathbf{0}$ then simply $u \in \Gamma$.

The holes $\circ_1, \ldots, \circ_k$ represent "missing arguments" of the operator F. Since holes are new identifiers, one can safely identify $Ft_1 \ldots t_d \circ_1 \ldots \circ_k$ with $Ft_1 \ldots t_d$ if this is convenient.

The idea of our simulation is that the top of pds (an expression u) represents a variable-free expression u' occurring in a derivation of $\mathcal{G}$. Since the pds alphabet must be finite, the term u can only be an "approximation" of u'. This approximation is "evaluated" to yield an approximate representation of the next step of reduction. The contents of the pds represents an environment in which the evaluation takes place. The environment is searched if one needs to find the meaning of a variable, or to find a missing argument.

The bottom pds symbol is $\mathcal{S}$, the initial nonterminal. This is our approximation in the first step of reduction. The automaton then works in phases, each phase beginning and ending in the distinguished state q_1. We define automaton informally, by describing the possible behaviour in a phase, beginning with a configuration (q_1, s).

T1 Let $top(s) = \mathcal{F}u_1 \ldots u_n$, where $\mathcal{F}$ is a nonterminal, and let the corresponding production be $\mathcal{F}x_1 \ldots x_n \Rightarrow u$. The automaton executes the instruction $\delta(q_1, \mathcal{F}u_1 \ldots u_n) = (push_1\langle\, u\,\rangle, q_1)$, so that our next approximation is u.

T2 If $top(s) = ft_1 \ldots t_r$ where f is a terminal, then the automaton executes the instruction $\delta(q_1, ft_1 \ldots t_r) = (f, p_1, \ldots, p_r)$, followed (at the i-th branch of the run) by a pop_1 and $push_1\langle\, t_i\,\rangle$. Thus the next approximation at the i-th branch is t_i. (If f is a constant, there is no further step.)

T3 Let $top(s) = x$, where x is an (ordinary) variable of type $\mathbf{0}$. To "evaluate" x, the automaton restores the environment where x was defined. It executes

pop_1 and inspects the new top symbol. It should be of the form $\mathcal{F}t_1 \dots t_e$, where $\mathcal{F}$ is a nonterminal. In addition, the variable x should be one of the formal parameters of $\mathcal{F}$, say, the j-th one. The next approximation is t_j, and it should be evaluated in the present environment (that of the caller). Another pop_1 is now executed, followed by a $push_1\langle\, t_j \,\rangle$, and the machine returns to state q_1.

T4 Let $top(s) = \varphi u_1 \dots u_h$ where φ is a variable of type $\mathbf{0}^h \to \mathbf{0}$. If we now executed a pop_1 as above then the information about the actual parameters $u_1, \dots, u_h$ would be lost. Instead, a $push_2$ is executed, followed by a pop_1. As in the previous case, the top of the stack should be of the form $\mathcal{F}t_1 \dots t_e$ and φ should be one of the formal parameters of $\mathcal{F}$, say, the j-th one. However now the new approximant t_j is an expression of a functional type and we actually place $t_j \circ_1 \dots \circ_h$ on the pds rather than t_j.

T5 The last case is when $top(s)$ is a hole, say $\circ_i$. The automaton gets now into a panic. After the panic move the top of the pds should be $\psi v_1 \dots v_l$ for some variable ψ. The new approximation is v_i. We execute in order pop_1 and $push_1\langle\, v_i \,\rangle$ and return to state q_1.

To explain the last case let us first observe that holes (missing arguments) are created when we attempt to evaluate a function variable (the fourth case). Holes correspond to the arguments (actual parameters) of this function variable that were "left behind" for a while. Later, a hole is found at top of the pds when we need to evaluate such a missing argument of an operator. In order to do so, we must restore the situation from the time of the call. The crucial thing is that the "time stamp" of the topmost item points out to exactly the moment in the past we need, namely to the stage when the hole was created.

5 Pushdown Systems

By now, we have considered panic automata as tree acceptors, with the aim to characterize hyperalgebraic grammars. For the model-checking applications, it is convenient to use a related concept of pushdown systems which do not take inputs, but are equipped with ranks and alternation (c.f. [3, 20]).

