Unsafe grammars, panic automata, and decidability

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

October 25, 2004

Abstract

We show that the monadic second-order theory of any hyperalgebraic tree, i.e., an infinite tree generated by a higher-order grammar of level 2, is decidable. Thus, for grammars of level 2, we can remove the safety restriction from our previous decidability result [10], although the question if this extends to higher levels remains open. The 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*. To show that the MSO theory of a tree recognized by a panic automaton is decidable, we give a reduction to a suitable parity game, which itself can be MSO-defined within a 2-tree. We then use the decidability of the MSO theory of a 2-tree [18].

Introduction

Context-free tree grammars constitute the basic level in an infinite hierarchy of higher-order grammars introduced by W. Damm [5] (building on the earlier ideas of [7]). Courcelle [4] 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* [10, 11] 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*. In this paper we show the decidability for all grammars of the second level of the hierarchy, also called *hyperalgebraic grammars*. In order to achieve this, we introduce a new model of stack automata that captures all hyperalgebraic grammars in the same way as second-order pushdown automata capture safe second-level grammars.

Context-free grammars can be seen as program schemes, i.e., recursive definitions of functions. Higher-order grammars correspond to higher-order program schemes, i.e., when functions can

[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

take higher-order arguments. The tree generated by such a grammar describes completely the 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. This requirement was crucial in the previous decidability results [10, 11]. The concept of safe (tree) grammars has been further justified by a characterization in terms of higher-order pushdown automata originally introduced by Maslov [12]: 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 [11]. Caucal [3] gave a very elegant, "internal" characterization of the same hierarchy of trees, now often referred to as *Caucal hierarchy*. Starting from regular trees, he has obtained the subsequent levels of the hierarchy by applying two graph operations: inverse regular mapping and unfolding.

The above mentioned results suggest that some new tools may be needed to cope with unsafe grammars. In this paper we characterize the second-level (hyperalgebraic) grammars by an extension of second-order pushdown automata. We introduce 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 use the characterization of grammars by panic automata to show that the monadic second-order theory of an infinite tree generated by any hyperalgebraic grammar is decidable. As the algorithm is uniform, we obtain decidability of the *model-checking* problem for the hyperalgebraic grammars. To show that the MSO theory of a tree recognized by a panic automaton (of level 2) is decidable, we follow the ideas of [17]. That is, we reduce the problem to solving a suitable parity game, which itself can be defined within a 2-tree. We then use the decidability of the MSO theory of a 2-tree [18].

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, de Miranda and Ong have studied the safety restriction [13]. 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 applicable to the setting we consider here.

While finishing the work on this paper, the authors became aware that a result similar to the main result of this paper had just been independently obtained by Klaus Aehlig [1].

1 Tree grammars

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 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 $T = \mathit{dom}\ t$. For $v \in \mathit{dom}\ t$, the *subtree* of t induced by v is a Σ-tree $t.v$ such that $\mathit{dom}\ t.v = (\mathit{dom}\ t).v$, and $t.v(w) = t(vw)$, for $w \in \mathit{dom}\ t.v$.

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 $\{w \in \mathit{dom}\ t \mid |w| \leq n\}$. 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 labelled 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.

Monadic second-order logic

Let R be a *relational vocabulary*, i.e., a set of relational symbols, each r in R given with an arity $\rho(r) > 0$. The formulas of *monadic second order (MSO) logic* over vocabulary R use two kinds of variables: *individual variables* $x_0, x_1, \ldots$, and *set variables* $X_0, X_1, \ldots$ Atomic formulas are $x_i = x_j$, $r(x_{i_1}, \ldots, x_{i_{\rho(r)}})$, and $X_i(x_j)$. The other formulas are built using propositional connectives $\vee, \neg$, and the quantifier $\exists$ ranging over both kinds of variables. (The connectives $\wedge, \Rightarrow$, etc., as well as the quantifier $\forall$ are introduced in the usual way as abbreviations.) A formula without free variables is called a *sentence*. Formulas are interpreted in relational structures over the vocabulary R, which we usually present as $\mathbf{A} = \langle A, \{r^{\mathbf{A}} : r \in R\}\rangle$, where A is the *universe* of $\mathbf{A}$, and $r^{\mathbf{A}} \subseteq A^{\rho(r)}$ is a $\rho(r)$-ary relation on A. A *valuation* is a mapping v from the set of variables (of both kinds), such that $v(x_i) \in A$, and $v(X_i) \subseteq A$. The *satisfaction* of a formula φ in $\mathbf{A}$ under the valuation v, in symbols $\mathbf{A}, v \models \varphi$ is defined by induction on φ in the usual manner. The *monadic second-order theory* of $\mathbf{A}$ is the set of all MSO sentences satisfied in $\mathbf{A}$, in symbols $MSO(\mathbf{A}) = \{\varphi : \mathbf{A} \models \varphi\}$.

Let Σ be a typed alphabet of level 1, and suppose that the maximum of the arities of symbols in Σ exists and equals m_Σ. A tree $t \in T^{\infty}(\Sigma)$ can be viewed as a logical structure $\mathbf{t}$, over the vocabulary $R_\Sigma = \{p_f : f \in \Sigma\} \cup \{d_i : 1 \leq i \leq m_\Sigma\}$, with $\rho(p_f) = 1$, and $\rho(d_i) = 2$:

$$\mathbf{t} = \langle dom\, t,\ \{p_f^{\mathbf{t}} : f \in \Sigma\} \cup \{d_i^{\mathbf{t}} : 1 \leq i \leq m_\Sigma\}\rangle.$$

The universe of $\mathbf{t}$ is the domain of t, and the predicate symbols are interpreted by $p_f^{\mathbf{t}} = \{w \in dom\, t :\ t(w) = f\}$, for $f \in \Sigma$, and $d_i^{\mathbf{t}} = \{(w, wi) : wi \in dom\, t\}$, for $1 \leq i \leq m_\Sigma$. We refer the reader to [16] for a survey of the results on monadic second-order theory of trees.

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.

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 rewriting relation $\rightarrow_{\mathcal{G}}$ between terms over $\Sigma \cup V$. Informally speaking, $t \rightarrow_{\mathcal{G}} t'$ 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 $\rightarrow_{\mathcal{G}} \subseteq T(\Sigma \cup V) \times T(\Sigma \cup V)$ is defined inductively by the following clauses.

