Unsafe grammars and panic automata

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

July 9, 2005

Abstract

We show that the problem of checking if an infinite tree generated by a higher-order grammar of level 2 satisfies a given μ-calculus formula (equivalently, if it is accepted by an alternating parity automaton) is decidable, actually 2EXPTIME-complete. Consequently, the monadic second-order theory of any hyperalgebraic tree is also decidable, so that the safety restriction can be removed from our previous decidability result [18]. The last decidability result has been independently obtained by Aehlig, de Miranda and Ong [2]. 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 systems with panic. The key step is to reduce such a game to a game over a 2nd order pushdown system without panic, for which a 2-EXPTIME procedure has been already shown [4]. However, the problem if panic adds the power to deterministic higher-order pushdown automata or, equivalently, if unsafe hyperalgebraic grammars define more trees than the safe ones, remains open.

Introduction

Context-free tree grammars constitute the basic level in an infinite hierarchy of higher-order grammars introduced by W. Damm [11] (built on the earlier ideas of [15]). Courcelle [9] proved decidability of the monadic second-order (MSO) theory of any *algebraic* tree, i.e., a tree generated by an algebraic (context-free) tree grammar. Later Knapik *et al* [18, 19] attempted to extend the decidability result for algebraic trees 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.

[1]ERIM, Université de la Nouvelle Calédonie, BP 4477, 98897 Nouméa Cedex, Nouvelle Calédonie
[2]Institute of Informatics, Warsaw University, Banacha 2, 02-097 Warsaw, POLAND
[3]LaBRI, Université Bordeaux-1, 351, Cours de la Libération, F-33 405, Talence cedex, France

Higher-order grammars can be seen as program schemes, where functions can take higher-order arguments. The tree generated by such a grammar describes the initial semantics of the program scheme. Thus decidability of 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 [20]: 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 [19]. See [8, 10] for another characterizations 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 [4] showed that the model-checking problem for the μ-calculus and the trees recognized by the pushdown automata of level n is in nEXPTIME, actually nEXPTIME-complete [5, 6].

In this paper we show that for the level 2 (hyperalgebraic), the model checking problem remains decidable, actually 2EXPTIME-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 seems to be essential for capturing the power of unsafe grammars, as it permits 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 [26, 4]. 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-EXPTIME procedure has been already shown [4].

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][1], 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 (2-EXPTIME).

At present we do not know if the safety requirement really restricts the generating power of the tree grammars. We conjecture that it is the case, and hope that panic automata can be useful in proving this conjecture. 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. Thus this result is not directly

[1] By kind assent of these authors, the preliminary version of [2] and of the present paper [21] appeared simultaneously in the Internet on 29 October, 24.

applicable to the case of trees we consider here.

1 Trees

Types and terms

We consider a set of types $\mathcal{T}$ constructed from a unique *basic* type $\mathbf{0}$. Thus, $\mathbf{0}$ is a type and, if τ_1, τ_2 are types, so is $(\tau_1 \to \tau_2) \in \mathcal{T}$. The operator $\to$ is assumed to associate to the right. Note that each type is of the form $\tau_1 \to \cdots \to \tau_n \to \mathbf{0}$, for some $n \geq 0$. A type $\mathbf{0} \to \cdots \to \mathbf{0}$ with $n+1$ occurrences of $\mathbf{0}$ is also written $\mathbf{0}^n \to \mathbf{0}$.

The level $\ell(\tau)$ of a type τ 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 $n > 0$. A type $\tau_1 \to \cdots \to \tau_n \to \mathbf{0}$ is *homogeneous* (where $n \geq 0$) if each τ_i is homogeneous and $\ell(\tau_1) \geq \ell(\tau_2) \geq \ldots \geq \ell(\tau_n)$. For example a type $((\mathbf{0} \to \mathbf{0}) \to \mathbf{0}) \to (\mathbf{0} \to \mathbf{0}) \to (\mathbf{0} \to \mathbf{0} \to \mathbf{0}) \to \mathbf{0} \to \mathbf{0}$ is homogeneous, but a type $\mathbf{0} \to (\mathbf{0} \to \mathbf{0}) \to \mathbf{0}$ is not.

A *typed alphabet* is a set Γ of symbols, each symbol γ in Γ given with a type τ in $\mathcal{T}$, which we write by $\gamma : \tau$. We extend Γ to the set $T(\Gamma)$ of *applicative terms* over Γ, inductively, by the rule

- if $t : \tau_1 \to \tau_2$ and $s : \tau_1$ then $(ts) : \tau_2$.

Note that each applicative term can be presented in a form $Zt_1 \ldots t_n$, where $n \geq 0$, $Z \in \Gamma$, and $t_1, \ldots, t_n$ are applicative terms. We adopt the usual notational convention that application associates to the left, i.e. we write $t_0 t_1 \ldots t_n$ instead of $(\cdots((t_0 t_1)t_2)\cdots)t_n$. For applicative terms $t, t_1, \ldots, t_m$, and symbols $z_1, \ldots, z_m$, of appropriate types, the term $t[z_1{:=}t_1, \ldots, z_k{:=}t_k]$ is defined as the result of simultaneous replacement in t of z_i by t_i, for $i = 1, \ldots, m$.

Trees

The free monoid generated by a set X is written X^* and the empty word is written ε. The length of word $w \in X^*$ is denoted by $|w|$.

A *tree* is any nonempty prefix-closed subset T of X^* (with ε as the *root*). If $u \in T$, $x \in X$, and $ux \in T$ then ux is an *immediate successor* of u in T. For $w \in T$, the set $T.w = \{v \in X^* : wv \in T\}$ is the *subtree* of T induced by w. Note that $T.w$ is also a tree, and $T.\varepsilon = T$.

Now let Σ be a *signature*, i.e., a typed alphabet of level 1. A symbol f in Σ is of type $\mathbf{0}^n \to \mathbf{0}$, for $n \geq 0$, and can be viewed as a (first-order) function symbol of *arity* n. Let $T \subseteq \omega^*$ be a tree. A mapping $t\colon T \to \Sigma$ is called a Σ-*tree* provided that if $t(w) : \mathbf{0}^k \to \mathbf{0}$ then w has exactly k immediate successors which are $w1, \ldots, wk$ (hence w is a leaf whenever $t(w) : \mathbf{0}$). The set of Σ-trees is written $T^\infty(\Sigma)$.

If $t\colon T \to \Sigma$ is a Σ-tree, then T is called the *domain* of t and denoted by $T = dom\ t$. For $v \in dom\ t$, the *subtree* of t induced by v is a Σ-tree $t.v$ such that $dom\ t.v = (dom\ t).v$, and $t.v(w) = t(vw)$, for $w \in dom\ t.v$.

We also need a concept of limit. For a Σ-tree t, let $t{\restriction}n$ be its truncation to the level n, i.e., the restriction of the function t to the set $\{w \in dom\ t \mid |w| \leq n\}$. Suppose $t_0, t_1, \ldots$ is a sequence of Σ-trees such that, for all k, there is an m, say $m(k)$, such that, for all $n, n' \geq m(k)$, $t_n{\restriction}k = t_{n'}{\restriction}k$. (This is a Cauchy condition in a suitable metric space of trees.) Then the *limit* of the sequence t_n, in symbols $\lim t_n$, is a Σ-tree t which is the set-theoretical union of the functions $t_n{\restriction}m(n)$ (understanding a function as a set of pairs).

In the sequel, we will often consider labeled trees (of bounded branching) more freely, but it will be always clear from the context that they can be presented in the above way, with a suitable choice of signature Σ.

Grammars

Let $\mathcal{X} = \{\mathcal{X}_\tau\}_{\tau \in \mathcal{T}}$ be an infinite typed alphabet of *variables* (or *parameters*).A *grammar* is a tuple $\mathcal{G} = (\Sigma, V, \mathcal{S}, E)$, where Σ is a *signature* (i.e., a finite alphabet of level 1), V is a finite set of *nonterminals*, each of a fixed, homogeneous type, $\mathcal{S} \in V$ is a *start symbol* of type $\mathbf{0}$, and E is a set of productions of the form

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

where $\mathcal{F} : \tau_1 \to \tau_2 \cdots \to \tau_m \to \mathbf{0}$ is a nonterminal in V, z_i is a variable of type τ_i, and w is an applicative term of type $\mathbf{0}$ in $T(\Sigma \cup V \cup \{z_1 \ldots z_m\})$. The *level* of a grammar is the highest level of its nonterminals. We assume that for each $\mathcal{F}$ in V, there is exactly one production in E with $\mathcal{F}$ occurring on the left hand side.

In this paper, we are interested in grammars as generators of Σ-trees. Let $\Sigma^\perp = \Sigma \cup \{\perp\}$, with $\perp : \mathbf{0}$. First, with any applicative term t over $\Sigma \cup V$, we associate an expression $t^\perp$ over signature $\Sigma^\perp$ inductively as follows.

- If $t = f$, $f \in \Sigma$, then $t^\perp = f$.
- If $t = X$, $X \in V$, then $t^\perp = \perp$.
- If $t = (sr)$ then if $s^\perp \neq \perp$ then $t^\perp = (s^\perp r^\perp)$, otherwise $t^\perp = \perp$.

Informally speaking, the operation $t \mapsto t^\perp$ replaces in t each nonterminal, together with its arguments, by $\perp$. It is easy to see that if t is an applicative term (over $\Sigma \cup V$) of type $\mathbf{0}$ then $t^\perp$ is an applicative term over $\Sigma^\perp$ of type $\mathbf{0}$. Recall that applicative terms over $\Sigma^\perp$ of type $\mathbf{0}$ can be identified with finite trees.

