Collapsible Pushdown Automata and Labeled Recursion Schemes Equivalence, Safety and Effective Selection

Arnaud Carayol
LIGM (Université Paris Est & CNRS), Paris, France

Olivier Serre
LIAFA (Université Paris Diderot – Paris 7 & CNRS), Paris,France

***Abstract*—Higher-order recursion schemes are rewriting systems for simply typed terms and they are known to be equi-expressive with collapsible pushdown automata (CPDA) for generating trees. We argue that CPDA are an essential model when working with recursion schemes. First, we give a new proof of the translation of schemes into CPDA that does not appeal to game semantics. Second, we show that this translation permits to revisit the safety constraint and allows CPDA to be seen as Krivine machines. Finally, we show that CPDA permit one to prove the effective MSO selection property for schemes, subsuming all known decidability results for MSO on schemes.**

***Keywords*-Recursion Schemes, Collapsible Pushdown Automata, Safety Constraint, MSO Effective Selection**

I. Introduction

Higher-order recursion schemes are rewriting systems for simply typed terms and in recent years they have received much attention as a method of constructing rich and robust classes of possibly infinite ranked trees. Remarkably these trees have decidable monadic second-order (MSO) theories, subsuming most of the examples of structures for which MSO is decidable. Since the original proof of Ong [15] based on traversals (a tool from game semantics), several alternative proofs (and extensions) were obtained using different techniques: automata [9], [2], intersection types [13], the Krivine machine [18].

In this article we focus on the automata approach. In [9], schemes were shown to be equi-expressive with an extension of the standard model of pushdown automata, called collapsible pushdown automata (CPDA). The translation from schemes into CPDA crucially relied on traversals. The decidability of MSO was obtained by solving parity games played on transition graphs of CPDA. In [2] a refinement of this proof was used to show that the family of trees generated by schemes is MSO-reflective, *i.e.* for any scheme $\mathcal{S}$ and any MSO formula $\varphi(x)$ with one first-order free variable x, one can build another scheme that produces the same tree as $\mathcal{S}$ except that now all nodes satisfying $\varphi(x)$ are marked.

In this article, we focus on the merits of CPDA for studying recursion schemes. As CPDA are more naturally associated with a labeled transition system (LTS) than with a tree, we introduce a variant of recursion schemes, *labeled recursion schemes*, that admit a canonical LTS. In both cases, the tree generated is simply the unfolding of the LTS. Although not technically difficult, we think that this notion and the associated family of LTS can be the subject of further studies.

Our first main result is a *simplified* and *syntactic* proof of the translation of a scheme into an equivalent CPDA. This is the first proof of the equi-expressivity result of [9] that *does not* use game semantics. A comparison of the obtained CPDA can be found at the beginning of Section III.

Furthermore this translation also permits one to view a CPDA as a Krivine machine, hence inheriting the simplified proof of [18] for decidability of μ-calculus model-checking.

We also show that when translating a *safe* scheme we obtain a CPDA that does not need to use the links. This result, independently obtained by Blum and Broadbent [1], unifies the work of [10] on safe schemes and sheds a new light on safety. As a spin-off result, we give a more natural definition of safety based on Damm's original work [6].

Finally, the true gain of the apparently more involved CPDA model is demonstrated by showing that the trees defined by recursion schemes enjoy the effective MSO selection property: for any scheme $\mathcal{S}$ and any formula $\exists X\, \varphi(X)$ if the tree t generated by $\mathcal{S}$ satisfies $\exists X\, \varphi(X)$, one can build another scheme generating the tree t where a set of nodes U satisfying $\varphi(X)$ is marked. This new result subsumes all previously known MSO-decidability results on recursion schemes (while keeping the same complexity, in particular the one of [13]) and relies on a careful analysis of the winning strategies in CPDA parity games.

II. Preliminaries

A. Trees and Terms

Let A be a finite alphabet. We denote by A^* the set of finite words over A. A tree t (with directions in A) is a non-empty prefix-closed subset of A^*. Elements of t are called *nodes* and ε is called the *root* of t. For any node $u \in t$ and any direction $a \in A$, we refer to ua, when it belongs to t, as the a*-child* of u. A node with no child is a *leaf.*

A *ranked alphabet* A is an alphabet that comes together with an arity function, $\varrho : A \to \mathbb{N}$. The *terms* built over a ranked alphabet A are those trees with directions $\overrightarrow{A} \stackrel{\text{def}}{=} \bigcup_{f \in A} \overrightarrow{f}$ where $\overrightarrow{f} = \{f_1, \ldots, f_{\varrho(f)}\}$ if $\varrho(f) > 0$ and $\overrightarrow{f} = \{f\}$ if $\varrho(f) = 0$. For a tree t with directions in $\overrightarrow{A}$ to be a

term, we require, for all nodes u, that the set $A_u = \{d \in \overrightarrow{A} \mid ud \in t\}$ is empty iff u ends with some $f \in A$ (hence $\varrho(f) = 0$) and if A_u is non-empty then it is equal to some $\overrightarrow{f} \in \overrightarrow{A}$. We denote by $\mathrm{Terms}(A)$ the set of terms over A.

For $c \in A$ of arity 0, we denote by c the term $\{\varepsilon, c\}$. For $f \in A$ of arity $n > 0$ and for terms $t_1, \ldots, t_n$, we denote by $f(t_1, \ldots, t_n)$ the term $\{\varepsilon\} \cup \bigcup_{i \in [1,n]} \{f_i\} \cdot t_i$. These notions are illustrated in Figure 1.

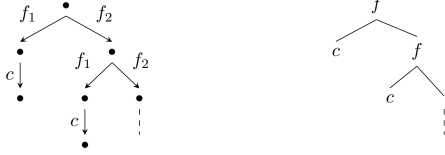


Figure 1. Two representations of the infinite term $f_2^*\{f_1 c, f_1, \varepsilon\} = f(c, f(c, f(\cdots)))$ over the ranked alphabet $\{f, c\}$ assuming that $\varrho(f) = 2$ and $\varrho(c) = 0$.

B. Labeled Transition Systems

A rooted labeled transition system (LTS for short) is an edge-labeled directed graph with a distinguished vertex, called the root. When considering LTS associated with computational models, it is usual to allow silent transitions. The symbol for silent transitions is usually ε but here, to avoid confusion with the empty word, we will instead use e. We forbid a vertex to be the source of both a silent transition and of a non-silent transition. When Σ is an alphabet we let $\Sigma_e = \Sigma \backslash \{e\}$.