- $\mathcal{F} t_1 \dots t_k \rightarrow_{\mathcal{G}} t[z_1{:=}t_1, \dots, z_k{:=}t_k]$ 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 $i = 1, \dots, k$.
- If $t \rightarrow_{\mathcal{G}} t'$ then $(st) \rightarrow_{\mathcal{G}} (st')$ and $(tq) \rightarrow_{\mathcal{G}} (t'q)$, whenever the expressions in question are applicative terms.

A *reduction* is a finite or infinite sequence of terms in $T(\Sigma \cup V)$, $t_0 \rightarrow_{\mathcal{G}} t_1 \rightarrow_{\mathcal{G}} \dots$. As usual, the symbol $\twoheadrightarrow_{\mathcal{G}}$ stands for the reflexive transitive closure of $\rightarrow_{\mathcal{G}}$. We also define the relation $t \twoheadrightarrow_{\mathcal{G}}^{\infty} t'$, 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 $t = t_0 \rightarrow_{\mathcal{G}} \dots \rightarrow_{\mathcal{G}} t_n = t'$, or
- t' is infinite, and there is an infinite reduction sequence $t = t_0 \rightarrow_{\mathcal{G}} t_1 \rightarrow_{\mathcal{G}} \dots$ 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

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

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 $S = t_0 \rightarrow_{\mathcal{G}} t_1 \rightarrow_{\mathcal{G}} \dots$ 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.

2 Panic automata

Let us start with basic concepts. A *level* 1 *pushdown store* (or a 1-*pds* or 1-stack) over an alphabet A is simply a non-empty word $a_1 \dots a_l$ over A. A *level* 2 *pds* (or a 2-*pds* or 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. The intuition is that push-down stores grow to the right, so that, for instance, the symbol a is on top of the 1-stack wa, while wa is on top of $s[wa]$.

The following operations are possible on push-down stores of level 1:

- $push_1\langle a \rangle(s) = sa$;
- $pop_1(sa) = s$;

On push-down stores of level 2 one can perform the following operations:

- $push_2(s[w]) = s[w][w]$;
- $pop_2(s[v][w]) = s[v]$;
- $push_1\langle a \rangle(s[w]) = s[wa]$;
- $pop_1(s[wa]) = s[w]$;

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). We will use $top(s)$ to denote the top element of the top row of a 2-stack s.

Now let Σ be a signature, and let Q and Γ be finite sets. Let $\perp \in \Gamma$ be a distinguished element of Γ. The set $\mathcal{I}$ of *instructions* (parameterized by Σ, Q and Γ) consists of all tuples of the following forms:

1. $(push_1\langle a \rangle, p)$, where $p \in Q$, and $a \in \Gamma$, $a \neq \perp$.
2. $(push_2, p)$, where $p \in Q$.
3. (pop_k, p), where $p \in Q$ and $k = 1, 2$.
4. $(panic, p)$, where $p \in Q$.
5. $(f, p_1, \dots, p_r)$, where $f \in \Sigma_r$ and $p_1, \dots, p_r \in Q$.

A *panic automaton* is defined as a tuple

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

where Σ is a signature, 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*. That is, a panic automaton is a second-order push-down store automaton (cf. [11]) with one additional type of instruction $(panic, p)$. Because of this new instruction, a push-down store of a panic automaton contains some additional information.

A *configuration* of an automaton $\mathcal{A}$ as above is a pair (q, s), where $q \in Q$, and s is a 2-pds over $\Gamma \times \omega$ (where ω denotes the set of natural numbers). The *initial configuration* is $(q_1, [(\perp, 0)])$.

We define the relation $\to_{\mathcal{A}}$ on configurations as follows. In all the conditions below it is assumed that $s = [s_1] \ldots [s_l]$, and $(q, m) = top(s)$.

1. If $\delta(q, a) = (push_1\langle\, a' \,\rangle, q')$ then $\langle q, s\rangle \to_{\mathcal{A}} \langle q', push_1\langle\, a', l-1 \,\rangle(s)\rangle$.
2. If $\delta(q, a) = (push_2, q')$, then $\langle q, s\rangle \to_{\mathcal{A}} \langle q', push_2(s)\rangle$.
3. If $\delta(q, a) = (pop_k, q')$, $k = 1, 2$, then $\langle q, s\rangle \to_{\mathcal{A}} \langle q', pop_k(s)\rangle$ (provided $pop_k(s)$ is defined).
4. If $\delta(q, a) = (panic, q')$ then $\langle q, s\rangle \to_{\mathcal{A}} \langle q', [s_1] \ldots [s_m]\rangle$, provided that $2 \le m < l$. In this case we write $[s_1] \ldots [s_m] = panic(s)$.
5. If $\delta(q, a) = (f, p_1, \ldots, p_r)$ then $\langle q, s\rangle \to_{\mathcal{A}} \langle p_i, s\rangle$, for all $i = 1, \ldots, r$.

If one of the clauses (1)–(4) holds then we can write $\to^{\circ}_{\mathcal{A}}$ instead of $\to_{\mathcal{A}}$. The symbol $\twoheadrightarrow_{\mathcal{A}}$ stands for the reflexive and transitive closure of $\to_{\mathcal{A}}$, and similarly for $\twoheadrightarrow^{\circ}_{\mathcal{A}}$.

The idea of the second component in the push-down store is the following. A new symbol $a \in \Gamma$ is placed on the top of the stack together with 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 first appeared in the stack.

It is easy to see that if a 2-pds $[s_1] \ldots [s_l]$ can be obtained by $\twoheadrightarrow_{\mathcal{A}}$ from the initial configuration then, for each symbol (a, m) in the row s_i, we have $m < i$. In particular, $panic(s)$ is defined (case 4 above), provided $l \ge 2$.

In the sequel it will be convenient to separate the "material content" of a push-down store (symbols in Γ) from the "history information" given at the second component. More specifically, let $s = [s_1] \ldots [s_n]$ be a 2-stack over $\Gamma \times \omega$, with $s_i = (a_{i,1}, m_{i,1}) \ldots (a_{i,k_i}, m_{i,k_i})$, 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 is monotone in both its arguments and $\theta(i, j) < i$ for all i. Let $\pi_1(s)$ be the 2-pds over Γ obtained by projection of s on the first component. Clearly, given $\pi_1(s)$ 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 $s' = \pi_1(s)$ and $\theta = \theta_s$. We will also occasionally write $panic(s')$ for the projection on the first component of $panic(s)$.