We will now define the single-step reduction relation $\to_{\mathcal{G}}$ between terms over $\Sigma \cup V$. therefore Informally speaking, $t \to_{\mathcal{G}} t'$ 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. Such a replacement is allowed only if F occurs as a head of a subterm of type $\mathbf{0}$. More precisely, the relation $\to_{\mathcal{G}} \subseteq T(\Sigma \cup V) \times T(\Sigma \cup V)$ is defined inductively by the following clauses.

1. $\mathcal{F}t_1 \dots t_k \to_{\mathcal{G}} t[z_1{:=}t_1, \dots, z_k{:=}t_k]$ if there is a production $\mathcal{F}z_1 \dots z_k \Rightarrow t$ (with $z_i : \rho_i$, $i = 1, \dots, k$), and $t_i \in T(\Sigma \cup V)_{\rho_i}$, for $i = 1, \dots, k$.
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.

When (1) above holds then we speak of a *single-step outermost reduction relation*, written $\to^{o}_{\mathcal{G}}$.

A *reduction* is a finite or infinite sequence of terms in $T(\Sigma \cup V)$, $t_0 \to_{\mathcal{G}} t_1 \to_{\mathcal{G}} \dots$. As usual, the symbol $\twoheadrightarrow_{\mathcal{G}}$ stands for the reflexive transitive closure of $\to_{\mathcal{G}}$. We also define the relation $t \twoheadrightarrow^{\infty}_{\mathcal{G}} t'$, where t is an applicative term in $T(\Sigma \cup V)$ and t' is a tree in $T^{\infty}(\Sigma^{\perp})$, by

- t' is a finite tree, and there is a finite reduction sequence $t = t_0 \to_{\mathcal{G}} \dots \to_{\mathcal{G}} t_n = t'$, or
- t' is infinite, and there is an infinite reduction sequence $t = t_0 \to_{\mathcal{G}} t_1 \to_{\mathcal{G}} \dots$ 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 $t'(w) = \perp$. (In other words, t' is obtained from t by replacing some of its subtrees by $\perp$.) Then we let

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

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. Furthermore, it is routine to show that if an infinite reduction $S = t_0 \to_{\mathcal{G}} t_1 \to_{\mathcal{G}} \dots$ is *fair*, i.e., any occurrence of a nonterminal symbol is eventually rewritten, then its result $t' = \lim t_n^{\perp}$ is $[\![\mathcal{G}]\!]$.

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, whom we call Eve and Adam. It can be presented 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 $\Omega : Q \to \omega$ is the ranking function.

The players start a play in the position p_1 and then move the token according to the move relation (always to an immediate 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 make a move, this player 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 is the winner.

Parity games were introduced by Emerson and Jutla [13, 22] and since then have been recognized as a very useful tool for solving the model checking problems. An important property is that not only the game is *determined*, i.e., one of the players has always a winning strategy, but the winner has also a *positional* winning strategy. Recall that a positional strategy, say for Eve, can be presented as a (partial) function $\sigma : V_\exists \to V$, and it is winning (for Eve) if every play $p_1 = q_1, q_2, \ldots$, such that if $q_\ell \in V_\exists$ then $q_{\ell+1} = \sigma(q_\ell)$. (See [25, 16] for basic introduction to parity games.)

In the sequel, we sometimes simply say that Eve *wins* a game $\mathcal{G}$ to mean that she has a winning strategy; similarly for Adam.

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 $[\![\mathcal{G}]\!] \models \varphi$. To this end, we need to present syntactic trees as logical structures (see, e.g., [19]). The most expressive logics considered in literature are the *monadic second-order logic* (MSO) or the *μ-calculus*. Both formalisms have the same expressive power over trees, but the μ-calculus is more tractable algorithmically. While, for the MSO, the problem is non-elementary already for regular trees, it is *n-EXPTIME*-complete for the μ-calculus and trees generated by *safe* grammars of level n [4]. In this paper we avoid logical machinery by using another equivalent formalism, namely *alternating automata on trees* (defined below). A polynomial-time translation is well-known [13] from the μ-calculus to alternating automata, $\varphi \mapsto A_\varphi$, such that, for any tree t, $t \models \varphi$ iff t is recognized by A_φ; in fact this translation is used in most of the model checking procedures involving the μ-calculus.

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 is a symbol in Σ of 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 in terms of a suitable parity game. We first define a labeled tree $r : dom\, r \to Q$, which can be viewed as a "tree of all computations" of $\mathcal{B}$ on t. We let $dom\, r \subseteq (\omega \times \omega)^*$, such that, for any $u \in dom\, r$, the projection $u{\downarrow_1}$ on the first component is a node in $dom\, t$. Moreover, we assume that $r(\epsilon) = q_1$, and whenever $r(u) = q$ and $t(\downarrow_1) = f$, then for any transition of the form $q \to f(q_1, \ldots, q_k)$, and all $i = 1, \ldots, k$, u has a successor $u(i, j)$, for some j, with $r(u(i, j)) = q_i$. 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 say that $\mathcal{B}$ *accepts* t if Eva wins this game.

The model checking problem considered in this paper is the following.

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

2 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 $a_1 \dots a_k$ over $\Gamma \times \omega$. A *level* 2 *pds* (2-*pds*, 2-stack) is a non-empty sequence $s_1 \dots s_l$ of 1-pds's, which may also be written as $[s_1][s_2]\dots[s_l]$ or as $s'[s_l]$, where s' stands for $[s_1][s_2]\dots[s_{l-1}]$. 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.

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

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 $m_{i,j} \le i-1$; moreover $j \le j'$ 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* $\mathcal{A}$.

Panic distribution. In the sequel it will be convenient to separate the "material content" of a push-down store (symbols in Γ) from the "time stamps" given at the second component. More specifically, let $s = [s_1] \dots [s_n]$ be a 2-stack over $\Gamma \times \omega$, with $s_i = (a_{i,1}, m_{i,1}) \dots (a_{i,k_i}, m_{i,k_i})$, for all i. We define

$$D(s) = \{(i, j) : 1 \le i \le n \text{ and } q \le j \le k_i\}.$$

So pairs in $D(s)$ identify all positions in the 2-stack s. We call the mapping $\theta_s : (i, j) \mapsto m_{i,j}$, for $(i, j) \in D(s)$, the *panic distribution for* s. Again, it is easy to see that if the 2-pds is reachable from the initial configuration then θ_s is monotone in both its arguments and $\theta_s(i, j) < i$ for all i. Let $s\downarrow_1$ be the 2-pds over Γ obtained by projection of s on the first component. Clearly, given $s\downarrow_1$ and θ_s, we can non-ambiguously reconstruct s.

In the sequel, we will usually present a configuration (q, s) as a triple (q, s', θ), where $s' =\downarrow_1 s$ and $\theta = \theta_s$. We will also occasionally write $panic(s')$ for the projection on the first component of $panic(s)$.

3 Automata to grammars

This section is not relevant to our main result and is added merely for completeness. Here we prove that if a panic automaton accepts a tree then this tree must be hyperalgebraic.

We assume that the given automaton $\mathcal{A}$ has m states, identified with numbers $1, \dots, m$, and we use the abbreviation $\mathbf{1} = \mathbf{0}^m \to \mathbf{0}$. The grammar we construct has the following nonterminals:

- For each $q \in Q$ and each $a \in \Gamma$, there is a nonterminal $\mathcal{F}^a_q$ of type $\mathbf{1}^m \to \mathbf{0}^m \to \mathbf{0}^m \to \mathbf{0}$.
- In addition there are nonterminals $\mathcal{S} : \mathbf{0}$, $\mathit{Void}_1 : \mathbf{1}$ and $\mathit{Void}_2 : \mathbf{0}$.

Before introducing the rules of the grammar we will show how we are going to represent stacks by expressions. more Let $s = [s_1]\dots[s_l]$ be a 2-pds, and let $s_i = a_{i,1}\dots a_{i,k_i}$, for all $i = 1, ..., l$. By $s|_{i,j}$ we denote the pds $[s_1]\dots[s_{i-1}][a_{i,1}\dots a_{i,j}]$. Of course we have $D(s|_{i,j}) \subseteq D(s)$. Fix a panic distribution θ for s. Then a restriction of θ to $D(s|_{i,j})$ is a panic distribution for $s|_{i,j}$, also denoted by θ. By induction with respect to (i,j) we define m-tuples of terms:

- $C_i : \mathbf{0}^m$, representing the 2-pds $s|_{i,k_i}$;
- $c_{i,j} : \mathbf{1}^m$, representing the 1-pds $[a_{i,1}, \dots a_{i,j}]$.

A stack is represented by an m-tuple. The q-th component $\downarrow_q C_i$ of this tuple should be understood as the representation of the stack *in state* q.

We begin with C_0 and $c_{i,0}$, defined as follows

$$C_0 = (\mathit{Void}_2, \dots, \mathit{Void}_2), \qquad c_{i,0} = (\mathit{Void}_1, \dots, \mathit{Void}_1).$$

Suppose $C_{j'}$ and $c_{i',j'}$ are defined for all (i',j') lexicographically preceding (i,j). Then:

- $c_{i,j} = (\mathcal{F}^{a_{i,j}}_1 c_{i,j-1} C_d, \dots, \mathcal{F}^{a_{i,j}}_m c_{i,j-1} C_d)$ where $d = \theta(i,j)$ or $d = 0$ if $i = 1$;
- $C_i = (\downarrow_1 c_{i,k_i} C_{i-1}, \dots, \downarrow_m c_{i,k_i} C_{i-1})$;

Write $\mathit{Code}_{q\,s,\theta}$ for $\downarrow_q C_l$. This term represents the configuration (q, s, θ).