A *2nd order pushdown system with panic* (or 2-PPDS, for short) is a tuple $\mathcal{C} = (P, P_\exists, P_\forall, \Gamma, p_1, \Delta, \Omega)$ where P is a finite set of control locations partitioned into $P_\exists$ and $P_\forall$, Γ is a finite store alphabet, $p_1 \in P$ is the initial location, $\Omega : P \to \omega$ is the rank function, and $\Delta \subseteq P \times \Gamma \times P \times Op^\Gamma$ is the finite set of (unlabeled) transition rules (where Op^Γ is as in Sect. 3). A transition $(p, a, p', \alpha) \in \Delta$ is often written by $p, a \to_\Delta p', \alpha$.

A *configuration* of a 2-PPDS $\mathcal{C}$ is a pair (p, s) where $p \in P$ and s is a 2-stack over Γ. A 2-PPDS $\mathcal{C}$ induces a parity game $Game(\mathcal{C}) = (V_\exists, V_\forall, E, (p_1, [\bot, 0]), \Omega)$, where

- $V_\exists$ is the set of configurations of $\mathcal{C}$ with a location from $P_\exists$, similarly for $V_\forall$;
- $(p, s)E(p', s')$ iff $\exists (p, a, p', \alpha) \in \Delta,\ top(s) = a$ and $s' = \alpha(s)$;
- $\Omega(p, s) = \Omega(p)$.

Problem 2. Given a 2-PPDS $\mathcal{C}$, decide if Eva wins the game $Game(\mathcal{C})$.

Proposition 5.2. *Problem 1 and Problem 2 are polynomial-time equivalent.*

Proof: *(Sketch)* By Theorem 4.1, we reduce Problem 1 to the question if a tree recognized by a panic automaton $\mathcal{A}$ is accepted by an alternating tree automaton $\mathcal{B}$. Then a suitable 2-PPDS $\mathcal{C}$ is obtained by a standard product construction, such that Eve wins $Game(\mathcal{C})$ if and only if she wins $Game(\mathcal{B}, t_{\mathcal{A}})$

The converse transformation of Problem 2 to Problem 1 is also easy, after introduction of a suitable functional signature Σ (depending on given $\mathcal{C}$). □

Before proceeding to the proof of our main result, we show a useful transformation of 2-PPDS's, which will allow us to control the dynamics of ranks by the topmost symbols on the pushdown store.

Let $Tr(\mathcal{C})$ be the tree obtained by unfolding the graph of $Game(\mathcal{C})$. That is, the root of $Tr(\mathcal{C})$ is labeled by the initial configuration, and a node labeled by (p, s) has a successor labeled (p', s'), whenever $(p, s)E(p', s')$. Note that a node of the tree determines a computation up to this node.

Definition 5.3 *Let a node v of $Tr(\mathcal{C})$ be labeled with $\langle q, s\rangle$ such that $panic(s)$ is defined. Let v' be the closest to v ancestor of v labeled with $\langle q', panic(s)\rangle$, for some q'. We call v' the* panic ancestor of v. *The* panic rank of v *is the maximal rank of a state occurring between the panic ancestor of v and v, including the rank of v and excluding the rank of the panic ancestor of v.*

A 2-PPDS $\mathcal{C}$ is rank-aware *iff there exists a function $Rank : \Gamma \to \mathrm{Rg}(\Omega)$ such that the panic rank of every node v of $Tr(\mathcal{C})$ labeled (q, s) is equal to $Rank(top(s))$. That is, the panic rank is determined by the top of the stack.*

Lemma 5.4. *For every 2-PPDS $\mathcal{C}$, one can construct in polynomial time a rank-aware 2-PPDS $\mathcal{C}'$, such that Eve wins $Game(\mathcal{C}')$ iff she wins in $Game(\mathcal{C})$.*

Proof: *(Idea)* The stack alphabet of $\mathcal{C}'$ is defined as $\Gamma' = \Gamma \times \{0, \ldots, d\}^2$, where d is the highest rank in $\mathrm{Rg}(\Omega)$. The construction ensures that if (a, m_p, m_l) is currently on top of the stack, then m_p is the panic rank of the node, and m_l is the highest rank of a state seen since the creation of the current top row.