Formally, a *rooted labeled transition system with silent transitions* $\mathcal{L}$ is a tuple $\langle D, r, \Sigma, (\xrightarrow{a})_{a \in \Sigma} \rangle$ where D is a finite or countable set called the *domain*, $r \in D$ is a distinguished element called the *root*, Σ is a finite set of *labels* that contains a distinguished symbol denoted e and for all $a \in \Sigma$, $\xrightarrow{a} \subseteq D \times D$ is a binary relation on D.

For any $a \in \Sigma$ and any $(s,t) \in D^2$ we write $s \xrightarrow{a} t$ to indicate that $(s,t) \in \xrightarrow{a}$, and we refer to it as an a-*transition* with *source* s and *target* t. Moreover, we require that for all $s \in D$, if s is the source of a e-transition, then s is not the source of any a-transition with $a \neq e$. For a word $w = a_1 \cdots a_n \in \Sigma^*$, we define a binary relation $\xrightarrow{w}$ on D by letting $s \xrightarrow{w} t$ (meaning that $(s,t) \in \xrightarrow{w}$) if there exists a sequence $s_0, \ldots, s_n$ of elements in D such that $s_0 = s$, $s_n = t$, and for all $i \in [1,n]$, $s_{i-1} \xrightarrow{a_i} s_i$. These definitions are extended to languages over Σ by taking, for all $L \subseteq \Sigma^*$, the relation $\xrightarrow{L}$ to be the union of all $\xrightarrow{w}$ for $w \in L$.

For all words $w = a_1 \cdots a_n \in \Sigma_e^*$, we denote by $\stackrel{w}{\Longrightarrow}$ the relation $\xrightarrow{L_w}$ where $L_w \stackrel{\text{def}}{=} e^* a_1 e^* \cdots e^* a_n e^*$ is the set of words over Σ obtained by inserting arbitrarily many occurrences of e in w.

An LTS is said to be *deterministic* if for all s, t_1 and t_2 in D and all a in Σ, if $s \xrightarrow{a} t_1$ and $s \xrightarrow{a} t_2$ then $t_1 = t_2$.

Caveat 1. *From now on, we always assume that the* LTS *we consider are deterministic.*

We associate a tree to every LTS $\mathcal{L}$, denoted $\mathrm{Tree}(\mathcal{L})$, with directions in Σ_e, reflecting the possible behaviours of $\mathcal{L}$ starting from the root. For this we let $\mathrm{Tree}(\mathcal{L}) \stackrel{\text{def}}{=} \{w \in \Sigma_e^* \mid \exists s \in D,\, r \stackrel{w}{\Longrightarrow} s\}$. As $\mathcal{L}$ is deterministic, $\mathrm{Tree}(\mathcal{L})$ is obtained by unfolding the underlying graph of $\mathcal{L}$ from its root and contracting all e-transitions. Figure 2 presents an LTS with silent transitions together with its associated tree $\mathrm{Tree}(\mathcal{L})$.

As illustrated in Figure 2, the tree $\mathrm{Tree}(\mathcal{L})$ does not reflect the diverging behaviours of $\mathcal{L}$ (*i.e.* the ability to perform an infinite sequence of silent transitions). For instance in the LTS of Figure 2, the vertex s diverges whereas the vertex t does not. A more informative tree can be defined in which diverging behaviours are indicated by a $\perp$-child for some fresh symbol $\perp$. This tree, denoted $\mathrm{Tree}^\perp(\mathcal{L})$, is defined by letting $\mathrm{Tree}^\perp(\mathcal{L}) \stackrel{\text{def}}{=} \mathrm{Tree}(\mathcal{L}) \cup \{w\perp \in \Sigma_e^* \perp \mid \forall n \geqslant 0, r \stackrel{we^n}{\Longrightarrow} s_n \text{ for some } s_n\}$.

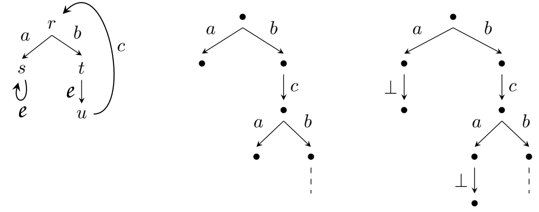


Figure 2. An LTS $\mathcal{L}$ with silent transitions of root r (on the left), the tree $\mathrm{Tree}(\mathcal{L})$ (in the center) and the tree $\mathrm{Tree}^\perp(\mathcal{L})$ (on the right).

C. Types, Applicative Terms

Types are generated by the grammar $\tau ::= o \mid \tau \to \tau$. Every type $\tau \neq o$ can be uniquely written as $\tau_1 \to (\tau_2 \to \cdots (\tau_n \to o) \ldots)$ where $n \geqslant 0$ and $\tau_1, \ldots, \tau_n$ are types. The number n is the *arity* of the type and is denoted by $\varrho(\tau)$. To simplify the notation, we take the convention that the arrow is associative to the right and we write $\tau_1 \to \cdots \to \tau_n \to o$ (or $(\tau_1, \ldots, \tau_n, o)$ to save space).

The *order* measures the nesting of a type: $ord(o) = 0$ and $ord(\tau_1 \to \tau_2) = \max(ord(\tau_1) + 1, ord(\tau_2))$.

Let X be a set of typed symbols. Every symbol $f \in X$ has associated a type τ; we write $f : \tau$ to mean that f has type τ. The set of *applicative terms of type* τ *generated from* X, denoted $\mathrm{Terms}_\tau(X)$, is defined by induction over the following rules. If $f : \tau$ is an element of X then $f \in \mathrm{Terms}_\tau(X)$; if $s \in \mathrm{Terms}_{\tau_1 \to \tau_2}(X)$ and $t \in \mathrm{Terms}_{\tau_1}(X)$ then the applicative term obtained by applying s to t, denoted $s\,t$, belongs to $\mathrm{Terms}_{\tau_2}(X)$. For every applicative term t, and every type τ, we write $t : \tau$ to mean that t is an applicative term of type τ. By convention, the application is considered to be left-associative, thus we write $t_1 t_2 t_3$ instead of $(t_1 t_2) t_3$.