The initial production of our grammar is

$$\mathcal{S} \Rightarrow \mathcal{F}^{\perp}_{q_1}\ \overrightarrow{\mathit{Void}_1}\, \overrightarrow{\mathit{Void}_2}\, \overrightarrow{\mathit{Void}_2},$$

where $\overrightarrow{\mathit{Void}_1}$ stands for m occurrences of Void_1, and similarly for $\overrightarrow{\mathit{Void}_2}$. Other productions are as follows. Below we write $\vec{\varphi}$ for $\varphi^1 \dots \varphi^m$ and similarly for $\vec{x}$ and $\vec{y}$. The intuition to be associated with the rules is that a term of the form $\mathcal{F}^a_q\ \vec{\varphi}\,\vec{x}\,\vec{y}$ is a representation of a configuration of the automaton.

1. $\mathcal{F}^a_q\ \vec{\varphi}\,\vec{x}\,\vec{y} \Rightarrow \mathcal{F}^b_p(\mathcal{F}^a_1\ \vec{\varphi}\,\vec{x}) \dots (\mathcal{F}^a_m\ \vec{\varphi}\,\vec{x})\ \vec{y}\,\vec{y}$, if $\delta(q,a) = (\mathit{push}_1\langle b\rangle, p)$.
2. $\mathcal{F}^a_q\ \vec{\varphi}\,\vec{x}\,\vec{y} \Rightarrow \mathcal{F}^a_p\ \vec{\varphi}\,\vec{x}\ (\mathcal{F}^a_1\ \vec{\varphi}\,\vec{x}\,\vec{y}) \dots (\mathcal{F}^a_m\ \vec{\varphi}\,\vec{x}\,\vec{y})$, if $\delta(q,a) = (\mathit{push}_2, p)$.
3. $\mathcal{F}^a_q\ \vec{\varphi}\,\vec{x}\,\vec{y} \Rightarrow \varphi^p\ \vec{y}$, if $\delta(q,a) = (\mathit{pop}_1, p)$.
4. $\mathcal{F}^a_q\ \vec{\varphi}\,\vec{x}\,\vec{y} \Rightarrow y^p$, if $\delta(q,a) = (\mathit{pop}_2, p)$.
5. $\mathcal{F}^a_q\ \vec{\varphi}\,\vec{x}\,\vec{y} \Rightarrow x^p$, if $\delta(q,a) = (\mathit{panic}, p)$.

6. $\mathcal{F}_q^a\ \vec{\varphi}\,\vec{x}\,\vec{y} \Rightarrow f(\mathcal{F}_{p_1}^a\ \vec{\varphi}\,\vec{x}\,\vec{y},\ldots,\mathcal{F}_{p_r}^a\ \vec{\varphi}\,\vec{x}\,\vec{y})$, if $\delta(q,a)=(f,p_1,\ldots,p_r)$.

The next two lemmas show that the implementation of 2-pds by above grammar is correct and that there is a correspondence between the moves of $\mathcal{A}$ and outermost reduction steps in $\mathcal{G}$.

Lemma 3.1 Let $(q,s,\theta)\rightarrow_{\mathcal{A}}^{\circ}(p,s',\theta')$. Then $Code_{q,\,s,\,\theta}\rightarrow_{\mathcal{G}}^{o} Code_{p,\,s',\,\theta'}$.

Proof: Suppose that $(q,s,\theta)\rightarrow_{\mathcal{A}}(p,s',\theta')$. We have

$$Code_{q,\,s,\,\theta} = \Downarrow_q C_l = \Downarrow_q c_{l,k_l}C_{l-1} = \mathcal{F}_q^{a_{l,k_l}}\, c_{l,k_l-1}C_{\theta(l,k_l)}C_{l-1}$$

In the case of $\delta(q,a_{l,k_l})=(push_1\langle a_{l,k_l+1}\rangle,p)$ we have

$$\begin{array}{rcl}\mathcal{F}_q^{a_{l,k_l}}\, c_{l,k_l-1}C_{\theta(l,k_l)}C_{l-1} & \rightarrow_{\mathcal{G}} & \mathcal{F}_p^{a_{l,k_l+1}}(\mathcal{F}_1^{a_{l,k_l}}\, c_{l,k_l-1}C_{\theta(l,k_l)})\ldots(\mathcal{F}_m^{a_{l,k_l}}\, c_{l,k_l-1}C_{\theta(l,k_l)})C_{l-1}C_{l-1}\\ & = & \mathcal{F}_p^{a_{l,k_l+1}}\, c_{l,k_l}C_{\theta'(l,k_l+1)}C_{l-1}\end{array}$$

where $\theta'=\theta\cup\{(l,k_l+1)\mapsto l-1\}$. Observe that the latter term is of the form $\downarrow_p C_l'$ where C_l' represents $s'=push_1\langle a_{l,k_l+1}\rangle(s)$ (and θ'). Now $\downarrow_p C_l' = Code_{p,\,push_1\langle a_{l,k_l+1}\rangle(s),\,\theta'}$.

In the case of $\delta(q,a_{l,k_l})=(push_2,p)$ we have

$$\begin{array}{rcl}\mathcal{F}_q^{a_{l,k_l}}\, c_{l,k_l-1}C_{\theta(l,k_l)}C_{l-1} & \rightarrow_{\mathcal{G}} & \mathcal{F}_p^{a_{l,k_l}}\, c_{l,k_l-1}C_{\theta(l,k_l)}(\mathcal{F}_1^{a_{l,k_l}}\, c_{l,k_l-1}C_{\theta(l,k_l)}C_{l-1})\ldots\\ & & \ldots(\mathcal{F}_m^{a_{l,k_l}}\, c_{l,k_l-1}C_{\theta(l,k_l)}C_{l-1})\\ & = & \mathcal{F}_p^{a_{l,k_l}}\, c_{l,k_l-1}C_{\theta(l,k_l)}(\Downarrow_1 c_{l,k_l}C_{l-1})\ldots(\Downarrow_m c_{l,k_l}C_{l-1})\\ & = & \mathcal{F}_p^{a_{l,k_l}}\, c_{l,k_l-1}C_{\theta(l,k_l)}C_l\\ & = & \mathcal{F}_p^{a_{l+1,k_{l+1}}}\, c_{l+1,k_{l+1}}C_{\theta(l+1,k_{l+1})}C_l\end{array}$$

because $a_{k+1,k_{l+1}}=a_{l,k_l}$, $c_{k+1,k_{l+1}}=c_{l,k_l}$, and $\theta(l,k_l)=\theta'(l+1,k_{l+1})$. Now

$$\mathcal{F}_p^{a_{l+1,k_{l+1}}}\, c_{l+1,k_{l+1}}C_{\theta(l+1,k_{l+1})}C_l = Code_{p,\,push_2(s),\,\theta'}.$$

In the case of $\delta(q,a_{l,k_l})=(pop_1,p)$ we have

$$\begin{array}{rcl}\mathcal{F}_q^{a_{l,k_l}}\, c_{l,k_l-1}C_{\theta(l,k_l)}C_{l-1} & \rightarrow_{\mathcal{G}} & \downarrow_p c_{l,k_l-1}C_{l-1}\\ & = & Code_{p,\,pop_1(s),\,\theta'}.\end{array}$$

In the case of $\delta(q,a_{l,k_l})=(pop_2,p)$ we have

$$\begin{array}{rcl}\mathcal{F}_q^{a_{l,k_l}}\, c_{l,k_l-1}C_{\theta(l,k_l)}C_{l-1} & \rightarrow_{\mathcal{G}} & \downarrow_p C_{l-1}\\ & = & Code_{p,\,pop_2(s),\,\theta'}.\end{array}$$

In the case of $\delta(q,a_{l,k_l})=(panic,p)$ we have

$$\begin{array}{rcl}\mathcal{F}_q^{a_{l,k_l}}\, c_{l,k_l-1}C_{\theta(l,k_l)}C_{l-1} & \rightarrow_{\mathcal{G}} & \downarrow_p C_{\theta(l,k_l)}\\ & = & Code_{p,\,panic(s),\,\theta'}.\end{array}$$

∎

Lemma 3.2 Let $Code_{q,s,\theta} \rightarrow^{o}_{\mathcal{G}} t$ where t is a term starting with a nonterminal. Then $t = Code_{p,s',\theta'}$ and $(q,s,\theta) \rightarrow^{\circ}_{\mathcal{A}} (p,s',\theta')$.

Proof: Since the rules of G are in one-to-one correspondence with the elements of transition function of $\mathcal{A}$, we may reverse the reasoning of the previous proof. ∎

The correspondence between the moves of $\mathcal{A}$ and outermost reduction steps in $\mathcal{G}$ can be made precise thanks to the following two definitions.

Definition 3.3 The *tree of all the computations of* $\mathcal{A}$, denoted $Tr(\mathcal{A})$, is defined by the following conditions.

- The root is labeled by the initial configuration of $\mathcal{A}$.
- If a vertex is labeled by $\langle q,s,\theta\rangle$ and $\delta(q, top(s)) = (f, q'_1, \ldots, q'_r)$ then there are r sons labeled $\langle q'_1, s, \theta\rangle, \ldots, \langle q'_r, s, \theta\rangle$ respectively.
- If a vertex is labeled by $\langle q,s,\theta\rangle$ but $\delta(q, top(s))$ is not of the form $(f, q'_1, \ldots, q'_r)$ then there is only one son labeled by $\langle q', s', \theta'\rangle$ such that $\langle q,s,\theta\rangle \rightarrow_{\mathcal{A}} \langle q', s', \theta'\rangle$.

Definition 3.4 The *tree of the outermost derivations of* $\mathcal{G}$, denoted $Tr(\mathcal{G})$, is defined by the following conditions.