Let $t : T \to \Sigma$ be a Σ-tree. A partial function $\varrho : T \to \mathcal{C}$ (where $\mathcal{C}$ is the set of all configurations) defined on an initial fragment of T is called a *partial run* of $\mathcal{A}$ on t iff the following condition holds:

> If $\varrho(w) = \langle q, s, \theta\rangle$ for some $w \in T$ then $\delta(q, a) = (f, p_1, \ldots, p_r)$, where $f = t(w)$ and $a = top(s)$. In addition $\langle p_i, s\rangle \twoheadrightarrow^{\circ}_{\mathcal{A}} \varrho(wi)$, for each $i = 1, \ldots, r$ when $\varrho(wi)$ is defined.

If a partial run is a total function then it is called 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 accepted by* $\mathcal{A}$.

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 $1, \ldots, m$, 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. Let $s = [s_1] \ldots [s_l]$ be a 2-pds, and let $s_i = a^1_i \ldots a^{k_i}_i$, for all $i = 1, ..., l$. By $s|_{i,j}$ we denote the pds $[s_1] \ldots [s_{i-1}][a^1_i \ldots a^j_i]$. 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 $[a^1_i, \ldots a^j_i]$.

A stack is represented by an m-tuple. The q-th component $\pi_q(C_i)$ 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

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

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

- $c_{i,j} = (\mathcal{F}^{a^j_i}_1 c_{i,j-1} C_d, \ldots, \mathcal{F}^{a^j_i}_m c_{i,j-1} C_d)$ where $d = \theta(i, j)$ or $d = 0$ if $i = 1$;
- $C_i = (\pi_1(c_{i,k_i})C_{i-1}, \ldots, \pi_m(c_{i,k_i})C_{i-1})$;

Write $Code_{q,s,\theta}$ for $\pi_q(C_l)$. This term represents the configuration (q, s, θ).

The initial production of our grammar is

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

where $\overrightarrow{Void}_1$ stands for m occurrences of $Void_1$, and similarly for $\overrightarrow{Void}_2$. 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.

- $\mathcal{F}^a_q\ \vec{\varphi}\vec{x}\vec{y} \ \Rightarrow\ \mathcal{F}^a_p\ \vec{\varphi}\vec{x}\ (\mathcal{F}^a_1\ \vec{\varphi}\vec{x}\vec{y}) \dots (\mathcal{F}^a_m\ \vec{\varphi}\vec{x}\vec{y}),\quad$ if $\delta(q, a) = (push_2, p)$.
- $\mathcal{F}^a_q\ \vec{\varphi}\vec{x}\vec{y} \ \Rightarrow\ \mathcal{F}^b_p(\mathcal{F}^a_1\ \vec{\varphi}\vec{x}) \dots (\mathcal{F}^a_m\ \vec{\varphi}\vec{x})\ \vec{y}\,\vec{y},\quad$ if $\delta(q, a) = (push_1\langle\, b\,\rangle, p)$.
- $\mathcal{F}^a_q\ \vec{\varphi}\vec{x}\vec{y} \ \Rightarrow\ y^p,\quad$ if $\delta(q, a) = (pop_2, p)$.
- $\mathcal{F}^a_q\ \vec{\varphi}\vec{x}\vec{y} \ \Rightarrow\ \varphi^p\ \vec{y},\quad$ if $\delta(q, a) = (pop_1, p)$.
- $\mathcal{F}^a_q\ \vec{\varphi}\vec{x}\vec{y} \ \Rightarrow\ x^p,\quad$ if $\delta(q, a) = (panic, p)$.
- $\mathcal{F}^a_q\ \vec{\varphi}\vec{x}\vec{y} \Rightarrow f(\mathcal{F}^a_{p_1}\ \vec{\varphi}\vec{x}\vec{y}, \dots, \mathcal{F}^a_{p_r}\ \vec{\varphi}\vec{x}\vec{y}),\quad$ if $\delta(q, a) = (f, p_1, \dots, p_r)$.

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

Proof: The proof is similar to the proof for ordinary pds automata [11] in cases when the computation step $\to_{\mathcal{A}}$ is an ordinary push-down operation. In the case of *panic*, one should observe that the production $\mathcal{F}^a_q\ \vec{\varphi}\vec{x}\vec{y} \ \Rightarrow\ x^p$ applied to $Code_{q,s,\theta}$, where $s = [s_1, \dots, s_l]$ and s_l is of length k_l, delivers the code of $s_{\theta(l,k_l)}$. ∎

Lemma 3.2 Let ϱ be the run of $\mathcal{A}$ on $t : T \to \Sigma$ and let $w \in T$. Let $\varrho(w) = (q, s, \theta)$, with $top(s) = a$, and let $\delta(q, a) = (f, p_1, \dots, p_r)$. Then $\mathcal{S} \twoheadrightarrow_{\mathcal{G}} t'$, for some finite tree t' with $t'(w) = Code_{q,s,\theta}$.

Proof: Induction with respect to the length of w. ∎

Proposition 3.3 A tree accepted by a panic automaton must be hyperalgebraic.

Proof: From Lemmas 3.1 and 3.3. ∎

4 Grammars to automata

Now we show the converse of Proposition 3.3. 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 $t : T \to \Sigma$. With no loss of generality we may assume that the initial nonterminal $\mathcal{S}$ does not occur at the right hand sides of productions.

Let $\circ_1, \circ_2, \ldots$ 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 $u = Ft_1 \ldots t_d$ be such a subterm, where F is an operator 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 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 $top(s) = \mathcal{F}u_1 \ldots u_d$, where $\mathcal{F}$ is a nonterminal, and let the corresponding production be $\mathcal{F}x_1 \ldots x_n \Rightarrow u$. The right hand side u of this production is our next approximation. That is, the automaton executes the instruction $\delta(q_1, \mathcal{F}u_1 \ldots u_d) = (push_1\langle\, u \,\rangle, q_1)$.

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 $top(s) = ft_1 \ldots t_n$ where f is a terminal, then the the automaton executes the instruction $\delta(q_1, ft_1 \ldots t_n) = (f, p_1, \ldots, p_r)$, followed (at the i-th branch of the run) by a pop_1 and $push_1\langle\, t_i \,\rangle$. 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 $\mathcal{F}t_1 \ldots t_e$, 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 $top(s) = \varphi u_1 \ldots u_h$ 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 $u_1, \ldots, u_h$ would be lost. Instead, a $push_2$ is executed, followed by a pop_1. Then we proceed as in the previous case, but now the new approximant t_j is an expression of a functional type and we actually place $t_j \circ_1 \ldots \circ_h$ 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 $\psi v_1 \ldots v_l$ 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 $s = [s_1, \ldots, s_l]$ with $s_i = [u_{i,1}, \ldots, u_{i,k_i}]$, 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 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 $d = \theta(i, j)$ 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 $h \geq r$).