Example 1. *Assuming that* $f : (\mathrm{o} \to \mathrm{o}) \to \mathrm{o} \to \mathrm{o}$, $g : \mathrm{o} \to \mathrm{o}$ *and* $c : \mathrm{o}$, *we have* $g\,c : \mathrm{o}$, $f\,g : \mathrm{o} \to \mathrm{o}$, $f\,g\,c = (f\,g)\,c : \mathrm{o}$ *and* $f\,(f\,g)\,c : \mathrm{o}$.

The set of subterms of t, denoted $\mathrm{Subs}(t)$, is inductively defined by $\mathrm{Subs}(f) = \{f\}$ for $f \in X$ and $\mathrm{Subs}(t_1\,t_2) = \mathrm{Subs}(t_1) \cup \mathrm{Subs}(t_2) \cup \{t_1\,t_2\}$. The subterms of the term $f\,(f\,g)\,c : \mathrm{o}$ in Example 1 are $f\,(f\,g)\,c$, f, $f\,g$, $f\,(f\,g)$, c and g. A less permissive notion is that of *argument subterms* of t, denoted $\mathrm{ASubs}(t)$, which only keep those subterms that appear as an argument. The set $\mathrm{ASubs}(t)$ is inductively defined by letting $\mathrm{ASubs}(t_1 t_2) = \mathrm{ASubs}(t_1) \cup \mathrm{ASubs}(t_2) \cup \{t_2\}$ and $\mathrm{ASubs}(f) = \varnothing$ for $f \in X$. In particular if $t = F t_1 \cdots t_n$, $\mathrm{ASubs}(t) = \cup_{i=1}^{n}(\mathrm{ASubs}(t_i) \cup \{t_i\})$. The argument subterms of $f\,(f\,g)\,c : \mathrm{o}$ are $f\,g$, c and g. In particular, for all terms t, one has $|\mathrm{ASubs}(t)| < |t|$ (the size $|t|$ of a term is the length of the word representation of t).

Remark 1. *A ranked alphabet A can be seen as a typed alphabet by assigning to every symbol f of A the type* $\underbrace{o \to \cdots \to o \to}_{\varrho(f)} o$. *In particular, every symbol in A has order* 0 *or* 1. *The finite terms over A (seen as a ranked alphabet) are in bijection with the applicative ground terms over A (seen as a typed alphabet).*

D. Labeled Recursion Schemes

Recursion schemes are grammars for simply typed terms, and they are often used to generate a possibly infinite term. Traditionally, recursion schemes are not associated with an LTS. Here we provide an alternative definition based on LTS.

For each type τ, we assume an infinite set V_τ of variables of type τ, such that V_{τ_1} and V_{τ_2} are disjoint whenever $\tau_1 \neq \tau_2$, and we write V for the union of those sets V_τ as τ ranges over types. We use letters $x, y, \varphi, \psi, \ldots$ to range over variables.

A deterministic *labeled recursion scheme* is a 5-tuple $\mathcal{S} = \langle \Sigma, N, \mathcal{R}, Z, \bot \rangle$ where

- Σ is a finite set of labels and $\bot$ is a distinguished symbol in Σ,
- N is a finite set of typed *non-terminals*; we use upper-case letters $F, G, H, \ldots$ to range over non-terminals,
- $Z : \mathrm{o} \in N$ is a distinguished *initial symbol* which does not appear in any right-hand side,
- $\mathcal{R}$ is a finite set of *production rules* of the form

$$F\,x_1\,\cdots\,x_n \xrightarrow{a} e$$

where $a \in \Sigma\backslash\{\bot\}$, $F : (\tau_1, \cdots, \tau_n, o) \in N$, the x_is are distinct variables, each x_i is of type τ_i, and e is a ground term over $(N\backslash\{Z\}) \cup \{x_1, \ldots, x_n\}$.
In addition, we require that there is at most one production rule starting with a given non-terminal and labeled by a given symbol.

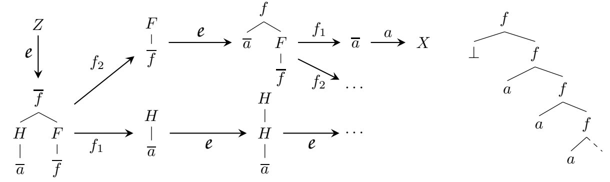


Figure 3. The LTS and the tree associated with the scheme $\mathcal{S}$ of Example 2.

The LTS associated with $\mathcal{S}$ has the set of ground terms over N as domain, the initial symbol Z as root, and, for all $a \in \Sigma$, the relation $\xrightarrow{a}$ is defined by:

$$F\,t_1 \ldots t_{\varrho(F)} \xrightarrow{a} e[t_1/x_1, \ldots, t_{\varrho(F)}/x_{\varrho(F)}]$$

if $F\,x_1\,\cdots\,x_n \xrightarrow{a} e$ is a production rule.

The tree generated by a labeled recursion scheme $\mathcal{S}$, denoted $\mathrm{Tree}^\bot(\mathcal{S})$, is the tree $\mathrm{Tree}^\bot$ of its associated LTS. To use labeled recursion schemes to generate terms over ranked alphabet A, it is enough to enforce that for every non-terminal $F \in N$:

- either there is a unique production starting with F which is labeled by e,
- or there is a unique production starting with F which is labeled by some symbol c of arity 0 and whose right-hand side starts with a non-terminal that comes with no production rule in the scheme,
- or there exists a symbol $f \in A$ with $\varrho(f) > 0$ such that the set of labels of production rules starting with F is exactly $\overrightarrow{f}$.

Example 2. *Consider the order-*1 *scheme* $\mathcal{S} = \langle \Sigma, N, \mathcal{R}, Z, \bot \rangle$ *where* $\Sigma = \{a, f_1, f_2, \bot\}$, N *consists of* $Z, X, \overline{a} : o$, $H : (o, o)$, $\overline{f} : (o, o, o)$ *and* $F : ((o, o, o), o)$, *and* $\mathcal{R}$ *is given below*

$$\begin{array}{lclclcl} Z & \xrightarrow{e} & \overline{f}\,(H\,\overline{a})\,(F\,\overline{f}) & \quad & \overline{a} & \xrightarrow{a} & X \\ H\,z & \xrightarrow{e} & H\,(H\,z) & & \overline{f}\,x\,y & \xrightarrow{f_1} & x \\ F\,\varphi & \xrightarrow{e} & \varphi\,\overline{a}\,(F\,\varphi) & & \overline{f}\,x\,y & \xrightarrow{f_2} & y \end{array}$$

The LTS *and the tree associated with* $\mathcal{S}$ *are depicted in Figure 3.*

Remark 2. *A more standard definition of* recursion schemes *[9] comes with a ranked alphabet A of* terminal symbols *that can be used in the right hand side of the rewriting rules; moreover the rules are no longer labeled. Applying rewriting rules from the initial symbol one derives finite terms over the set of terminal and non-terminal symbols. Replacing in such a term t any non-terminal, together with its argument, by a fresh symbol $\bot : \mathrm{o}$ leads a term $t^\bot$ over $A \cup \{\bot\}$. As the rewriting is confluent, there exists a supremum of all terms $t^\bot$ where t ranges over terms that can be rewritten from the initial symbol, and this (possibly infinite) term is defined as the* value term *of the scheme.*

It is easily seen that labeled recursion schemes and (usual) recursions schemes generate the same terms; the translations are linear and preserve both order and arity.