- The root is labeled by the initial configuration of S.
- If a vertex is labeled by $ft_1 \ldots t_r$ and $f \in \Sigma$ then there are r sons labeled $t_1, \ldots, t_r$ respectively.
- If a vertex is labeled by a term t the head symbol of which is not in Σ, then there is only one son labeled by t' such that $t \rightarrow^{o}_{\mathcal{G}} t'$.

There is a close connection between $Tr(\mathcal{A})$ and $Tr(\mathcal{G})$. However, a meaningful correspondence between the nodes of the two trees is not one-to-one in this case. Indeed, when $\mathcal{A}$ performs the instruction of the form $(f, p_1, \ldots, p_r)$, there are r sons in $Tr(\mathcal{A})$, whereas, due to the corresponding rule of $\mathcal{G}$, in $Tr(\mathcal{G})$ we have only one son (with the label of the form $ft_1 \ldots t_r$) which in turn has r sons. Moreover the root of $Tr(\mathcal{G})$ does not correspond to a configuration of $\mathcal{A}$. This motivates the definition of the following mapping $h\colon dom\,Tr(\mathcal{G}) - \{\varepsilon\} \rightarrow dom\,Tr(\mathcal{A})$.

- $h(1) = \varepsilon$.
- If w has a unique son labeled by $ft_1 \ldots t_r$ with $f \in \Sigma$, then $h(w1) = h(w)$.
- If w has k sons, none of which is labeled by $ft_1 \ldots t_r$ with $f \in \Sigma$, then $h(wi) = h(w)i$ for $i = 1, \ldots, k$.

By examining the rules of $\mathcal{G}$ we see that if a son has a label of the form $ft_1 \ldots t_r$ with $f \in \Sigma$, then it is a unique son of his parent. The head symbol of the label of the parent is in V. Moreover, if a node has several sons, then the head symbol of each of these sons is in V. The definition of h has therefore a correct basis and the next lemma shows that h makes sense.

Lemma 3.5 Let t with the head symbol $\mathcal{F}_q^a$ be the label of a node $w \in dom\, Tr(\mathcal{G}) - \{\varepsilon\}$. Then

1. $t = Code_{q,s,\theta}$ and $h(w)$ has a label $\langle q, s, \theta \rangle$.
2. Moreover, if the label of $w1$ is of the form $ft_1 \dots t_r$ with $f \in \Sigma$, then $\delta(q,a) = (f, p_1, \dots, p_r)$ and $h(w1) = h(w)$.
3. h is well defined.

Proof: We show (1) and (2) by induction on the depth of a node in $Tr(\mathcal{G})$.

For $w = 1$, we have the label $\mathcal{F}_{q_1}^{\perp} \overrightarrow{Void_1}\, \overrightarrow{Void_2}\, \overrightarrow{Void_2}$ in $Tr(\mathcal{G})$, $d(1) = \varepsilon$ and the root of $Tr(\mathcal{A})$ has the label $\langle q_1, [\perp], \{(1,1) \mapsto 0\} \rangle$. Now

$$\mathcal{F}_{q_1}^{\perp} \overrightarrow{Void_1}\, \overrightarrow{Void_2}\, \overrightarrow{Void_2} = Code_{q_1,[\perp],\{(1,1)\mapsto 0\}}$$

If in addition the label of $w1$ is of the form $ft_1 \dots t_r$ then we have precisely

$$\mathcal{F}_{q_1}^{\perp} \overrightarrow{Void_1}\, \overrightarrow{Void_2}\, \overrightarrow{Void_2} \rightarrow_{\mathcal{G}}^{o} f(\mathcal{F}_{p_1}^{\perp} \overrightarrow{Void_1}\, \overrightarrow{Void_2}\, \overrightarrow{Void_2}, \dots, \mathcal{F}_{p_r}^{\perp} \overrightarrow{Void_1}\, \overrightarrow{Void_2}\, \overrightarrow{Void_2})$$

according to a grammar rule which must correspond to a transition $\delta(q_1, \perp) = (f, p_1, \dots, p_r)$. By definition of h we have then also $h(11) = \varepsilon$.

We assume that (1) and (2) hold for a node w whose label starts with a nonterminal and consider the closest descendant u of w whose label starts with a nonterminal.

If u is the son of w then we conclude (1) for u due to Definition 3.4 and Lemma 3.2.

If u is not the son of w then there exists the son v of w obtained via a rule of type (6) of $\mathcal{G}$. Due to the transition $\delta(q,a) = (f, p_1, \dots, p_r)$ applicable in $h(w)$, we conclude (1) for u by Definition 3.4.

We check in addition that (2) holds for u in both cases.

Clause (3) is established similarly by induction. ∎

We state now the converse of the above lemma.

Lemma 3.6 Let $\langle q, s, \theta \rangle$ be the label of a node $w' \in dom\, Tr(\mathcal{A})$.

1. If $\delta(q, top(s))$ is of the form $(f, p_1, \dots, p_r)$ then $h^{-1}(w') = \{w, w1\}$ for some $w \in dom\, Tr(\mathcal{G}) - \{\varepsilon\}$, the label of w is $Code_{q,s,\theta}$ and the label of $w1$ is of the form $ft_1 \dots t_r$.
2. If $\delta(q, top(s))$ is not of the form $(f, p_1, \dots, p_r)$ then $h^{-1}(w') = \{w\}$ for some $w \in dom\, Tr(\mathcal{G}) - \{\varepsilon\}$, and the label of w is $Code_{q,s,\theta}$.

Proof: Straightforward, by induction on the depth of a node $w' \in dom\, Tr(\mathcal{A})$. ∎

Proposition 3.7 The trees accepted by $\mathcal{A}$ and generated by $\mathcal{G}$ are isomorphic.

Proof: We transform $Tr(\mathcal{A})$ (resp. $Tr(\mathcal{G})$) as follows.

1. We remove all labels except for nodes where an instruction of the form $(f, p_1, \dots, p_r)$ applies (resp. where the label starts with f). The latter labels are replaced by f.
2. Every maximal infinite branch with no labels is replaced by a single node labeled $\perp$.
3. Every maximal finite branch with no labels is replaced by a single vertex.

It is obvious that by this transformation we obtain the tree accepted by $\mathcal{A}$ (resp. generated by $\mathcal{G}$). The trees obtained from $Tr(\mathcal{G})$ and from $Tr(\mathcal{A})$ are isomorphic due to Lemmas 3.5 and 3.6. ∎

4 Grammars to automata

Now we show the converse of Proposition 3.7. If a tree is generated by an arbitrary grammar of level 2, then it is accepted by a panic automaton.

Suppose a second-order grammar $\mathcal{G}$ generates $t : T \to \Sigma$. Without loss of generality we may assume that the initial nonterminal $\mathcal{S}$ does not occur at the right hand sides of productions.

Let $\circ_1, \circ_2, \dots$ be new identifiers of type $\mathbf{0}$, not occurring in $\mathcal{G}$. These identifiers are called *holes*. The word *operator* abbreviates "variable or nonterminal".

We construct a panic automaton $\mathcal{A}$ accepting t. The push-down alphabet Γ of $\mathcal{A}$ consists of subterms of the right hand sides of the productions of G, possibly applied to some holes, so that the result is of type $\mathbf{0}$. More precisely, let $u = Ft_1 \dots t_d$ be such a subterm, where F is an operator of type $\tau_1 \to \dots \to \tau_d \to \mathbf{0}^k \to \mathbf{0}$. Then $u \circ_1 \dots \circ_k \in \Gamma$. In particular, if $u : \mathbf{0}$ then simply $u \in \Gamma$.

The holes $\circ_1, \dots, \circ_k$ represent "missing arguments" of the operator F. Since holes are new identifiers, one can safely identify $Ft_1 \dots t_d \circ_1 \dots \circ_k$ with $Ft_1 \dots t_d$ 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 first approximation. The automaton then works in phases, each phase beginning and ending in the distinguished state q_1. We now describe the possible behaviour in a phase, beginning with a configuration (q_1, s, θ).

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

In other words, as long as the current approximation begins with a nonterminal, we can determine the next production. The top-level information about such productions is recorded on the pds. In some cases, this top-level information is sufficient to make actual progress: we finally obtain an expression beginning with a terminal symbol.

T2 If $top(s) = ft_1 \dots t_r$ where f is a terminal, then the the automaton executes the instruction $\delta(q_1, ft_1 \dots t_r) = (f, p_1, \dots, p_r)$, followed (at the i-th branch of the run) by a pop_1 and $push_1\langle\, t_i\,\rangle$. That is, the next approximation at the i-th branch is t_i. (The latter applies only when $r \neq 0$. In case when $r = 0$, i.e., when f is a constant, there is of course 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.

The last case requires some explanation. Let us observe first of all 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. It now becomes important how the panic distribution is defined. It points out to exactly the moment in the past we need, namely to the stage when the hole was created.

Now we show the correctness of this construction. Assume for the beginning that we have a 2-pds $s = [s_1, \dots, s_l]$ with $s_i = [u_{i,1}, \dots, u_{i,k_i}]$, for all $i = 1, ..., l$. Each $u_{i,j}$ is a subterm of the right hand side of some rule (possibly filled with holes) and has type $\mathbf{0}$. We have defined the automaton so that the stack content (in state q_1, at the end of each phase) will have the following additional properties:

P1 Every $u_{i,j}$, except possibly for $j = k_i$, is a term starting with a nonterminal. Terms u_{i,k_i}, for $i < l$ must begin with a variable, and only the topmost term u_{l,k_l} is arbitrary.