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 $u_{i,j}$.

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. ∎

The next step is to define the *meaning* of an expression u at (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 $t_1 t_2$ 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 $u_{i,j} = F t_1 \ldots t_r$, for some F, then $[\![u]\!]_{i,j} = [\![t_d]\!]_{i,j}$.
- If u is a hole $\circ_r$ then $[\![\circ_r]\!]_{i,j} = [\![\bar{x}_r]\!]_{l,k_l}$, where $l = \theta(i,j)$.

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 $top(s) = \mathcal{F} u_1 \ldots u_n$ and $top(s') = u$ where $\mathcal{F} x_1 \ldots x_n \Rightarrow u$ is a production. Then $[\![s]\!]_\theta = (\mathcal{F} x_1 \ldots x_n)[x_l := [\![u_l]\!]_{i,j}]_{l=1}^n$ and $[\![s']\!]_{i,j+1} = u[x_l := [\![u_l]\!]_{i,j}]_{l=1}^n$, 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 $[\![s]\!]_{i,j} = [\![x]\!]_{i,j} = [\![t_e]\!]_{i,j-1} = [\![s']\!]_\theta$, where t_e is the appropriate actual parameter.

For the case of T4 we have $top(s) = \varphi u_1 \ldots u_d$ with φ a variable of type $\mathbf{0}^d \rightarrow \mathbf{0}$. Then $[\![s]\!]_\theta = [\![\varphi]\!]_{i,j} [\![u_1]\!]_{i,j} \cdots [\![u_d]\!]_{i,j} = [\![t_e]\!]_{i,j-1} [\![u_1]\!]_{i,j} \cdots [\![u_d]\!]_{i,j} = [\![t_e]\!]_{i+1,j-1} [\![\circ_1]\!]_{i+1,j-1} \cdots [\![\circ_d]\!]_{i+1,j-1} = [\![s']\!]_{\theta'}$.

Observe that the equality $[\![t_e]\!]_{i,j-1} = [\![t_e]\!]_{i+1,j-1}$ 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 $[\![s]\!]_\theta = [\![\circ_r]\!]_{i,j} = [\![\bar{x}_r]\!]_{d,k_d}$, where $d = \theta(i,j)$. 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: 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 **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, $\varrho(w)$ is a configuration with the state q_1. Define t_n as the term obtained by truncating t to level n and inserting $[\![s_w]\!]_{\theta_w}$ at node w. We claim that $\mathcal{S} \twoheadrightarrow_{\mathcal{G}} t_n$ for every n.

To prove the claim observe the following. If $top(s_w) = fu_1 \ldots u_n$ (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 (leftmost) reductions $t_1 \to_{\mathcal{G}} t_2 \to_{\mathcal{G}} \cdots$ 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 [8]. The difference between the construction in Section 4 and that in [8] 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 Automata as games

In the previous sections we have considered panic automata as acceptors of trees over some signature. We will now study in more detail the full computation trees of automata. However, in view of the subsequent applications, we will present here an extension of the concept of panic automaton by two features: alternation and ranks.

An *alternating panic automaton with ranks* can be presented by

$$\mathcal{A} = \langle\, \Sigma \cup \{e\}, Q, Q_\exists, \Gamma, q_1, \delta, \bot, \Omega \,\rangle,$$

where e is a fresh symbol of type $\mathbf{0}^2 \to \mathbf{0}$, $Q_\exists \subseteq Q$ is the set of *existential* states, and the function $\Omega : Q \to \omega$ assigns a *rank* $\Omega(q)$ to each state q.

Here $\langle\, \Sigma \cup \{e\}, Q, \Gamma, q_1, \delta, \bot \,\rangle$ is a panic automaton as defined in Section 2 with the only restriction that, for $q \in Q_\exists$ and any $a \in \Gamma$, the corresponding instruction has the form $\delta(q, a) = (e, p_1, p_2)$, for some $p_1, p_2 \in Q$. (Intuitively, this is a nondeterministic choice.) The states in $Q - Q_\exists$ are called *universal*.

The definition of the relation $\to_{\mathcal{A}}$ applies without changes.

Definition 5.1 The *tree of all the computations of* $\mathcal{A}$, denoted $\mathit{Tr}(\mathcal{A})$, is defined by the following conditions.

- The root is labelled by the initial configuration of $\mathcal{A}$.
- If a vertex is labelled by $\langle q, s, \theta\rangle$ and $\delta(q, \mathit{top}(s)) = (f, q_1', \ldots, q_r')$ then there are r sons labelled $\langle q_1', s, \theta\rangle, \ldots, \langle q_r', s, \theta\rangle$ respectively.
- If a vertex is labelled by $\langle q, s, \theta\rangle$ but $\delta(q, \mathit{top}(s))$ is not of the form $(f, q_1', \ldots, q_r')$ then there is only one son labelled by $\langle q', s', \theta'\rangle$ such that $\langle q, s, \theta\rangle \to_{\mathcal{A}} \langle q', s', \theta'\rangle$.

We will now view $\mathit{Tr}(\mathcal{A})$ as an arena of a *parity game*, denoted $\mathit{Gr}(\mathcal{A})$. The game is played by two players, Eve and Adam. The set of positions of the game is $\mathit{dom}\ \mathit{Tr}(\mathcal{A})$. The positions of Eve are those labelled by $\langle q, s, \theta\rangle$ with $q \in Q_\exists$, the remaining ones are positions of Adam. The *rank* of a position labelled by $\langle q, s, \theta\rangle$ is $\Omega(q)$. The players start the play in the root of $\mathit{Tr}(\mathcal{A})$, 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 tree. If a player cannot make a move, this player looses. Otherwise, the result of the play is an infinite path in $\mathit{dom}\ \mathit{Tr}(\mathcal{A})$ labelled by

$\langle q_0, s_0, \theta_0\rangle, \langle q_1, s_1, \theta_1\rangle, \ldots$ (with $\langle q_0, s_0, \theta_0\rangle$ being an initial configuration). Eve wins the play if $\limsup_{n\to\infty} \Omega(q_n)$, i.e., the highest rank repeating infinitely often is even, otherwise Adam is the winner. We refer the reader to [16] and, e.g., [2], for basic introduction to parity games.