E. Examples of Trees Defined by Labeled Recursion Schemes

We provide some examples of trees defined by labeled recursion schemes. Given a language L over Σ, we denote by $\mathrm{Pref}(L)$ the tree containing all prefixes of words in L.

Example 3. *Using order-2 schemes, it is possible to go beyond deterministic context-free languages and to define for instance the tree $T_1 = \mathrm{Pref}(\{a^n b^n c^n \mid n \geqslant 0\})$. Consider for instance the order-2 scheme $\mathcal{S}_1$ given by:*

$$\begin{array}{lcl@{\qquad}lcl} Z & \xrightarrow{a} & F\,I\,(\mathrm{K}\,C\,I) & B\,x & \xrightarrow{b} & x \\ F\,\varphi\,\psi & \xrightarrow{a} & F\,(\mathrm{K}\,B\,\varphi)\,(\mathrm{K}\,C\,\psi) & C\,x & \xrightarrow{c} & x \\ F\,\varphi\,\psi & \xrightarrow{b} & \varphi(\psi\,X) & I\,x & \xrightarrow{e} & x \\ \mathrm{K}\,\varphi\,\psi\,x & \xrightarrow{e} & \varphi(\psi(x)) & & & \end{array}$$

with $Z, X : o$, $B, C, I : o \to o$, $F : ((o \to o), (o \to o), o)$ and $\mathrm{K} : ((o \to o), (o \to o), o, o)$.

*Intuitively, the non-terminal K plays the role of the composition of functions of type $o \to o$ (*i.e.* for any terms $F_1, F_2 : o \to o$ and $t : o$, $\mathrm{K}\,F_1\,F_2 t \xrightarrow{e} F_1(F_2\,t)$). For any term $G : o \to o$, we define G^n for all $n \geqslant 0$ by taking $G^0 = I$ and $G^{n+1} = \mathrm{K}\,G\,G^n$. For any ground term t, $G^n\,t$ behaves as $\underbrace{G(\ldots(G}_{n}(It))\ldots)$ and in particular $B^n\,X \xRightarrow{b^n} X$. For all $n \geqslant 0$, we have: $Z \xrightarrow{a^n} F\,B^{n-1}\,C^n \xrightarrow{b} B^{n-1}(C^n\,X) \xRightarrow{b^{n-1}c^n} X$.*

Example 4. *We present a tree T_U proposed by Urzyczyn which exemplify the full expressivity of order-2 schemes (see Section IV). The tree T_U has directions in $\{\,(,),\star\,\}$. A word over $\{\,(,)\,\}$ is* well bracketed *if it has as many opening brackets as closing brackets and if for every prefix the number of opening brackets is not smaller than the number of closing brackets.*

The language U is defined as the set of words of the form $w\star^n$ where w is a prefix of a well-bracketed word and n is equal to $|w| - |u| + 1$ where u is the longest suffix of w which is well-bracketed. In other words, n equals 1 if w is well-bracketed, and otherwise it is equal to the index of the last unmatched opening bracket plus one.

For instance, the words $()((())\star\star\star\star$ and $()()()\star$ belong to U. The tree T_U is simply $\mathrm{Pref}(U)$. The following scheme $\mathcal{S}_U$ generates T_U.

$$\begin{array}{lcl@{\qquad}lcl} Z & \xrightarrow{e} & G\,(H\,X) & F\,\varphi\,x\,y & \xrightarrow{(} & F\,(F\varphi x)\,y\,(Hy) \\ G\,z & \xrightarrow{(} & F\,G\,z\,(Hz) & F\,\varphi\,x\,y & \xrightarrow{)} & \varphi\,(H\,y) \\ G\,z & \xrightarrow{\star} & X & F\,\varphi\,x\,y & \xrightarrow{\star} & x \\ H\,u & \xrightarrow{\star} & u & & & \end{array}$$

with $Z, X : o$, $G, H : o \to o$ and $F : (o \to o, o, o)$.

To better explain the inner workings of this scheme, let us introduce some syntactic sugar. With every integer, we associate a ground term by letting $\mathbf{0} = X$ and, for all $n \geqslant 0$, $\mathbf{n+1} = H\,\mathbf{n}$. With every sequence $[\mathbf{n_1} \ldots \mathbf{n}_\ell]$ of integers, we associate a term of type $o \to o$ by letting $[\,] = G$ and $[\mathbf{n_1} \ldots \mathbf{n}_\ell\,\mathbf{n}_{\ell+1}] = F\,[\mathbf{n_1} \ldots \mathbf{n}_\ell]\,\mathbf{n}_{\ell+1}$. Finally we write $([\mathbf{n_1} \ldots \mathbf{n}_\ell], \mathbf{n})$ to denote the ground term $[\mathbf{n_1} \ldots \mathbf{n}_\ell]\,\mathbf{n}$.