P2 Every $u_{i,j}$ is of type $\mathbf{0}$ and of the form $u' \circ_1 \ldots \circ_k$ where u' contains no holes (holes can only occur at the end). Only u_{l,k_l} may simply be a hole.

P3 Variables appearing in $u_{i,j}$ are the formal parameters of the head nonterminal symbol in $u_{i,j-1}$. For all i, the element $u_{i,1}$ is always $\mathcal{S}$.

P4 If $u_{i,j}$ is not a hole and ends with $h > 0$ holes then $d = \theta(i,j)$ is defined and u_{d,k_d} is a term starting with a variable of type $\mathbf{0}^h \to \mathbf{0}$.

P5 If the top term u_{l,k_l} is a hole $\circ_r$ then $d = \theta(l,k_l)$ is defined and u_{d,k_d} is a term starting with a variable of arity at least r (of type $\mathbf{0}^h \to \mathbf{0}$, where $h \geq r$).

Lemma 4.1 Every reachable configuration (q_1, s, θ) of the automaton satisfies **P1**–**P5**.

Proof: It follows from the definition that the conditions **P1**–**P5** are invariants of the computations of the automaton. ∎

For each operator F of type $\tau_1 \to \cdots \to \tau_r \to \mathbf{0}$ we define the *virtual parameters* of F as $\bar{x}_1, \ldots, \bar{x}_r$. These are new variables, i.e., variables not occurring in $\mathcal{G}$ and different from holes (to avoid confusion we put bars over virtual parameters). The *virtual parameters at* (i,j) (not to be confused with variables actually occurring in the expressions $u_{i,j}$) are the virtual parameters of the head symbol of $u_{i,j}$.

The next step is to define the *meaning* of an expression u at position (i,j), written as $[\![u]\!]_{i,j}$. This meaning is taken with respect to a fixed panic distribution θ. The expression u can contain variables occurring in $u_{i,j}$, virtual parameters at (i,j), and holes.

- If F is a signature constant or a nonterminal, then $[\![F]\!]_{i,j} = F$ for all i,j.
- If u is an application t_1t_2 then $[\![u]\!]_{i,j} = [\![t_1]\!]_{i,j}[\![t_2]\!]_{i,j}$.
- If u is a variable x then this variable is a formal parameter of the head symbol in $u_{i,j-1}$, say the e-th one. We put $[\![x]\!]_{i,j} = [\![\bar{x}_e]\!]_{i,j-1}$.
- If u is a virtual parameter $\bar{x}_d$, and $u_{i,j} = Ft_1 \ldots t_r$, for some F, then $[\![u]\!]_{i,j} = [\![t_d]\!]_{i,j}$.
- If u is a hole $\circ_r$ then $[\![\circ_r]\!]_{i,j} = [\![\bar{x}_r]\!]_{d,k_d}$, where $d = \theta(i,j)$.

Write $[\![s]\!]_\theta$ for $[\![u_{l,k_l}]\!]_{l,k_l}$. This is the expression that will result from an "evaluation" of *top*(s) under the environment provided by s and θ.

Lemma 4.2 For every reachable configuration (q_1, s, θ) of the automaton, the meaning $[\![s]\!]_\theta$ is a term of type $\mathbf{0}$, which contains no variables.

Proof: Straightforward. ∎

Lemma 4.3 If $(q_1, s, \theta) \twoheadrightarrow_{\mathcal{A}} (q_1, s', \theta')$ and $top(s)$ does not begin with a terminal then $[\![s]\!]_\theta \twoheadrightarrow_{\mathcal{G}} [\![s']\!]_{\theta'}$.

Proof: Suppose that $(q_1, s, \theta) \twoheadrightarrow_{\mathcal{A}} (q_1, s', \theta')$ in at least one step, and that in addition no configuration of the form (q_1, s'', θ'') occurs during this computation. We show that:

1. If $top(s)$ begins with a variable or $top(s)$ is a hole, then $[\![s]\!]_\theta = [\![s']\!]_{\theta'}$.
2. If $top(s)$ begins with a nonterminal, then $[\![s]\!]_\theta \rightarrow_{\mathcal{G}} [\![s']\!]_{\theta'}$.

We consider five cases, depending on the types T1–T5 of moves that $\mathcal{A}$ can do. The appropriate type is in turn determined by $top(s)$.

If $\mathcal{A}$ has made move of type T1 then $top(s) = \mathcal{F}u_1 \dots u_n$ and $top(s') = u$ where $\mathcal{F}x_1 \dots x_n \Rightarrow u$ is a production. Then

$$[\![s]\!]_\theta = [\![\mathcal{F}u_1 \dots u_n]\!]_{l,k_l} = \mathcal{F}[\![u_1]\!]_{l,k_l} \cdots [\![u_n]\!]_{l,k_l} \quad \text{and} \quad [\![s']\!]_{\theta'} = [\![u]\!]_{l,k_l+1} = u[x_l := [\![u_l]\!]_{i,j}]_{l=1}^n$$

so that indeed $[\![s]\!]_\theta \rightarrow_{\mathcal{G}} [\![s']\!]_{\theta'}$.

Move of type T2 is not possible as $top(s)$ does not start with a terminal.

In case of T3 we have $top(s) = x : \mathbf{0}$ then $s = s''[wux]$ where $u = \mathcal{F}t_1 \dots t_e$ and x is a j-th formal parameter of $\mathcal{F}$. Now $s' = s''[wt_j]$ and $[\![s]\!]_\theta = [\![x]\!]_{l,k_l} = [\![\overline{x}_j]\!]_{l,k_l-1} = [\![t_j]\!]_{l,k_l-1} = [\![s']\!]_{\theta'}$.

For the case of T4 we have $top(s) = \varphi u_1 \dots u_d$ with φ a variable of type $\mathbf{0}^d \rightarrow \mathbf{0}$. Then $s = s''[wuu']$ with $u' = \varphi u_1 \dots u_d$ and $u = \mathcal{F}t_1 \dots t_e$, where φ is a r-th formal parameter of $\mathcal{F}$. Now $s' = s''[wuu'][w\,(t_r \circ_1 \dots \circ_d)]$. We have

$$[\![s]\!]_\theta = [\![\varphi u_1 \dots u_d]\!]_{l,k_l} = [\![\varphi]\!]_{l,k_l}[\![u_1]\!]_{l,k_l} \cdots [\![u_d]\!]_{l,k_l} = [\![\overline{x}_r]\!]_{l,k_l-1}[\![u_1]\!]_{l,k_l} \cdots [\![u_d]\!]_{l,k_l} =$$
$$= [\![t_r]\!]_{l,k_l-1}[\![u_1]\!]_{l,k_l} \cdots [\![u_d]\!]_{l,k_l}$$

and $[\![s']\!]_{\theta'} = [\![t_r \circ_1 \dots \circ_d\,]\!]_{l+1,k_l-1} = [\![t_r]\!]_{l+1,k_l-1}[\![\,\circ_1\,]\!]_{l+1,k_l} \cdots [\![\,\circ_d\,]\!]_{l+1,k_l}$. Since the last pds operation of $\mathcal{A}$ has been $push_1\langle t_r \circ_1 \dots \circ_d\rangle$, we have $\theta'(l+1, k_l - 1) = l$. Hence $[\![\,\circ_i\,]\!]_{l+1,k_l} = [\![\overline{x}_i]\!]_{l,k_l} = [\![u_i]\!]_{l,k_l}$ for $i = 1, \dots, d$. In order to prove that $[\![s']\!]_\theta = [\![s']\!]_{\theta'}$, it is now enough to show that $[\![t_r]\!]_{l,k_l-1} = [\![t_r]\!]_{l+1,k_l-1}$. The latter equality follows from the fact that $[\![t]\!]_{i,j}$ depends only on the contents of s_i and on the contents of the 1-pds's reachable from $s|_{i,j}$ by panic moves.

Finally, in the case of T5, we have $top(s) = \circ_r$ then $[\![s]\!]_\theta = [\![\,\circ_r\,]\!]_{l,k_l} = [\![\bar{x}_r]\!]_{d,k_d}$, where $d = \theta(l, k_l)$. Given the shape of s' (cf. condition **P5**) we have again $[\![s]\!]_\theta = [\![s']\!]_{\theta'}$. ∎

Lemma 4.4 If (q_1, s, θ) is a reachable configuration then $(q_1, s, \theta) \twoheadrightarrow_{\mathcal{A}} (q_1, s', \theta')$, where $top(s')$ begins with a terminal or with a nonterminal.

Proof: Assume that a phase begins in (q_1, s, θ) and ends in (q_1, s', θ'). Observe that if $top(s)$ begins with a function variable and $top(s')$ does not begin with a terminal or with a nonterminal then it must again begin with a function variable. Moreover in this case, the top row of s' is shorter than that of s. Thus the number of such steps is bounded.

Now suppose that $top(s)$ is a variable of type $\mathbf{0}$ or a hole. Then s' is of a smaller size than s. Thus, after a finite number of steps, the term on the top of the stack must begin with a terminal or a nonterminal or with a function variable. But then the previous case applies. ∎

Lemma 4.5 Suppose that $\mathcal{A}$ accepts a tree t. Then t is the tree generated by $\mathcal{G}$.

Proof: Let ϱ be the run of $\mathcal{A}$ on t. Observe that, for every $w \in dom\, t$, $\varrho(w)$ is a configuration with the state q_1. We write it $\langle q_1, s_w, t_w \rangle$. Define t_n as the term obtained by truncating t to level n and inserting $[\![s_w]\!]_{\theta_w}$ at every node w on level n. We claim that $\mathcal{S} \twoheadrightarrow_{\mathcal{G}} t_n$ for every n.