Now let $\mathcal{A}$ be an ordinary panic automaton, as defined in Section 2 (it can be viewed as a special case of the above, with $Q_\exists = \emptyset$ and Ω constantly 0). Let $\hat{Tr}(\mathcal{A})$ be a tree with $dom\ \hat{Tr}(\mathcal{A}) = dom\ Tr(\mathcal{A})$ such that $\hat{Tr}(\mathcal{A})(w) = \langle q, top(s)\rangle$, whenever $Tr(\mathcal{A})(w) = \langle q, s, \theta\rangle$. Thus $\hat{Tr}(\mathcal{A})$ projects $Tr(\mathcal{A})$ on the set $Q \times \Gamma$. As the label (q, a) determines the branching degree of a node, we can view $Q \times \Gamma$ as a (finite) signature and $\hat{Tr}(\mathcal{A})$ as a $(Q \times \Gamma)$-tree. We can further consider the monadic second-order theory of this tree as in Section 1 page 4, where we have a predicate for each element of $Q \times \Gamma$.

Remark: It is easy to see that the tree t accepted by the automaton $\mathcal{A}$ (cf. page 8) is MSO-definable within the tree $\hat{Tr}(\mathcal{A})$. Therefore, in order to establish decidability of the MSO theory of t, it is enough to show decidability of the MSO theory of $\hat{Tr}(\mathcal{A})$.

To this end, we will use a translation of MSO logic to automata, introduced already by Rabin [15]. More specifically, let Σ be a finite signature. Then, for any MSO sentence φ over the vocabulary R_Σ (cf. Section 1, page 4), one can construct a non-deterministic parity tree automaton $\mathcal{B}_\varphi$ which accepts precisely the Σ-trees satisfying φ.[1]

Let us briefly recall the concept of a non-deterministic parity tree automaton over a signature Σ. It can be presented as a tuple

$$\mathcal{B} = \langle \Sigma, Q, q_1, \delta, \Omega\rangle$$

where 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 transition relation, $\delta \subseteq \bigcup_{f\in\Sigma}(Q \times \{f\} \times Q^{ar(f)})$, where $f : \mathbf{0}^{ar(f)} \to \mathbf{0}$.

A *run* of the automaton $\mathcal{B}$ on a Σ-tree t is a tree $r : dom\, r \to Q$ with $dom\, r = dom\, t$, consistent with the transition relation. That is, $\langle r(w), f, t(w1), \ldots, t(war(f))\rangle \in \delta$, whenever $t(w) = f$. A run r is *accepting* if $\limsup_{n\to\infty} \Omega(r(v_n))$ is even, for any infinite path $v_0, v_1, v_2, \ldots$ The automaton accepts a tree t if it admits some accepting run on it.

By the considerations above,

> **Problem 1:** Given a 2nd order grammar $\mathcal{G}$ and an MSO sentence φ; decide if the tree $[\![\mathcal{G}]\!]$ satisfies φ?

can be reduced to the problem:

> **Problem 2:** Given a panic automaton $\mathcal{A}$ and a parity tree automaton $\mathcal{B}$ (over signature $Q \times \Gamma$); does $\mathcal{B}$ accept $\hat{Tr}(\mathcal{A})$?

[1] Originally, Rabin showed it for n-ary trees in slightly different setting, but the extension to the above statement is straightforward; it can be found, e.g., in [14].

We will now reduce the last problem to solving the game $Gr(\mathcal{C})$ for a suitable alternating panic automaton $\mathcal{C}$ with ranks. The automaton $\mathcal{C}$ will be obtained by a rather standard product construction. We will use symbols $Q^{\mathcal{A}}$, $Q^{\mathcal{B}}$, $Q^{\mathcal{C}}$ etc. to distinguish between the items of the respective automata. For simplicity, let us assume that, for each state $p \in Q^{\mathcal{B}}$, and for each symbol $(q, a) \in Q^{\mathcal{A}} \times \Gamma^{\mathcal{A}} = \Sigma^{\mathcal{B}}$, the automaton $\mathcal{B}$ has exactly two transitions: $tr_1(p, q, a)$ and $tr_2(p, q, a)$.

The states of $\mathcal{C}$ are $Q^{\mathcal{A}} \times Q^{\mathcal{B}} \cup Q^{\mathcal{A}} \times \delta^{\mathcal{B}}$, and the existential states are precisely $Q^{\mathcal{A}} \times Q^{\mathcal{B}}$. The initial state is, of course, $(q_1^{\mathcal{A}}, q_1^{\mathcal{B}})$. The stack alphabet of $\mathcal{C}$ coincides with the one of $\mathcal{A}$. The signature of $\mathcal{C}$ is $\Sigma \cup \{e\}$. The transitions of $\mathcal{C}$ at the existential states are

$$\delta^{\mathcal{C}}(\langle q, p\rangle, a) = (e, (q, tr_1(p, q, a)), (q, tr_2(p, q, a)))$$

For universal states, the transitions simply follow the transitions of the two automata. It is enough to define $\delta^{\mathcal{C}}((q, \gamma), a)$ for the case where the transition γ of $\mathcal{B}$ corresponds to the symbol (q, a). (By definition, only such states will be reachable.)

If $\delta^{\mathcal{A}}(q, a) = (I, q')$ where I is one of the operations $push_1\langle a' \rangle, push_2, pop_k, panic$, we let

$$\delta^{\mathcal{C}}((q, \gamma), a) = (I, (q', p')) \qquad \text{where } \gamma = (p, (q, a), p').$$

If $\delta^{\mathcal{A}}(q, a) = (f, q_1, \ldots, q_r)$, then (q, a) seen as a signature symbol of the automaton $\mathcal{B}$ is r-ary. In this case we let

$$\delta^{\mathcal{C}}((q, \gamma), a) = (f, (q_1, p_1), \ldots, (q_r, p_r)) \qquad \text{where } \gamma = (p, (q, a), p_1, \ldots, p_r)$$

We claim the following.

Lemma 5.2 Eve has a winning strategy in the game $Gr(\mathcal{C})$ iff the automaton $\mathcal{B}$ accepts the tree $\hat{Tr}(\mathcal{A})$.

Proof: The construction of $\mathcal{C}$ induces an obvious mapping from the arena of $Gr(\mathcal{C})$, i.e., *dom* $Tr(\mathcal{C})$, to *dom* $Tr(\mathcal{A})$. As *dom* $Tr(\mathcal{A}) =$ *dom* $\hat{Tr}(\mathcal{A})$, we can think of it as of a mapping between $Gr(\mathcal{C})$ and $\hat{Tr}(\mathcal{A})$. In particular, an Eve's position labelled by $((q, p), s, \theta)$ in $Tr(\mathcal{C})$ corresponds to a node labelled by (q, s, θ) in $Tr(\mathcal{A})$, and by $(q, top(s))$ in $\hat{Tr}(\mathcal{A})$.