The scheme can be revisited as follows (note that the two rules labelled by (are now merged):

$$\begin{array}{l} Z \xrightarrow{e} ([\,], \mathbf{1}) \qquad ([\,], \mathbf{n}) \xrightarrow{\star} \mathbf{0} \\ ([\mathbf{n_1} \ldots \mathbf{n}_\ell], \mathbf{n}) \xrightarrow{\star} \mathbf{n}_\ell \qquad \mathbf{n+1} \xrightarrow{\star} \mathbf{n} \\ ([\mathbf{n_1} \ldots \mathbf{n}_\ell], \mathbf{n}) \xrightarrow{(} ([\mathbf{n_1} \ldots \mathbf{n}_\ell\,\mathbf{n}], \mathbf{n+1}) \\ ([\mathbf{n_1} \ldots \mathbf{n}_\ell], \mathbf{n}) \xrightarrow{)} ([\mathbf{n_1} \ldots \mathbf{n}_{\ell-1}], \mathbf{n+1}) \end{array}$$

*Let $w = w_0 \ldots w_{|w|-1}$ be a prefix of a well-bracketed word. We have $Z \xRightarrow{w} ([\,\mathbf{n_1} \ldots \mathbf{n}_\ell\,], |\mathbf{w}| + \mathbf{1})$ where $[\,n_1 \ldots n_\ell\,]$ is the sequence (in increasing order) of those indices of unmatched opening brackets in w. In turn, $([\,\mathbf{n_1} \ldots \mathbf{n}_\ell\,], |\mathbf{w}| + \mathbf{1}) \xrightarrow{\star} \mathbf{n}_\ell \xrightarrow{\star^{n_\ell}} \mathbf{0}$. Hence, as expected, the number of $\star$ symbols is equal to 1 if w is well-bracketed (*i.e.* $\ell = 0$), and otherwise it is equal to the index of the last unmatched opening bracket plus one.*

F. Collapsible Pushdown Automata

Fix a finite stack alphabet Γ and a distinguished *bottom-of-stack symbol* $\bot \notin \Gamma$. An order-1 stack is a sequence $\bot, a_1, \ldots, a_\ell \in \bot\Gamma^*$ which is denoted $[\,\bot a_1 \ldots a_\ell\,]_1$. An *order-$k$ stack* (or a *k-stack*), for $k > 1$, is a non-empty sequence $s_1, \ldots, s_\ell$ of order-$(k-1)$ stacks which is written $[\,s_1 \ldots s_\ell\,]_k$. For convenience, we may sometimes see an element $a \in \Gamma$ as an order-0 stack, denoted $[\,a\,]_0$. We denote by Stacks_k the set of all order-k stacks and $\mathrm{Stacks} = \bigcup_{k \geqslant 1} \mathrm{Stacks}_k$ the set of all higher-order stacks. The height of the stack s denoted $|\,s\,|$ is simply the length of the sequence. We denote by $ord(s)$ the order of the stack s.

A *substack* of an order-1 stack $[\,\bot a_1 \ldots a_h\,]_1$ is a stack of the form $[\,\bot a_1 \ldots a_{h'}\,]_1$ for some $0 \leqslant h' \leqslant h$. A *substack* of an order-k stack $[\,s_1 \ldots s_h\,]_k$, for $k > 1$ is either a stack of the form $[\,s_1 \ldots s_{h'}\,]_k$ with $0 < h' \leqslant h$ or a stack of the form $[\,s_1 \ldots s_{h'}\,s'\,]_k$ with $0 \leqslant h' \leqslant h-1$ and s' a substack of $s_{h'+1}$. We denote by $s \sqsubseteq s'$ the fact that s is a substack of s'.

In addition to the operations $push_1^a$ and pop_1 that respectively pushes and pops a symbol in the topmost order-1 stack, one needs extra operations to deal with the higher-order stacks: the pop_k operation removes the topmost order-k stack, while the $push_k$ duplicates it.

For an order-n stack $s = [\,s_1 \ldots s_\ell\,]_n$ and an order-k stack t with $0 \leqslant k < n$, we define $s +\!\!+\, t$ as the order-n stack obtained by pushing t on top of s:

$$s +\!\!+\, t = \begin{cases} [\,s_1 \ldots s_\ell\,t\,]_n & \text{if } k = n-1, \\ [\,s_1 \ldots (s_\ell +\!\!+\, t)\,]_n & \text{otherwise.} \end{cases}$$

We first define the (partial) operations pop_i and top_i with $i \geqslant 1$: $top_i(s)$ returns the top $(i-1)$-stack of s, and $pop_i(s)$ returns s with its top $(i-1)$-stack removed. Formally, for an order-n stack $[\,s_1 \cdots s_{\ell+1}\,]_n$ with $\ell \geqslant 0$

$$
\begin{aligned}
top_i(s) &= \begin{cases} s_{\ell+1} & \text{if } i = n \\ top_i(s_{\ell+1}) & \text{if } i < n \end{cases} \\
pop_i(s) &= \begin{cases} [\,s_1 \cdots s_\ell\,]_n & \text{if } i = n \text{ and } \ell \geqslant 1 \\ [\,s_1 \cdots s_\ell \, pop_i(s_{\ell+1})\,] & \text{if } i < n \end{cases}
\end{aligned}
$$

By abuse of notation, we let $top_{ord(s)+1}(s) = s$. Note that $pop_i(s)$ is defined if and only if the height of $top_{i+1}(s)$ is strictly greater than 1. For example $pop_2([\,[\,\bot\, a\, b\,]_1\,]_2)$ is undefined.

We now introduce the operations $push_i$ with $i \geqslant 2$ that duplicates the top $(i-1)$-stack of a given stack. More precisely, for an order-n stack s and for $2 \leqslant i \leqslant n$, we let $push_i(s) = s +\!\!+\, top_i(s)$.

The last operation, $push_1^a$ pushes the symbol $a \in \Gamma$ on top of the top 1-stack. More precisely, for an order-n stack s and for a symbol $a \in \Gamma$, we let $push_1^a(s) = s +\!\!+ [\,a\,]_0$.

Example 5. *Let s be the order-3 stack of height 2 given by* $s = [\,[\,[\bot baac]_1\,[\bot bb]_1\,[\bot bcc]_1\,[\bot cba]_1\,]_2\,[\,[\bot baa]_1\,[\bot bc]_1\,[\bot bab]_1\,]_2\,]_3$. *Then* $top_3(s)$ *is the* 2*-stack* $[\,[\bot baa]_1\,[\bot bc]_1\,[\bot bab]_1\,]_2$ *and* $pop_3(s)$ *is the stack* $s' = [\,[\,[\bot baac]_1\,[\bot bb]_1\,[\bot bcc]_1\,[\bot cba]_1\,]_2\,]_3$. *Note that* $pop_3(pop_3(s))$ *is undefined. Then* $push_2(s')$ *is the stack* $[\,[\,[\bot baac]_1\,[\bot bb]_1\,[\bot bcc]_1\,[\bot cba]_1\,[\bot cba]_1\,]_2\,]_3$ *and* $push_1^c(s') = [\,[\,[\bot baac]_1\,[\bot bb]_1\,[\bot bcc]_1\,[\bot cbac]_1\,]_2\,]_3$.