To prove the claim observe the following. If $top(s_w) = f u_1 \ldots u_n$ (and thus $t(w) = f$) then $[\![s_{wd}]\!]_{\theta_{wd}} = [\![u_d]\!]_{l,k_l}$, for $d = 1, \ldots, n$, so that $[\![s_w]\!]_\theta \to_{\mathcal{G}} f[\![s_{w1}]\!]_{\theta_{w1}} \cdots [\![s_{wn}]\!]_{\theta_{wn}}$. The above, together with Lemma 4.3, implies the claim. ∎

Lemma 4.6 Suppose that $\mathcal{G}$ generates a tree t. Then $\mathcal{A}$ accepts t.

Proof: We claim that any (properly) partial run can be extended. The automaton started in any reachable configuration must eventually encounter a $top(s)$ beginning with a terminal or nonterminal (Lemma 4.4). In the latter case, a production is simulated and we again apply Lemma 4.4. This cannot run forever.

Indeed, since $\mathcal{G}$ generates a tree, we can assume that every sequence of (outermost) reductions $t_1 \to^o_{\mathcal{G}} t_2 \to^o_{\mathcal{G}} \cdots$ must lead to a term beginning with a terminal. (Nonterminals that generate loops can be eliminated as unreachable.) ∎

Proposition 4.7 Each hyperalgebraic tree is accepted by a panic automaton.

Proof: From Lemmas 4.5 and 4.6. ∎

Remark. The proof of proposition 4.7 is based on an idea from [17]. The difference between the construction in Section 4 and that in [17] is that, in the latter case, individual parameters were passed *by value*. Thus, a nonlocal access to an individual meant an immediate access to a register value and did not require any additional evaluation. In the present case, parameters of type $\mathbf{0}$ are passed *by name*. This means that a variable of type $\mathbf{0}$ must be evaluated, and for this a proper environment must be restored. This is exactly the use of *panic*: to restore the environment in which a ground type expression is to be evaluated.

5 Pushdown systems

In the previous sections, we have considered panic automata as tree acceptors, with the aim to characterize (unsafe) grammars. In order to study model-checking problem, it is convenient to consider pushdown systems without input alphabet, but equipped with ranks and alternation, similarly as in [4, 26].

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 s finite store alphabet, $p_1 \in P$ is the initial location, $\Omega : P \to \omega$ is the rank function, and $\Delta \subseteq P \times \Gamma \times P \times Op_2$ is the finite set of (unlabeled) transition rules. Here Op_2 is the set of pushdown operations: $push_1\langle a\rangle, pop_1, push_2, pop_2, panic, skip$. These operations act on 2nd level stacks (2-stacks) as described on page 8; the operation *skip* does not change the stack. A transition $(p, a, p', \alpha) \in \Delta$ will be sometimes 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 an 2-stack over $\Gamma \times \omega$. Like for panic automata, we will often present it as a triple (q, s', θ), where s' is a projection of s on Γ, and θ is the panic distribution of s (c.f., page 8).

A 2-PPDS as above defines a parity game $Game(\mathcal{C}) = (V_\exists, V_\forall, E, (p_1, [(\perp, 0)]), \Omega)$, 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 $s' = \alpha(s)$;
- $\Omega(p, s) = \Omega(p)$.

We will consider

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

Proposition 5.1 Problem 1 and Problem 2 are polynomial-time equivalent.

Proof: *(Sketch)* We first reduce Problem 1 (page 7) to Problem 2. Let $\mathcal{G}$ be a 2nd order grammar, and $\mathcal{B}$ an alternating parity tree automaton. By construction of section 4, we obtain (in polynomial time) a panic automaton $\mathcal{A}$ which accepts precisely the tree $t_\mathcal{A} = [\![\mathcal{G}]\!]$.

We have to construct a 2-PPDS $\mathcal{C}$ such that Eve wins the game $Game(\mathcal{C})$ if and only if she wins the game $Game(\mathcal{B}, t_\mathcal{A})$. The construction of $\mathcal{C}$ is, of course, based on a product of $\mathcal{A}$ and $\mathcal{B}$ and is fairly standard, although tedious.

The converse transformation of Problem 2 to Problem 1 requires introduction of a suitable functional signature Σ (not present in a 2-PPDS). This can be done on the basis of transitions of a 2-PPDS $\mathcal{C}$. Again the construction is straightforward and we omit the details. ∎

6 Making pushdown systems rank-aware

This section prepares the proof of our main result in section 7. We show that, by applying a non-expensive transformation if necessary, we can control the dynamics of ranks by the topmost symbols on the pushdown store.

When a pushdown system makes a panic move in a configuration (q, s, θ), the contents of the pds is brought back to a configuration which was already encountered earlier in the computation. More precisely, if the next configuration is (q', s', θ') then there has been a previous configuration (p, s', θ'), where s' and θ' were the same. We want to know what was the highest rank of a state visited between (p, s', θ') and (q, s, θ).

Let $\mathcal{C}$ be a 2-PPDS as in section 5, and let $Tr(\mathcal{C})$ be the tree obtained by unfolding the graph of the game $Game(\mathcal{C})$. That is, the root of $Tr(\mathcal{C})$ is labeled by the initial configuration, and a node label by a configuration (p, s) has a successor labeled (p', s') for each (p', s'), such that $(p, s)E(p', s')$.

We formulate the notion of a rank-aware automaton in terms of $Tr(\mathcal{C})$. It is useful to observe that a node of the tree determines a computation up to this node.

Definition 6.1 Consider a node v of $Tr(\mathcal{C})$ labeled with $\langle q, s, \theta\rangle$ such that $panic(s)$ is defined. Let v' be the closest to v ancestor of v labeled with $\langle q', panic(s), \theta'\rangle$, for some q' and θ'. We call v' the *panic ancestor of* v. A *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 rank of the panic ancestor of v.

Definition 6.2 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, θ) is equal to $Rank(top(s))$. That is, the panic rank is determined by the top of the stack.

Lemma 6.3 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 $Game(\mathcal{C})$.

The proof of Lemma 6.3 occupies the rest of this section. Consider a 2-PPDS $\mathcal{C} = (P, P_\exists, P_\forall, \Gamma, p_1, \Delta, \Omega)$. We construct a rank-aware 2-PPDS $\mathcal{C}' = (P', P'_\exists, P'_\forall, \Gamma', p'_1, \Delta', \Omega')$.

The interpretation of $Tr(\mathcal{C})$ in $Tr(\mathcal{C}')$ is quite straightforward. The idea is that configurations of $\mathcal{C}'$ are like configurations of $\mathcal{C}$ with additional information stored on the stack. The stack alphabet of $\mathcal{C}'$ is defined as $\Gamma' = \Gamma \times \{0, \dots, d\} \times \{0, \dots, d\}$, where d is the highest rank in the range of Ω. That is, each stack symbol is now of the form (a, m_p, m_l) and consists of an "ordinary" stack symbol a plus an additional pair of numbers m_p and m_l. These are used to code the information about panic rank as follows. If (a, m_p, m_l) is currently on top of the stack, then:

m_p is the panic rank of the node;

m_l is the highest rank of a state seen since the creation of the current top row.

We now define the transitions Δ' of $\mathcal{C}'$. Sometimes we describe only the expected behaviour of $\mathcal{C}'$, leaving details to the reader. The definition of $P'_\exists$ and $P'_\forall$ is implicit; they are extensions of $P_\exists$ and $P_\forall$, respectively, by some auxiliary locations. Below, the notation $\to$ and $\twoheadrightarrow$ abbreviates respectively $\to_{\mathcal{C}'}$ and $\twoheadrightarrow_{\mathcal{C}'}$.

Case 1: For a transition $q, a \to_\Delta q', skip$ of $\mathcal{C}$, we define transitions of $\mathcal{C}'$ so that for all v, w, θ,

$$(q, v[w(a, m_p, m_l)], \theta) \twoheadrightarrow (q'_i, v[w(a, m'_p, m'_l)], \theta),$$

where $m'_p = \max(m_p, \Omega(q'_i))$ and $m'_l = \max(m_l, \Omega(q'_i))$.

Case 2: For a transition $q, a \to_\Delta q', push_1\langle\, b\,\rangle$ of $\mathcal{C}$, we let

$$(q, (a, m_p, m_l) \to_{\Delta'} q', push_1\langle\, b, m'_p, m'_l\,\rangle) \in \Delta'$$

be a transition of $\mathcal{C}'$, where $m'_l = m'_p = \max(m_l, \Omega(q'))$.

Case 3: For a transition $q, a \to_\Delta q', push_2$, we let $\mathcal{C}'$ also execute $push_2$ and adjust the auxiliary information so that:

$$(q, v[w(a, m_p, m_l)], \theta) \twoheadrightarrow (q'_i, v[w(a, m_p, m_l)][w(a, m'_p, m'_l)], \theta'),$$

where $m'_p = \max(m_p, \Omega(q'))$ and $m'_l = \Omega(q')$.

Case 4: For a transition $q, a \to_\Delta q', pop_1$, the new PPDS must of course execute a pop_1, but the information at the top must be also adjusted so that we have:

$$(q, v[w(b, m_p^2, m_l^2)(a, m_p^1, m_l^1)], \theta) \twoheadrightarrow (q' v[w(b, m'_p, m'_l)], \theta')$$

where $m'_p = \max(m_p^2, m_p^1, \Omega(q'))$ and $m'_l = \max(m_l^1, \Omega(q'))$.

Case 5: For a transition $q, a \to_\Delta q', pop_2$, the new PPDS makes of course pop_2, but it must also update the auxiliary information:

$$(q, v[w^2(b, m_p^2, m_l^2)][w^1(a, m_p^1, m_l^1)], \theta) \twoheadrightarrow (q', v[w(b, m'_p, m'_l)], \theta)$$

where $m'_p = \max(m_p^2, m_l^1, \Omega(q'))$ and $m'_l = \max(m_l^1, m_l^2, \Omega(q'))$.