Now, a strategy for Eve in $Gr(\mathcal{C})$ can be viewed as a choice of one of the successors in every position of Eve. This corresponds to a non-deterministic choice of a transition by the automaton $\mathcal{B}$ in its run on $\hat{Tr}(\mathcal{A})$. So, given an accepting run, it is straightforward to construct a winning strategy and *vice versa*. ∎

Henceforth we will be interested in the decidability of:

> **Problem 3:** Given an alternating panic automaton with ranks $\mathcal{A}$; does Eve win the game $Gr(\mathcal{A})$?

6 Making panic automata rank-aware

Rather than solving directly the game $Gr(\mathcal{A})$ introduced in the previous section, we will first show that the problem can be reduced to solving games for automata $\mathcal{A}$ with some additional good properties.

When an automaton makes a panic move in a configuration (q, s, θ), the contents of the pds is brought back to a state 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).

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

Definition 6.1 Consider a node v of $Tr(\mathcal{A})$ labelled with $\langle q, s, \theta\rangle$ such that $panic(s)$ is defined. Let v' be the closest to v ancestor of v labelled 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.

Definition 6.2 A panic automaton $\mathcal{A}$ with ranks is *rank-aware* iff there exists a function $Rank : \Gamma \to \mathrm{Rg}(\Omega)$ such that the panic rank of every node v of $Tr(\mathcal{A})$ labelled (q, s, θ) is equal to $Rank(top(s))$. That is, the panic rank is determined by the top of the stack.

The goal of this section is to show that we can restrict our attention to rank-aware automata.

Lemma 6.3 For every automaton $\mathcal{A}$ with ranks there is a rank-aware automaton $\mathcal{A}'$ such that Eve has a winning strategy in $Gr(\mathcal{A}')$ iff she has one in $Gr(\mathcal{A})$.

The proof of Lemma 6.3 occupies the rest of this section. Consider an alternating panic automaton $\mathcal{A} = \langle\, \Sigma \cup \{e\}, Q, Q_\exists, \Gamma, q_0, \delta, \bot_0, \Omega\,\rangle$. We construct a rank-aware panic automaton $\mathcal{A}' = \langle\, \Sigma \cup \{e\}, Q', Q_\exists, \Gamma', q_0', \delta', \bot_0', \Omega'\,\rangle$

The interpretation of $Tr(\mathcal{A})$ in $Tr(\mathcal{A}')$ is quite straightforward. The idea is that configurations of $\mathcal{A}'$ are like configurations of $\mathcal{A}$ with additional information stored on the stack. In particular, the set of states Q is a subset of Q' and the existential states are the same. The stack alphabet of $\mathcal{A}$ 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 of the automaton $\mathcal{A}'$. Sometimes we describe only the expected behaviour of $\mathcal{A}'$, leaving details to the reader. Below, the notation $\to$ and $\twoheadrightarrow$ abbreviates respectively $\to_{\mathcal{A}'}$ and $\twoheadrightarrow_{\mathcal{A}'}$.

CASE 1: Let $\delta(q, a) = (f, q'_1, \ldots, q'_r)$ for some q and a. Define transitions of the automaton $\mathcal{A}'$ so that for all i (and 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 $m'_p = \max(m_p, \Omega(q'))$ and $m'_l = \max(m_l, \Omega(q'))$.

CASE 2: If $\delta(q, a) = (q', push_1\langle\, b \,\rangle)$ then

$$\delta'(q, (a, m_p, m_l)) = (q', push_1\langle\, b, m'_p, m'_l \,\rangle),$$

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

CASE 3: If $\delta(q, a) = (q', push_2)$ then $\mathcal{A}'$ also executes $push_2$ and adjusts 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: If $\delta(q, a) = (q', pop_1)$ then the new automaton must of course execute a pop_1, but the information at the top must be also adjusted so that we have:

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

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

CASE 5: Let $\delta(q, a) = (q', pop_2)$. The automaton makes of course pop_2 but it must also update the auxiliary information:

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

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

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

Each move of $\mathcal{A}$ is simulated by a sequence of moves of $\mathcal{A}'$, beginning and ending in nodes of the form (q, s, θ), where q is a state of $\mathcal{A}$. We call such nodes the *true nodes* of $Tr(\mathcal{A}')$. During the simulation, $\mathcal{A}'$ uses auxiliary states, not in Q. These auxiliary states are of rank zero and they have no importance for the game. The correctness of the construction is implied by Lemma 6.5 below. 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 $Tr(\mathcal{A})$ labelled with $\langle q, s, \theta\rangle$ and a position $(i, j) \in D(s)$. Let v' be the closest to v ancestor of v labelled with $\langle q', [s_1] \ldots [s_{\theta(i,j)}], \theta'\rangle$; where $s = [s_1] \ldots [s_l]$. 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 $Tr(\mathcal{A}')$ labelled 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) is $\max(\{m_p^{(i,j)}, m_p^{(i,k_i)}\} \cup \{m_l^{(x,k_x)} : x = i+1, \dots, n\})$.

P2 The value of $m_l^{(i,k_i)}$ is the maximal rank of a state visited since the i-th row of s was created, but not after the $i+1$-th row of s was created.

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

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 of $\Omega(q')$, i.e., the rank of the newly seen state.

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 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 state 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 that 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 $m_p^2 = \max\{m_l^{(i,k_i)} : i = d+1, \dots, n\}$, where $d = \theta(n, k_n)$. Thus the settings of m_p' and m_l' are correct. ∎

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

Let $\mathcal{C}$ be an alternating panic automaton with ranks over signature $\Sigma \cup \{e\}$. We assume that $\mathcal{C}$ is rank-aware, that is, we have a function $Rank : \Gamma \to \{0, \ldots, d\}$ such that for any node (q, s, θ) of $Tr(\mathcal{A})$, the value of $Rank(top(s))$ is the highest rank of a state visited between the panic ancestor and the current node.

We transform the game $Gr(\mathcal{C})$ into a yet another game $Game(\mathcal{C})$. The advantage of the latter game is that we are able to define it within a 2-tree – a structure whose MSO theory is known to be decidable.