We now define a richer structure of higher-order stacks where we allow links. Intuitively, a stack with links is a higher-order stack in which any symbol may have a link that points to an internal stack below it. This link may be used later to collapse part of the stack.

Order-n stacks with links are order-n stacks with a richer stack alphabet. Indeed, each symbol in the stack can be either an element $a \in \Gamma$ (*i.e.* not being the source of a link) or an element $(a, \ell, h) \in \Gamma \times \{2, \cdots, n\} \times \mathbb{N}$ (*i.e.* being the source of an ℓ-link pointing to the h-th $(\ell-1)$-stack inside the topmost ℓ-stack). Formally, order-n stacks with links over alphabet Γ are defined as order-n stacks [1] over alphabet $\Gamma \cup \Gamma \times \{2, \cdots, n\} \times \mathbb{N}$.

Example 6. *The stack s below is an order-3 stack with links* $[\,[\,[\bot baac]_1\,[\bot bb]_1\,[\bot bc(c,2,2)]_1\,]_2\,[\,[\bot baa]_1\,[\bot bc]_1\,[\bot b(a,2,1)(b,3,1)]_1\,]_2\,]_3$.

To improve readability when displaying n-stacks in examples, we shall explicitly draw the links rather than using stacks symbols in $\Gamma \times \{2, \cdots, n\} \times \mathbb{N}$. For instance, we shall rather represent s as follows:

$$[\,[\,[\bot baac]_1\,[\bot bb]_1\,[\bot bcc]_1\,]_2\,[\,[\bot baa]_1\,[\bot bc]_1\,[\bot bab]_1\,]_2\,]_3$$

[1]Note that we therefore slightly generalise our previous definition as we implicitly use an infinite stack alphabet, but this does not introduce any technical change in the definition.

In addition to the previous operations pop_i, $push_i$ and $push_1^a$, we introduce two extra operations: one to create links, and the other to collapse the stack by following a link. Link creation is made when pushing a new stack symbol, and the target of an ℓ-link is always the $(\ell-1)$-stack below the topmost one. Formally, we define $push_1^{a,\ell}(s) = push_1^{(a,\ell,h)}$ where we let $h = |top_\ell(s)| - 1$ and require that $h > 1$.

The collapse operation is defined only when the topmost symbol is the source of an ℓ-link, and results in truncating the topmost ℓ-stack to only keep the component below the target of the link. Formally, if $top_1(s) = (a, \ell, h)$ and $s = s' +\!\!+ [t_1 \cdots t_k]_\ell$ with $k > h$ we let $collapse(s) = s' +\!\!+ [t_1 \cdots t_h]_\ell$.

For any n, we let $\mathrm{Op}_n(\Gamma)$ denote the set of all operations over order-n stacks with links.

Example 7. *Let* $s = [\,[\,[\,\bot\, a\,]_1\,]_2\;[\,[\,\bot\,]_1\,[\,\bot\, a\,]_1\,]_2\,]_3$. *We have*

$$
\begin{aligned}
push_1^{b,2}(s) &= [\,[\,[\,\bot\, a\,]_1\,]_2\;[\,[\,\bot\,]_1\,[\,\bot\, a\, b\,]_1\,]_2\,]_3 \\
collapse\,(push_1^{b,2}(s)) &= [\,[\,[\,\bot\, a\,]_1\,]_2\;[\,[\,\bot\,]_1\,]_2\,]_3 \\
\underbrace{push_1^{c,3}(push_1^{b,2}(s))}_{\theta} &= [\,[\,[\,\bot\, a\,]_1\,]_2\;[\,[\,\bot\,]_1\,[\,\bot\, a\, b\, c\,]_1\,]_2\,]_3.
\end{aligned}
$$

Then $push_2(\theta)$ *and* $push_3(\theta)$ *are respectively*

$$[\,[\,[\,\bot\, a\,]_1\,]_2\;[\,[\,\bot\,]_1\,[\,\bot\, a\, b\, c\,]_1\,[\,\bot\, a\, b\, c\,]_1\,]_2\,]_3 \text{ and}$$

$$[\,[\,[\,\bot\, a\,]_1\,]_2\;[\,[\,\bot\,]_1\,[\,\bot\, a\, b\, c\,]_1\,]_2\;[\,[\,\bot\,]_1\,[\,\bot\, a\, b\, c\,]_1\,]_2\,]_3.$$

We have $collapse\,(push_2(\theta)) = collapse\,(push_3(\theta)) = collapse(\theta) = [\,[\,[\,\bot\, a\,]_1\,]_2\,]_3$.

An *order-n (deterministic) collapsible pushdown automaton* (n-CPDA) is a 5-tuple $\mathcal{A} = \langle\, \Sigma, \Gamma, Q, \delta, q_0 \,\rangle$ where Σ is an input alphabet containing a distinguished symbol denoted ε, Γ is a stack alphabet, Q is a finite set of control states, $q_0 \in Q$ is the initial state, and $\delta : Q \times (\Gamma \cup \{\bot\}) \times \Sigma \to Q \times \mathrm{Op}_n(\Gamma)$ is a (partial) transition function such that, for all $q \in Q$ and $\gamma \in \Gamma$, if $\delta(q, \gamma, \varepsilon)$ is defined then for all $a \neq \varepsilon$, $\delta(q, \gamma, a)$ is undefined, *i.e.* if some ε-transition can be taken, then no other transition is possible. We require δ to respect the convention that $\bot$ cannot be pushed onto or popped from the stack.

Let $\mathcal{A} = \langle\, \Sigma, \Gamma, Q, \delta, q_0 \,\rangle$ be an n-CPDA. A *configuration* of an n-CPDA is a pair of the form (q, s) where $q \in Q$ and s is an n-stack with link over Γ; we call $(q_0, [[\cdots[\bot]_1 \cdots]_{n-1}]_n)$ the *initial configuration*. It is then natural to associate with $\mathcal{A}$ a deterministic LTS denoted $\mathcal{L}_\mathcal{A} = \langle\, D, r, \Sigma, (\xrightarrow{a})_{a \in \Sigma} \,\rangle$ and defined as follows. We let D be the set of all configurations of $\mathcal{A}$ and r be the initial one. Then for all $a \in \Sigma$ and all $(q, s), (q', s') \in D$ we have $(q, s) \xrightarrow{a} (q', s')$ if and only if $\delta(q, top_1(s), a) = (q', op)$ and $s' = op(s)$.