Case 6: Finally, we consider the case $q, a \to_\Delta q', panic$. Of course, $\mathcal{C}'$ must panic too, but again we have to adjust the auxiliary values. Suppose that the configuration is $(q, v[w(a, m_p^1, m_l^1)], \theta)$ and after the panic we obtain $(q', v'[w'(b, m_p^2, m_l^2)], \theta')$. We want the new PPDS to do the updates so that it arrives at the configuration $(q', v'[w'(b, m'_p, m'_l)], \theta')$ where $m'_p = \max(m_p^1, m_p^2, \Omega(q'))$ and $m'_l = \max(m_p^1, m_l^2, \Omega(q'))$.

Each move of $\mathcal{C}$ is simulated by a sequence of moves of $\mathcal{C}'$, beginning and ending in nodes of the form (q, s, θ), where q is a location of $\mathcal{C}$. We call such nodes the *true nodes* of $Tr(\mathcal{C}')$. During the simulation, $\mathcal{C}'$ uses auxiliary locations, not in P. These locations are of rank zero and have no importance for the game. For all locations in P, Ω' coincides with Ω. The correctness of the construction is implied by Lemma 6.5 below. $Tr(\mathcal{C}')$ For the proof to go through we must also say what is the meaning of the labels "inside" the stack and this requires an extension of Definition 6.1.

Definition 6.4 Consider a node v of $Tr(\mathcal{C})$ labeled with $\langle q, s, \theta\rangle$, where $s = [s_1]\ldots[s_l]$, and a position $(i, j) \in D(s)$. Let v' be the closest to v ancestor of v labeled with $\langle q', [s_1]\ldots[s_{\theta(i,j)}], \theta'\rangle$. We call v' the *panic ancestor of the position* (i, j) *in* s. The *panic rank of* (i, j) *in* v is the maximal rank of a state occurring between the panic ancestor of (i, j) and v.

Remark A panic ancestor of a node (cf. Definition 6.1) is the panic ancestor of the topmost position of the stack from the label of the node.

The following lemma describes the invariant of the construction. In particular it implies the properties of m_p and m_l announced before.

Lemma 6.5 Let v be a true node of $Tr(\mathcal{C}')$ labeled with (q, s, θ). Let $(a^{(i,j)}, m_p^{(i,j)}, m_l^{(i,j)})$ be the element at the position (i, j) of the stack s. Let (n, k_n) be the topmost position in s and let (i, k_i) be the top position of the i-th row of s. We have the following.

P1 The panic rank of (i, j) in v is $\max(\{m_p^{(i,j)}, m_p^{(i,k_i)}\} \cup \{m_l^{(x,k_x)} : x = i+1, \ldots, n\})$.

P2 The value of $m_l^{(i,k_i)}$ is the maximal rank of a location visited between the creation of the most recent occurrence of the i-th row and the last appearance of this occurrence on the top of the 2-stack.

Proof: The properties hold in the root of $\mathcal{C}'$. We suppose that they hold in a true node v of $Tr(\mathcal{C}')$ and we show that they are satisfied in every next true node v' of $Tr(\mathcal{C}')$. We proceed by cases as in the definition of $\mathcal{C}'$.

A general observation is that the panic rank of a position (i, j) in v' is just the maximum of the panic rank of the position in v and of the rank $\Omega(q')$ of the newly seen location q'.

CASE 1 The effect of updates $m'_p = \max(m_p, \Omega(q'))$ and $m'_l = \max(m_l, \Omega(q'))$ is that **P1** is satisfied (due to the remark above). Also obviously the new m_l at the top of the stack should be the maximum of the previous value and $\Omega(q')$.

CASE 2 In the case of $push_1$, the panic return for the newly created position is the stack just below. Thus the panic rank of the new position should be the maximum of the previous m_l and of $\Omega(q')$. For the same reason, the new m_l should be the same as the new m_p. The panic returns of all the other positions satisfy **P1**.

CASE 3 In the case of $push_2$, the panic return for the newly created top symbol is the same as for the previous top. Thus m'_p is just $\max(m_p, \Omega(q'))$. The only location seen since the creation of the new level is q', thus $m'_l = \Omega(q')$. These two changes also guarantee **P1**.

CASE 4 In the case of pop_1, the correctness of our definition of m'_p follows from **P1**. Also m'_l is calculated from m_l^1 because the top row has not changed. To show that **P1** is satisfied, observe that $m'_l \geq \Omega(q')$ and that the return rank of any position in the stack is not smaller then m_p^2 (the other returns go at least to the same depth as the top one). It follows that property **P1** still holds also in the top row.

CASE 5 The case of pop_2 follows directly from the properties **P1** and **P2**.

CASE 6 The main observation here is that by property **P2** we have $m_p^2 = \max\{m_l^{(i,k_i)} : i = d+1, \ldots, n\}$, where $d = \theta(n, k_n)$. Thus the settings of m'_p and m'_l are correct. ∎

Applied to the topmost position, Lemma 6.5 shows that $\mathcal{C}'$ is rank-aware. In order to close the proof of Lemma 6.3, we have to establish that Eve has a winning strategy in $Game(\mathcal{C})$ iff she has one in $Game(\mathcal{C}')$. This is obvious since $Tr(\mathcal{C})$ and $Tr(\mathcal{C}')$ restricted to true nodes are isomorphic. The other nodes are universal, of rank 0 and have no impact on a play. Applying the same strategy on $Game(\mathcal{C})$ than on $Game(\mathcal{C}')$, Eve obtains the same outcome in both games.

7 Deciding the winner in $Game(\mathcal{C})$

A *2nd order pushdown system* (or 2-PDS, for short) is like a 2-PPDS defined in section 5, but *without* panic, that is, $\Delta \subseteq P \times \Gamma \times P \times Op_2 - \{panic\}$. The concept of configuration is then simplified in the obvious way: as we do not make use of the panic distribution, it is enough to consider the 2-stacks over Γ (instead of $\Gamma \times \omega$). The concept of $Game(\mathcal{C})$ is modified accordingly.

Games over such pushdown systems have been considered by Engelfriet [14], Cachat [4], and others (see [5] for a broad presentation), from where we take the following result.

Lemma 7.1 ([14, 4, 6]) It is decidable if Eve has a winning strategy in $Game(\mathcal{C})$ for a given 2nd-order pushdown system $\mathcal{C}$. There is an algorithm working in time $2^{\mathcal{O}(|\Gamma| 2^{\mathcal{O}(|P|)})}$ where $|P|$ is the number of locations and $|\Gamma|$ the size of the stack alphabet of $\mathcal{C}$. Moreover, the problem is 2EXPTIME-complete.

In the aforementioned references the dependence on the size of the stack alphabet (single exponential only) is not stated explicitly. However, in [26] it is explicitly shown that for 1st order systems there is a decision procedure of complexity $\mathcal{O}(|\Gamma| 2^{c|P|})$ constant c. The same kind of analysis in 2nd-order setting gives the desired result.

We will now show how to transform a 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 $Game(\mathcal{C})$.

Recall that the rank-awareness of $\mathcal{C}$ means that we have a function $Rank : \Gamma \to \{0, \ldots, d\}$ such that for any node (q, s, θ) of $Tr(\mathcal{C})$, the value of $Rank(top(s))$ is the highest rank of a location visited between the panic ancestor and the current node.

For technical convenience, we further assume that a transition $p, a \to_\Delta p', \alpha$ with $\alpha \neq skip$ is possible only for $p \in P_\exists$. Clearly this is not an essential restriction since we can always achieve this property by duplicating locations and using *skip* operation.

Before describing the construction, we introduce a useful ordering on ranks:

$$m \preccurlyeq n \quad \text{iff} \quad (-1)^m \cdot m \leq (-1)^n \cdot n.$$

(For example, for $d = 6$, this yields an ordering $5, 3, 1, 0, 2, 4, 6$.) Intuitively, if $m \preccurlyeq n$ then m is a rank at most as good as n with respect to the acceptance condition.

The set of returns is defined as:

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

i.e. as the set of all partial functions from states to ranks. Intuitively, such a function assigns to each state q the worst, in $\preccurlyeq$-order, panic rank still acceptable for the panic moves *ending* in state q.

We proceed with the definition of $\mathcal{C}' = \langle P', P'_\exists, P'_\forall, \Gamma', p'_1, \Delta', \Omega' \rangle$. The locations for Eve, $P'_\exists$, will be precisely the locations of $P_\exists$ plus the loosing state. The set of locations for Adam will consist of $P_\forall$ plus some annotated locations of P, the winning state, and some auxiliary set of locations Ver (for "verifying"):

$$\begin{aligned} P'_\exists =& P_\exists \cup \{\bot\}, \\ P'_\forall =& P_\forall \cup \{p_{push}, p_{ver}, p_{rank=i} : p \in P, i \in \{0,\dots,d\}\} \cup \{\top\} \cup \{ver\} \cup Aux \end{aligned}$$

The stack alphabet of $\mathcal{C}'$ is:

$$\Gamma' = Ret \cup (\Gamma \times Ret).$$

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

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

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

- If $p, a \to_\Delta p', skip$ then $p, (a, R) \to_{\Delta'} p', skip$, for all $R \in Ret$.
- If $p, a \to_\Delta p', push_1\langle a\rangle$ then $p, (a, R) \to_{\Delta'} p'_{ver}, push_1\langle (a', R')\rangle$, for all $R, R' \in Ret$. Additionally we have $p'_{ver}, (a', R') \to_{\Delta'} ver, skip$, and $p'_{ver}, (a', R') \to_{\Delta'} p', skip$.
- 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 $p, a \to_\Delta p', push_2$ then $p, (a, R) \to_{\Delta'} p'_{push}, push_1\langle R'\rangle$, for all $R, R' \in Ret$. Additionally we put in Δ' the rules[2] $p'_{push}, R' \to_{\Delta'} p', push_2; pop_1$, as well as $p'_{push}, R' \to_{\Delta'} p''_{rank=R'(p'')}, pop_1$, 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 $\bot$.