The construction of transitions of $\mathcal{C}'$ goes naturally case by case. The correctness proof is straightforward, but tedious. □

6 Deciding the Winner in $Game(\mathcal{C})$

A 2*nd order pushdown system* (or 2-PDS, for short) is like a 2-PPDS defined in section 5, but *without* panic, that is, $\Delta \subseteq P \times \Gamma \times P \times (Op^{\Gamma} - \{panic\})$. Then a level 1 pushdown store can be viewed just as a word over Γ (not $\Gamma \times \omega$). The concepts of 2-*pds*, configuration, and $Game(\mathcal{C})$ are simplified accordingly.

Games over such pushdown systems have been considered by Engelfriet [10], Cachat [3], and others.

Lemma 6.5 ([10, 3, 4]). *It is decidable if Eve wins $Game(\mathcal{C})$ for a given 2nd-order pushdown system $\mathcal{C} = (P, P_\exists, P_\forall, \Gamma, p_1, \Delta, \Omega)$. There is an algorithm working in time*[3] $2^{|\Gamma| 2^{\mathcal{O}(|P|)}}$*, moreover the problem is 2-*EXPTIME*-complete.*

We now transform a rank-aware 2nd order pushdown system with panic $\mathcal{C} = (P, P_\exists, P_\forall, \Gamma, p_1, \Delta, \Omega)$ into a 2nd order pushdown system without panic $\mathcal{C}' = (P', P'_\exists, P'_\forall, \Gamma', p'_1, \Delta', \Omega')$, such that Eve wins $Game(\mathcal{C})$ iff she wins $Game(\mathcal{C}')$.

For technical convenience, we assume that a transition $p, a \to_\Delta p', \alpha$ with $\alpha \neq \mathit{skip}$ is possible only for $p \in P_\exists$. Clearly, we can always achieve this property by duplicating locations and using *skip* operation.

Let $\mathit{Rank} : \Gamma \to \mathrm{Rg}(\Omega) = \{0, \ldots, d\}$ be the function of Definition 5.3. We introduce an ordering on ranks: $m \preccurlyeq n$ iff $(-1)^m \cdot m \leq (-1)^n \cdot n$ (e.g., for $d = 6$, this yields an ordering $5, 3, 1, 0, 2, 4, 6$).

We define the set of *returns* as

$$\mathit{Ret} = P \dot{\to} \{0, \ldots, d\} \ .$$

The intention is that a partial function $R \in \mathit{Ret}$ assigns to a location p the worst, in $\preccurlyeq$-order, panic rank still acceptable for the panic moves *ending* in position p. We let

$$\begin{aligned} P'_\exists &= P_\exists \cup \{\bot\}, \\ P'_\forall &= P_\forall \cup \{p_{push}, p_{ver}, p_{rank=i} : p \in P, i \in \{0, \ldots, d\}\} \cup \{\top\} \cup \{ver\} \cup \mathit{Aux} \\ \Gamma' &= \mathit{Ret} \cup (\Gamma \times \mathit{Ret}). \end{aligned}$$

The rank function Ω' is 0 for all the locations except for

$$\Omega'(p) = \Omega(p), \quad \Omega'(p_{rank=i}) = i; \quad \text{for all } p \in P.$$

The transition rules Δ' are defined by the following clauses.

- If $p, a \to_\Delta p', \mathit{skip}$ then $p, (a, R) \to_{\Delta'} p', \mathit{skip}$, for all $R \in \mathit{Ret}$.
- If $p, a \to_\Delta p', \mathit{push}_1\langle a \rangle$ then $p, (a, R) \to_{\Delta'} p'_{ver}, \mathit{push}_1\langle (a', R') \rangle$, for all $R, R' \in \mathit{Ret}$. Additionally we have $p'_{ver}, (a', R') \to_{\Delta'} \mathit{ver}, \mathit{skip}$, and $p'_{ver}, (a', R') \to_{\Delta'} p', \mathit{skip}$.
- From the location *ver* the moves are defined in such a way that Eve wins iff R in the top letter of the top-most 1-stack is the same as R at the top of 1-stack just below the top-most 1-stack.
- If $p, a \to_\Delta p', \mathit{push}_2$ then $p, (a, R) \to_{\Delta'} p'_{push}, \mathit{push}_1\langle R' \rangle$, for all $R, R' \in \mathit{Ret}$. Additionally we put in Δ' the rules[4] $p'_{push}, R' \to_{\Delta'} p', \mathit{push}_2; \mathit{pop}_1$, as well as