A 2-tree is a structure obtained from an ordinary tree by the $\star$-operation considered[2] in [18]. In general, for a structure $M = \langle D, r, \ldots \rangle$ with a universe D, over some relational vocabulary $\{\underline{r}, \ldots\}$, the $\star$-operation creates a new logical structure

$$M^\star = \langle D^*, son, clone, r^\star, \ldots \rangle$$

over the universe D^* (the free monoid generated by D). Here $son(w, wd)$ and $clone(wd, wdd)$ hold for every $w \in D^*$ and $d \in D$, and $r^\star(wd_1, \ldots, wd_k)$ iff $r(d_1, \ldots, d_k)$. It is shown in [18] that if the MSO theory of M is decidable, so is the MSO theory of $M^\star$.

Intuitively, if $T \subseteq X^*$ is a tree then the 2-tree $T^\star$ is a tree of trees.[3] Its nodes can be viewed as the 2-stacks over X (cf. Section 2). By the definition of a 2-tree, there are edges from $v[w]$ to $v[wx]$ for every $x \in X$; and there are also edges from $v[w]$ to $v[w][w]$ and to v. These edges permit to define basic stack operations (without *panic*) in the 2-tree.

To explain the construction to follow, let us first recall how the computations of a level 1 pushdown automaton can be coded into a full tree $(\Gamma \cup Q)^*$, where Γ is the stack alphabet and Q is the set of states. A configuration is represented as a string from $\Gamma^* Q$. We put an edge from waq to $waa'q'$ if there is a transition $\delta(q, a) = (q', push(a'))$ in the automaton. For a transition $\delta(q, a) = (q', pop)$, we put an edge from waq to wq'. This way we interpret in the tree the graph of configurations of a pushdown automaton. This definition is clearly formalizable in the MSO logic, hence by the Rabin Tree Theorem, the MSO theory of the graph of configurations of a pushdown automaton is decidable.

For an ordinary 2-pushdown automaton (i.e. one without panic moves) we proceed similarly but now in 2-trees. A configuration is represented as a sequence $(\Gamma^*)^*(\Gamma^* Q)$. A push move $\delta(q, a) = (q', push(a'))$ gives rise to an edge from $v[waq]$ to $v[waa'q']$. Similarly for *pop*. Now, a transition $\delta(q, a) = (q', push_2)$ is represented by an edge from $v[waq]$ to $v[wa][waq']$. A transition $\delta(q, a) = (q', pop_2)$ gives rise to an edge from $v[w'][waq]$ to $v[w'q']$. These new transitions are definable in MSO logic thanks to the additional relations we have in 2-tree. As the MSO-theory of a 2-tree $((\Gamma \cup Q)^*)^\star$ is decidable [18], we get the decidability result for the pushdown graphs of 2-stack automata.

The method does not work directly for panic automata because a move $\delta(q, a) = (q', panic)$ would require an edge from a node $v[waq]$ to $v'[w'q']$ where $v'[w']$ is the prefix of $v[w]$ deter-

[2] The idea of this operation can be traced back to Shelah, Semenov, and A.A. Muchnik (cf. [18]).

[3] Of course, $T^\star$ should not be confused with the subset of X^* obtained by the closure of T under the language-theoretical star operation.

mined by the panic distribution. It seems to be difficult (impossible?) to code panic distribution information directly into configurations. Therefore, we take here another approach, based on alternation: a nondeterministic panic prediction followed by a universal verification.

Let us explain the idea of panic prediction. When we perform a $push_2$ move, then we get a new possible place where panic moves can return, namely the row, say l-th, just below the newly created $(l+1)$-st row. At this moment we guess all possible states with which we will do a panic return to the l-th row. (Actually we need to know also the maximal rank of a state seen in the meantime, so we will guess an element of $Ret = Q \dot{\to} \{0, \ldots, d\}$, rather than just a subset of Q. But for a moment let us assume that it is just a set of states that we are guessing.) Guessing a set R' allows us to check *at the same time as simulating the* $push_2$ *move*, what will happen when we return to the stack of height l with a state from R'. We can do this because now we have stack of height l at hand (and this is what we will miss while making the panic move). Suppose that we have checked that, from all states in R', the behaviour of the automaton is correct. We can now keep this R' as a guarantee for all the positions from which we will want to do a panic move to the stack of height l. Whenever in such a position the automaton wants to make a panic move $(q', panic)$, we do not simulate this move at all, but simply check if q' is in the set of predicted states, i.e., in R'. If so, we know that we have explored the future of the computation already, so we can stop here and accept. If not, we have incorrectly predicted the set of returns and we abort the whole process. Hence, it is enough to keep the right R' in the right positions. This is what is happening below and this is why we change the stack alphabet to $\Gamma \times Ret$. The states are also from $Q \times Ret$, as we want to keep the last prediction handy in case we want to create a new position.

Before we describe the construction we introduce a useful ordering on ranks:

$$m \preccurlyeq n \quad \text{iff} \quad \begin{cases} m \leq n \text{ and both even} \\ m \geq n \text{ and both odd} \\ m \text{ odd and } n \text{ even} \end{cases}$$

Intuitively if $m \preccurlyeq n$ then m is a rank at most as good as n with respect to the acceptance condition. The least element in this order is the largest odd number in $\{0, \ldots, d\}$.

The set of returns can now be defined as:

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

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 $Game(\mathcal{C})$ over a 2-tree $(X^*)^\star$, where the underlying set X is

$$(\Gamma \times Ret) \ \cup \ (Q \times Ret) \ \cup \ Ret \ \cup \ \{\bot, \top\} \ \cup \ \{push_2(q, b) : q \in Q, b \in Ret \cup \{?\}\}$$

The positions of the game are just the elements of $(X^*)^\star$.

We are ready to define the moves of the game. They will correspond to the possible moves of $\mathcal{C}$. From a vertex $s[w(a, R)(q, R')]$ we put edges:

- to $s[w(a,R)(q_i,R')]$ for $\delta(q,a)=(f,q_1,\ldots,q_r)$, $i=1,\ldots,r$, where $f\in\Sigma\cup\{e\}$;
- to $s[w(a,R)(a',R')(q',R')]$ for $\delta(q,a)=(q',\mathit{push}(a'))$;
- to $s[w(a,R)R'][w(a,R)\mathit{push}_2(q',?)]$ for $\delta(q,a)=(q',\mathit{push}_2)$;
- to $s[w(q',R')]$ for $\delta(q,a)=(q',\mathit{pop}_1)$;
- to $s'[w''(q',R'')]$ for $\delta(q,a)=(q',\mathit{pop}_2)$, where $s=s'[w''R'']$;
- to $\top$ for $\delta(q,a)=(q',\mathit{panic})$ if $\mathit{Rank}(a)\succcurlyeq R(q')$;
- to $\bot$ for $\delta(q,a)=(q',\mathit{panic})$ if $\mathit{Rank}(a)\not\succcurlyeq R(q')$ or $R(q')$ not defined.