Theorem 7.2 *Eve wins $Game(\mathcal{C}')$ iff she wins $Game(\mathcal{C})$.*

The theorem is proved by the following two lemmas. But first we need some definitions.

Definition 7.3 Consider a sequence $v = [w_1 R_1] \dots [w_k R_k][w_{k+1}]$ where $w_i \in (\Gamma \times Ret)^*$ and $R_i \in Ret$. We say that v *represents* a stack with a panic distribution (s, θ) if

- $s = [s_1] \dots [s_{k+1}]$ and for all i, $s_i = w_i \downarrow_1$, i.e. s_i is the projection of w_i on first components (w_i is a word over $\Gamma \times Ret$ and s_i is a word over Γ).

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

- for every $(i,j) \in D(s)$ we have $R_{i,j} = R_{\theta(i,j)}$, where $R_{i,j}$ is the second component of the symbol occurring in w_i at the position j.

Definition 7.4 Let n and o be nodes of $Game(\mathcal{C})$. We say that o makes a *panic jump of type* (ρ, q', l) *over* n if

- $\delta(q, top(s)) = (q', panic)$; where (q, s, θ) is the configuration labeling o;
- panic ancestor of o is an ancestor of n;
- $panic(s)$ is a 2-stack containing l 1-stacks;
- $\rho = Rank(top(s))$.

Lemma 7.5 If Adam wins $Game(\mathcal{C})$ then he wins $Game(\mathcal{C}')$.

Proof: Fix a winning strategy σ for Adam in $Game(\mathcal{C})$. This is a subtree of the game tree $Game(\mathcal{C})$. We "copy" it to get a winning strategy in $Game(\mathcal{C}')$.

Suppose that the play is in the position $(q, v[w])$ and that there is an associated node n of $Game(\mathcal{C})$ with the following properties:

- the label of n is (q', s, θ) where $q' = q$ and $v[w]$ represents (s, θ);
- if in the strategy tree σ there is a descendant o of n that makes a panic jump of type (ρ, q', l) over n then $\rho \not\succcurlyeq R_l(q')$ where R_l is the top element of the l-th stack in $v[w]$, i.e., $v[w] = [w_1 R_1] \dots [w_k R_k][w]$.

We determine the next move of Adam in $Game(\mathcal{C}')$ by copying the move of Adam in σ from n. This is easy in all the cases but if the move from n is $(q', push_2)$. In this case, in $Game(\mathcal{C}')$ Eve moves to $(q'_{push}, v[wR'])$ for some $R' \in Ret$. If there is a state q'' such that in σ there is a panic jump from some descendant o of n to n of type (ρ, q'', l) with $\rho \succcurlyeq R'(q'')$ then Adam chooses to move to $(q''_{rank=R'(q'')}, v[w])$ and then to $(q'', v[w])$. The associated node becomes o', the successor of o. If it is not the case then Adam chooses the position $(q', v[wR'][w])$ and the associated node is the successor of n.

Because of the above properties Adam is assured not to reach $\top$. It is not difficult to see that he wins also all infinite plays. ∎

Lemma 7.6 If Eve wins $Game(\mathcal{C})$ then Eve wins $Game(\mathcal{C}')$.

Proof: Fix a strategy σ for Eve in $Game(\mathcal{C})$. It is a part of the arena of $Game(\mathcal{C})$.

We say that $v = [w_1 R_1] \dots [w_k R_k]$ *covers all panic jumps over* n if whenever in σ there is a panic jump of type (ρ, q', l) over n then $\rho \succcurlyeq R_l(q')$.

We want to define a strategy for Eve. Suppose that we have a position $(q, v[w])$ and that we were able to associate to it a node n of $Game(\mathcal{C})$ such that

- the label of n is (q', s, θ) with $q' = q$ and $v[w]$ representing (s, θ);
- all the panic jumps over n are covered by v.

We want to define the move from $(q, v[w])$ and associate a new node with the resulting position. We consider the case when $\delta(q, top(s)) = (q', push_2)$, the rest being easy.

If $\delta(q, top(s)) = (q', push_2)$ then, by the definition of $Game(\mathcal{C}')$, Eve needs to move from $(q, v[w])$ to $(q'_{push}, v[wR'])$ for some R'. We make her move to the position with R' defined by:

$$R'(q) = \min_{\preccurlyeq}\{\rho : \text{there is a panic jump to } n \text{ in } \sigma \text{ of a type } (\rho, q, l)\}$$

The minimum is taken with respect to the $\preccurlyeq$ order and the function is undefined if the set is empty.

Now, if Adam chooses $(q', v[wR'][w])$ then we associate to this position the successor n' of n. If, on the other hand, Adam chooses $(q''_{rank=R'(q'')}, v[w])$, for some $q'' \in dom\, R'$, then this position has the unique successor $(q'', v[w])$. In the strategy σ there is a descendant o of n which makes a panic jump of type $(R'(q''), q'', l)$. We choose the successor of o as the associated node to $(q'', v[w])$.

It is not difficult to check that this strategy is winning for Eve in $Game(\mathcal{C})$. ∎

This completes the proof of Theorem 7.2.

We are now ready to state our main result.

Theorem 7.7 *The problem of checking if Eva wins $Game(\mathcal{C})$ for a given 2-PPDS $\mathcal{C}$ is 2EXPTIME-complete (Problem 1). Consequently, so is the problem of checking if a given alternating parity tree automaton $\mathcal{A}$ accepts the tree $[\![\mathcal{G}]\!]$, for a given 2nd order grammar $\mathcal{G}$ (Problem 2).*

Proof: The 2EXPTIME hardness result follows from Lemma 7.1, and Proposition 5.1. To show the upper bound, let $\mathcal{C}$ be a 2-PPDS. By Lemma 6.3 we can assume that $\mathcal{C}$ is rank-aware. Making an automaton rank aware increases its size only by a linear factor. To solve $Game(\mathcal{C})$ we construct a 2-PDS $\mathcal{C}'$ (without panic), as earlier in this section, such that the winner in $Game(\mathcal{C}')$ is the same as in $Game(\mathcal{C})$ (Theorem 7.2). The system $\mathcal{C}'$ has the number of stack letters which is exponential in the size of $\mathcal{C}$ but the number of states of $\mathcal{C}'$ is linear in the number of states of $\mathcal{C}$. By the first part of Lemma 7.1, 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 order logic (c.f. section 1), we obtain the following.

Corollary 7.8 The model-checking problem

Given a 2nd order grammar $\mathcal{G}$, and a μ-calculus formula φ, decide if $[\![\mathcal{G}]\!] \models \varphi$

is decidable, actually 2EXPTIME-complete. 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. Foundations of Software Science and Computation Structures (FOSSACS '05), LNCS, to appear.

[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. Proceedings of the 7th International Conference on Typed Lambda Calculi and Applications (TLCA '05), LNCS, to appear.

[3] Arnold, A., and Niwiński, D., *Rudiments of μ-Calculus.* Elsevier Science, Studies in Logic and the Foundations of Mathematics, 146, North-Holland, Amsterdam, 2001.

[4] Cachat, T., Higher Order Pushdown Automata, the Caucal Hierarchy of Graphs and Parity Games, proceedings of ICALP 2003, LNCS 2719, pp. 556-569.

[5] Cachat, T., Games on Pushdown Graphs and Extensions, PhD thesis, RWTH Aachen, 2003.

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

[7] Carayol, A., Wöhrle, S., The Caucal Hierarchy of Infinite Graphs in Terms of Logic and Higher-Order Pushdown Automata. In Paritosh Pandya and Jaikumar Radhakrishnan, editors, *FST&TCS 2003*, LNCS 2914, Springer-Verlag, 2003, pp. 112–123.

[8] Caucal, D., On infinite terms having a decidable monadic second-order theory. In K. Diks and W. Rytter, editors, *27th Mathematical Foundations of Computer Science*, LNCS 2420, Springer-Verlag, 2002, pp. 65–176.

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

[10] 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.

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

[12] Damm, W., Goerdt, A., An automata-theoretic characterization of the OI-hierarchy, *Information and Control*, **71**, 1986, 1–32.

[13] 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.

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

[15] 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.

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

[17] Kfoury, A.J., Urzyczyn, P., Finitely typed functional programs, Part II: comparisons to imperative languages, Research report, Boston University, 1988.

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

[19] Knapik, T., Niwiński, D., Urzyczyn, P., Higher-order pushdown trees are easy, *Proc. FoSSaCS'02* (M. Nielsen, Ed.), LNCS 2303, Springer-Verlag, Berlin, 2002, pp. 205–222.

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

[21] Knapik, T., Niwiński, D., Urzyczyn, P., Walukiewicz, I., Unsafe grammars, panic automata, and decidability, http://www.labri.fr/Perso/∼igw/Papers/igw–amazonka.ps, October 2004.

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

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

[24] Rabin, M. O., Decidability of second-order theories and automata on infinite trees. *Trans. Amer. Soc*, 141:1–35, 1969.

[25] 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.

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

[27] Walukiewicz, I., Monadic second-order logic on tree-like structures, *Theoret. Comput. Sci.*, 275(1–2):311–346, 2002.