The tree generated by an n-CPDA $\mathcal{A}$, denoted $\mathrm{Tree}^{\perp}(\mathcal{A})$, is simply the tree $\mathrm{Tree}^{\perp}(\mathcal{L}_{\mathcal{A}})$ of its LTS.

III. From Recursion Schemes to Collapsible Pushdown Automata

In this section, we present a translation of schemes into CPDA. This translation generalizes at all orders the order-2 translation of [A4]. The translation from [9] assumes a normal form for the schemes but up to these normalisations, the CPDA obtained is the same as the one in [9]. Our contributions are to work direclty on schemes without normalisation and more importantly to prove the correctness of the translations without using game semantics as an intermediary tool as in [9]. Note that the converse translation from [9] (from CPDA into scheme) does not use game semantics and is therefore not presented here.

We construct, for any labeled recursion scheme $\mathcal{S}$, a collapsible pushdown automaton $\mathcal{A}$ of the same order defining the same tree as $\mathcal{S}$ – *i.e.* $\mathrm{Tree}^{\perp}(\mathcal{S}) = \mathrm{Tree}^{\perp}(\mathcal{A})$. To simplify the presentation, we assume that $\mathcal{S}$ does not contain any silent productions rule (*i.e.* production rule labeled by e). If $\mathcal{S}$ were to contain silent transitions, we would treat the symbol e as any other symbol[2] in Σ. For the rest of this section, we fix a labeled recursion scheme $\langle \Sigma, N, \mathcal{R}, Z, \perp \rangle$ of order $n \geqslant 1$ without silent transitions.

The automaton $\mathcal{A}$ has a distinguished state, denoted $q_\star$, and with the configurations of the form $(q_\star, s)$ we will associate a ground term over N denoted by $[\![s]\!]$. Other configurations correspond to internal steps of the simulation and are only the source of silent transitions. To show that the two LTS define the same trees, we will establish that, for any reachable configuration of the form $(q_\star, s)$ and for any $a \in \Sigma$, the following holds:

- if $(q_\star, s) \xrightarrow[\mathcal{A}]{a e^*} (q_\star, s')$ then $[\![s]\!] \xrightarrow[\mathcal{S}]{a} [\![s']\!]$;
- if $[\![s]\!] \xrightarrow[\mathcal{S}]{a} t$ then $(q_\star, s) \xrightarrow[\mathcal{A}]{a e^*} (q_\star, s')$ and $[\![s']\!] = t$.

Hence, the main ingredient of the construction is the partial mapping $[\![\cdot]\!]$ associating with any order-n stack a ground term over N. The main difficulty is to guarantee that any rewriting rule of $\mathcal{S}$ applicable to the encoded term $[\![s]\!]$ can be simulated by applying a sequence of stack operations to s. In Section III-A, we present the mapping $[\![\cdot]\!]$ together with its basic properties; in Section III-B, we give the definition of $\mathcal{A}$ and prove the desired properties.

To simplify the presentation we assume, without loss of generality, that all productions starting with a non-terminal A have the same left-hand side (*i.e.* they use the same variables in the same order) and that two productions starting with different non-terminals do not share any variables. Hence a variable $x \in V$ appears in a unique left-hand side $A\, x_1 \ldots, x_{\varrho(A)}$ and we denote by $\mathrm{rk}(x)$ the index of x in the sequence $x_1 \cdots x_{\varrho(A)}$ (*i.e.* $x = x_{\mathrm{rk}(x)}$).

[2] Formally, one labels all silent production rules of $\mathcal{S}$ by a fresh symbol e to obtain a labeled scheme $\mathcal{S}'$ without silent transitions. The construction presented in this section produces an automaton $\mathcal{A}'$ such that $\mathrm{Tree}^{\perp}(\mathcal{S}') = \mathrm{Tree}^{\perp}(\mathcal{A}')$. The automaton $\mathcal{A}$ obtained by replacing all e-labeled rules of $\mathcal{A}$ by e is such that $\mathrm{Tree}^{\perp}(\mathcal{S}) = \mathrm{Tree}^{\perp}(\mathcal{A})$.

Throughout the whole section, we will illustrate definitions and constructions using as a running example the order-2 scheme $\mathcal{S}_U$ generating the tree T_U of Example 4.

A. Stacks Representing Terms.

The stack alphabet Γ consists of the initial symbol and of the right-hand sides of the rules in $\mathcal{R}$ and their argument subterms, *i.e.* $\Gamma \stackrel{\text{def}}{=} \{ Z \} \cup \bigcup_{F\, x_1 \cdots x_{\varrho(x)} \xrightarrow{a} e} \{ e \} \cup \mathrm{ASubs}(e)$.

For the scheme $\mathcal{S}_U$, one gets $\Gamma = \{x, y, z, u, \varphi\} \cup \{ Z\,, G\,(H\,X)\,, H\,X\,, X\,, F\,(F\,\varphi\,x)\,y\,(Hy)\,, F\,\varphi\,x\,, Hy\,, F\,G\,z\,(Hz)\,, G\,, Hz\,, \varphi\,(Hy)\}$.

Notation 1. *For $\varphi \in V \cup N$, a φ-stack designates a stack whose top symbol starts with φ. By extension a stack s is said to be an N-stack (resp. a V-stack) if it is a φ-stack for some $\varphi \in N$ (resp. $\varphi \in V$).*

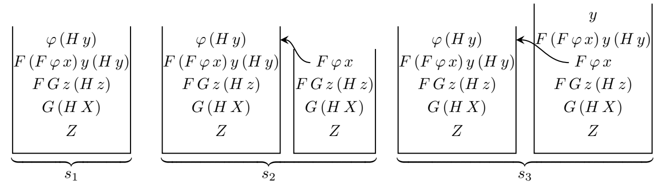
In order to represent a term in $\mathrm{Terms}(N)$, a stack over Γ must be *well-formed*, *i.e.* it must satisfy some syntactic conditions.