The intuition behind the transition to $\top$ is that if the priority seen since panic ancestor, which is given by $\mathit{Rank}(a)$, is not worse than previewed by $R(q')$ then we can just accept without further verification.

From a vertex $s[w\cdot\mathit{push}_2(q',?)]$ there is an edge to $s[w\cdot\mathit{push}_2(q',R')]$ for every $R'\in\mathit{Ret}$.

From a vertex $s[wR][w\cdot\mathit{push}_2(q',R')]$ we have the edges to:

- $s[wR][w(q',R')]$; and
- $s[w(q'',R)]$ for every $q''\in\mathit{dom}\,R'$ (recall that R' is a partial function).

This completes the definition of the underlying graph of the game $\mathit{Game}(\mathcal{C})$.

The vertices of Eve will be:

- the vertices ending in $\mathit{push}_2(q,?)$;
- the vertices where the rule (e,q_0,q_1) was applied (note that these vertices are well defined as the automaton is deterministic).

A finite play terminating in position $\top$ is won by Eve, while in $\bot$ Adam is the winner.

Finally, we need to define the winning condition for infinite plays. This will be again a parity condition, but this time given in terms of ranks associated to the edges of the game graph.

Definition 7.1 A *rank of an edge* is defined by the following clauses:

- if it links $s[wR][w\cdot\mathit{push}_2(q',R')]$ to $s[w(q'',R)]$ then the rank is $R'(q'')$;
- for all other edges it is the rank of the state in source node of the edge.

Lemma 7.2 For a given rank-aware alternating panic automaton $\mathcal{C}$, it is decidable if Eve has a winning strategy in $\mathit{Game}(\mathcal{C})$.

Proof: By construction, the arena of the game, i.e., the move relation and ranking of edges are definable in the MSO theory of the 2-tree $(X^*)^\star$. It is well known that the set of winning positions in a parity game is MSO definable in the arena; it follows in particular from the μ-calculus definition of this set (see, e.g., [2]). Then the result follows from the decidability of the MSO theory of the 2-tree $(X^*)^\star$ [18]. ∎

Theorem 7.3 *Eve wins in $Game(\mathcal{C})$ iff she wins in $Gr(\mathcal{C})$.*

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

Definition 7.4 We say that a vertex $v = [w_1R_1]\dots[w_kR_k][w_{k+1}(q,R_{k+1})]$ *represents* a configuration (q,s,θ) if

- $s = [s_1]\dots[s_{k+1}]$ and for all i, $s_i = w_i\downarrow_1$, i.e. s 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).
- for every $(i,j) \in D(s)$ we have $R_{i,j} = R_{\theta(i,j)+1}$, where $R_{i,j}$ is the second component of the symbol occurring in v at the position (i,j).

Definition 7.5 Let n, o, be nodes of $Gr(\mathcal{C})$. 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 labelling 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 7.6 If Adam wins in $Gr(\mathcal{C})$ then he wins in $Game(\mathcal{C})$.

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

Suppose that the play is in the position $v[w(q,R)] = [w_1R_1]\dots[w_kR_k][w_{k+1}(q,R_{k+1})]$ and there is an associated node n of $Gr(\mathcal{C})$ with the following properties:

- v represents the label of n which is (q,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 $\rho \not\succeq R_l(q')$.

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 $Game(\mathcal{C})$, Adam must move to $v[wR][w \cdot push_2(q', ?)]$. Now Eve chooses a position $v[wR][w \cdot push_2(q', R')]$. 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 $v[w(q'', R)]$ and the associated node becomes o', the successor of o. If it is not the case then Adam chooses the position $v[wR][w(q', R')]$ 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 7.7 If Eve wins in $Gr(\mathcal{C})$ then Eve wins in $Game(\mathcal{C})$.

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

We say that $v = [w_1R_1] \dots [w_kR_k][w_{k+1}(q, R_{k+1})]$ *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, that is to say what set to choose for each play ending either in a position of the form $v[wR][w \cdot push(q', ?)]$, or in a position of the form $v[w(q, R)]$ where the move (E, q_0, q_1) is played. To do this we will associate to each such position a node n of $Gr(\mathcal{A})$ such that

- the position represents the label of n;
- all the panic jumps over n are covered by the position.

Suppose that we have a position $v[w(q, R)]$ and an associated node n. Let (q, s, θ) be the label of n. We consider the case when $\delta(q, top(s)) = (q', push_2)$, the rest being easy.

Suppose $\delta(q, top(s)) = (q', push_2)$. We have in this case a unique successor of $v[w(q, R)]$ which is $v[w(a, R)][w \cdot push_2(q', ?)]$. Now it is the time for Eve to respond with R'. We put

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

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

Now, if Adam chooses $v[wR][w(q', R')]$ then we associate to this position the successor n' of n. If, on the other hand, Adam chooses $v[w(q'', R)]$, for some $q'' \in dom\, R'$, then we know that 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.

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

This completes the proof of Theorem 7.3.

Take a hyperalgebraic (i.e., level 2) grammar $\mathcal{G}$ and an MSOL formula φ. We want to decide if $[\![\mathcal{G}]\!] \vDash \varphi$ holds; recall that $[\![\mathcal{G}]\!]$ is the tree generated by $\mathcal{G}$. By Proposition 4.7 we have a panic automaton $\mathcal{A}$ accepting precisely $[\![\mathcal{G}]\!]$. Using the transformation discussed in Section 5 we reduce the problem to finding the winner in the game $Gr(\mathcal{C})$ for an alternating panic automaton with ranks $\mathcal{C}$ constructed from $\mathcal{A}$ and φ. By Lemma 6.3 we can assume that $\mathcal{C}$ is rank-aware. From $\mathcal{C}$ we construct a new game $Game(\mathcal{C})$ and know by Lemma 7.2 that finding the winner in this game is decidable. By Theorem 7.3, the same player wins in $Gr(\mathcal{C})$. Thus we obtain our main result.

Theorem 7.8 *The MSO theory of any hyperalgebraic tree is decidable.*

References

[1] Aehlig, K., Private communication.

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

[3] 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.

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

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

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

[7] 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.

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

[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] 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.

[11] 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.

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

[13] de Miranda J.G., and Ong, L., Safety is not a restriction at level 2 for string languages. Manuscript, 2004.

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

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

[16] 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.

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

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