[3] The single exponential dependence on $|\Gamma|$ is not made explicit in these references, but it follows from the analysis of level 1 pushdown systems in [20].

[4] Notation like $p, a \to_{\Delta'} p', \alpha_1; \alpha_2$ clearly abbreviates two rules.

$p'_{push}, R' \to_{\Delta'} p''_{rank=R'(p'')}, pop_1$, and $p''_{rank=R'(p'')}, (a, R) \to_{\Delta'} p'', skip$, for all p'' such that $R'(p'')$ is defined.
- If $p, a \to_\Delta p', pop_1$ then $p, (a, R) \to_{\Delta'} p', pop_1$, for all $R \in Ret$.
- If $p, a \to_\Delta p', pop_2$ then $p, (a, R) \to_{\Delta'} p', pop_2; pop_1$, for all $R \in Ret$.
- If $p, a \to_\Delta p', panic$ then $p, (a, R) \to_{\Delta'} \top, skip$ if $Rank(a) \succcurlyeq R(p')$, and we have $p, (a, R) \to_{\Delta'} \bot, skip$ otherwise, i.e., when $Rank(a) \prec R(p')$ or $R(p')$ is undefined.
- There are no transitions from states $\top$ and $\bot$. This implies that Eve wins in $\top$ and loses in $\bot$.

Proposition 6.6. *Eve wins in $Game(\mathcal{C}')$ iff she wins in $Game(\mathcal{C})$.*

Proof: *(Idea)* The construction guarantees that a 2-pds in a reachable configuration of $\mathcal{C}'$ has a form $v = [w_1 R_1] \dots [w_k R_k][w_{k+1}]$, where $w_i \in (\Gamma \times Ret)^*$ and $R_i \in Ret$. We say that v *represents* a 2-pds $s = [s_1] \dots [s_{k+1}]$ of $\mathcal{C}$ if, for all i, $s_i \downarrow_1 = w_i \downarrow_1$, and if $s_i \downarrow_2 = \ell$ then $w_i \downarrow_2 = R_\ell$. Note that a possibly infinite domain of "time stamps" (in $\mathcal{C}$) is represented here by the finite set Ret. Both players can ensure that the 2-stacks reachable in $Game(\mathcal{C}')$ represent the 2-stacks of $\mathcal{C}$. Together with the mechanism of returns, this allows them to transfer the strategies from $Game(\mathcal{C})$ to $Game(\mathcal{C}')$. □

We are now ready to state our main result.

Theorem 6.7. *The problem of checking if Eva wins $Game(\mathcal{C})$, for a given 2nd order pushdown system with panic $\mathcal{C}$, is 2-*Exptime*-complete (Problem 2). Consequently, so is the problem of checking if an alternating parity tree automaton $\mathcal{B}$ accepts the tree $[\![\mathcal{G}]\!]$ generated by an hyperalgebraic grammar $\mathcal{G}$ (Problem 1).*

Proof: The 2-Exptime hardness result follows from Lemma 6.5, and Proposition 5.2. To show the upper bound, let $\mathcal{C}$ be a 2-PPDS. By Lemma 5.4 we can assume that $\mathcal{C}$ is rank-aware. To solve $Game(\mathcal{C})$ we construct a 2-PDS $\mathcal{C}'$ (without panic), as in Proposition 6.6. The system $\mathcal{C}'$ has the number of stack letters which is exponential in the size of $\mathcal{C}$, but the number of locations of $\mathcal{C}'$ is linear in the number of locations of $\mathcal{C}$. By the first part of Lemma 6.5, the winner in the game $Game(\mathcal{C}')$ can be decided in time doubly exponential in the number of states and singly exponential in the number of stack symbols, which gives us the desired complexity. □

By the relation between automata, μ-calculus, and monadic logic (cf. Section 2), we obtain the following.

Corollary 6.8. *It is 2-*Exptime*-complete to check if $[\![\mathcal{G}]\!] \models \varphi$, for a given 2nd order grammar $\mathcal{G}$, and a given μ-calculus formula φ. Consequently, the analogous problem for the monadic second-order logic (MSO) is decidable, in particular the MSO theory of any hyperalgebraic tree is decidable.*

References

[1] Aehlig, K., de Miranda, J.G., and Ong, L., Safety is not a restriction at level 2 for string languages. In: Proc. FOSSACS '05, Springer LNCS 3441 (2005), 490–504.

[2] Aehlig, K., de Miranda J.G., and Ong, L., The monadic second order theory of trees given by arbitrary level-two recursion schemes is decidable. In: Proc. TLCA '05, Springer LNCS 3461 (2005), 39–54.

[3] Cachat, T., Higher Order Pushdown Automata, the Caucal Hierarchy of Graphs and Parity Games. In: Proc. ICALP 2003, Springer LNCS 2719 (2003), 556–569.

[4] Cachat, T., Walukiewicz, I., The complexity of games on higher order pushdown automata, manuscript, 2004.

[5] Caucal, D., On infinite terms having a decidable monadic second-order theory. In: Proc. MFCS 2002, Springer LNCS 2420 (2002), 65–176.

[6] Courcelle, B., The monadic second-order theory of graphs IX: Machines and their behaviours. *Theoretical Comput. Sci.*, 151:125–162, 1995.

[7] Courcelle, B., Knapik, T., The evaluation of if first-order substitution is monadic second-order compatible *Theoretical Comput. Sci.*, 281(1-2):177–206, 2002.

[8] Damm, W., The IO- and OI-hierarchies. *Theoretical Comput. Sci.*, 20(2):95–208, 1982.

[9] Emerson, E. A., Jutla, C. S., Tree automata, mu-calculus and determinacy. In: *Proceedings 32th Annual IEEE Symp. on Foundations of Comput. Sci.*, IEEE Computer Society Press, 1991, pp. 368–377.

[10] Engelfriet, J., Iterated push-down automata and complexity classes. In: *Proc. 15th STOC*, 1983, pp. 365–373.

[11] Engelfriet, J., Schmidt, E.M., IO and OI, *J. Comput. System Sci.* **15**, 3, 1977, pp. 328–353, and **16**, 1, 1978, pp. 67–99.

[12] Grädel, E., Thomas, W., and Wilke, T., Editors, *Automata, Logics, and Infinite Games. A Guide to Current Research*, LNCS 1500, Springer-Verlag, 2002.

[13] Knapik, T., Niwiński, D., and Urzyczyn, P., Deciding monadic theories of hyperalgebraic trees. In: *Typed Lambda Calculi and Applications, 5th International Conference*, Springer LNCS 2044 (2001), 253–267.

[14] Knapik, T., Niwiński, D., Urzyczyn, P., Higher-order pushdown trees are easy. In: *Proc. FoSSaCS'02*, Springer LNCS 2303 (2002), 205–222.

[15] Knapik, T., Niwiński, D., Urzyczyn, P., Walukiewicz, I., Unsafe grammars and panic automata, draft, http://www.mimuw.edu.pl/∼ niwinski/prace.html.

[16] Maslov, A.N., The hierarchy of indexed languages of an arbitrary level, *Soviet Math. Dokl.*, **15**, pp. 1170–1174, 1974.

[17] A. W. Mostowski. Games with forbidden positions. Technical Report 78, Instytut Matematyki, University of Gdansk, 1991.

[18] Niwiński, D., Fixed points characterization of infinite behaviour of finite state systems. *Theoret. Comput. Sci.*, 189:1–69, 1997.

[19] Thomas, W., Languages, automata, and logic. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 3, Springer-Verlag, 1997, pp. 389–455.

[20] Walukiewicz, I., Pushdown processes: Games and model checking. *Information and Computation*, 164(2):234–263, 2001.