Definition 1 (Well-formed stack)**.** *A non-empty stack of order-n over Γ is* well-formed *if every non-empty substack r of s satisfies the following two conditions:*

- *if $top_1(r)$ is not equal to Z nor to $\perp$ then $pop_1(r)$ is an A-stack for some $A \in N$ and $top_1(r)$ belongs to an A-production rule,*
- *if $top_1(r)$ is of type τ of order $k > 0$ then $top_1(r)$ is the source of an $(n-k+1)$-link and $collapse(r)$ is a φ-stack for some variable $\varphi \in V$ of type τ.*

We denote by $\mathrm{WStacks}$ *the set of all well-formed stacks.*

Example 8. *For the scheme $\mathcal{S}_U$, the following order-2 stacks are well-formed.*



Notation 2. *We write $s :: t$ for $s \in \mathrm{WStacks}$ and $t \in \Gamma$ to mean that if t belongs to the r.h.s. of a production starting with $A \in N$ then s is an A-stack. In particular, if $s \in \mathrm{WStacks}$ then $pop_1(s) :: top_1(s)$. We denote by* $\mathrm{CStacks}$ *the set of such $s :: t$, and define the* size *of an element $s :: t$ as the pair $(|s|, |t|)$ where $|s|$ denotes the number of stack symbols in s and $|t|$ the length of the term t. When comparing sizes, we use the standard lexicographic (total) order over $\mathbb{N} \times \mathbb{N}$.*

In Definition 4, we will associate, with any well-formed stack s, a ground term over N that we refer to as the

value of s. To define this value, we first associate, with any element $s :: t$ in CStacks, a value denoted $[\![s :: t]\!]$. This value is a term over N of the same type as t. Intuitively, it is obtained by replacing the variables appearing in the term t by values encoded in the stack s, and one should therefore understand $[\![s :: t]\!]$ as the value of the term t in the context (or environment) of s. See Remark 3 below for natural connections with Krivine machine.

Definition 2. *For all $\varphi \in V \cup N$, all $k \in [1, \varrho(\varphi)]$ and all φ-stack $s \in$ WStacks, we define an element of CStacks, denoted $Arg_k(s)$, representing the k-th argument of the term represented by s. More precisely if the top symbol of s is $\varphi\, t_1 \cdots t_\ell$, we take:*

$$\begin{cases} Arg_k(s) = pop_1(s) :: t_k & \text{if } k \leqslant \ell, \\ Arg_k(s) = Arg_{k-\ell}(collapse(s)) & \text{otherwise.} \end{cases}$$

Definition 3. *For all $s :: t \in$ CStacks, we define the value of t in the context of s:*

$$\begin{cases} [\![s :: t_1 t_2]\!] = [\![s :: t_1]\!][\![s :: t_2]\!] & \text{if } t_1, t_2 \in \Gamma \\ [\![s :: A]\!] = A & \text{if } A \in N \\ [\![s :: x]\!] = [\![Arg_{\mathrm{rk}(x)}(s)]\!] & \text{if } x \in V \end{cases}$$

Let us provide some intuitions regarding the definition of $[\![s :: t]\!]$. Unsurprisingly $[\![s :: t]\!]$ is defined by structural induction on t, and the cases for the application and the non-terminal symbols are straightforward. It remains to consider the case where t is a variable x appearing in $\mathrm{rk}(x)$-th position in the left-hand side $A\, x_1 \cdots x_{\varrho(A)}$. As $s :: t \in$ CStacks, $top_1(s)$ is of the form $At_1 \ldots t_\ell$ for some $\ell \leqslant \varrho(A)$. Note that ℓ is not necessarily equal to $\varrho(A)$ meaning that some arguments of A might be missing. There are now two cases — that correspond to the two cases in the definition of $\mathrm{Arg}_k(s)$ — depending on whether x references to one of the t_i's (*i.e.* $\mathrm{rk}(x) \leqslant \ell$) or one of the missing arguments (*i.e.* $\mathrm{rk}(x) > \ell$):

- If $\mathrm{rk}(x) \leqslant \ell$ then the term associated with x in s is equal to the term associated with $t_{\mathrm{rk}(x)}$ in $pop_1(s)$, *i.e.* $[\![s :: x]\!] = [\![pop_1(s) :: t_{\mathrm{rk}(x)}]\!]$.
- If $\mathrm{rk}(x) > \ell$ then the term $[\![s :: x]\!]$ is obtained by following the link attached to $top_1(s)$. Recall that, as s is a well-formed stack and $top_1(s)$ is not of ground type (as $\ell < \varrho(A)$), there exists a link attached to $top_1(s)$. Moreover, $collapse(s)$, the stack obtained by following the link, has a top-symbol of the form $\varphi\, t'_1 \ldots t'_m$ for some $\varphi \in V$ and $m \geqslant 0$. Intuitively, t'_i corresponds to the $(\ell + i)$-th argument of A. If $\mathrm{rk}(x)$ belongs to $[\ell + 1, \ell + m]$ then the term $[\![s :: x]\!]$ is defined to be the term $[\![pop_1(collapse(s)) :: t'_{\mathrm{rk}(x)-\ell}]\!]$. If $\mathrm{rk}(x)$ is greater than $\ell + m$ then the link attached to the top symbol of $collapse(s)$ is followed and the process is reiterated. As the size of the stack strictly decreases at each step this process terminates.

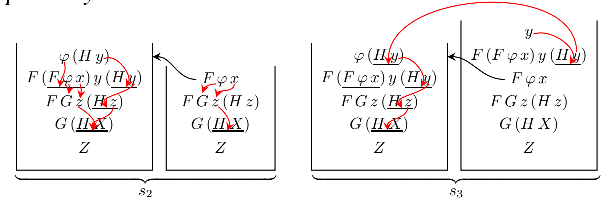
Now, if s is a well-formed φ-stack, its value is obtained by applying the value of φ in the context of $pop_1(s)$ to the value of all its $\varrho(\varphi)$ arguments. This leads to the following formal definition.

Definition 4. *The term associated with a well-formed φ-stack $s \in$ Stacks with $\varphi \in N \cup V$ is*

$$[\![s]\!] \overset{\mathrm{def}}{=} [\![pop_1(s) :: \varphi]\!][\![Arg_1(s)]\!] \cdots [\![Arg_{\varrho(\varphi)}(s)]\!].$$

Equiv., if $top_1(s) : o$ then: $[\![s]\!] = [\![pop_1(s) :: top_1(s)]\!]$. If $top_1(s) : \tau_1 \to \ldots \to \tau_\ell \to o$ then: $[\![s]\!] = [\![pop_1(s) :: top_1(s)]\!] [\![\mathrm{Arg}_1(collapse(s))]\!] \cdots [\![\mathrm{Arg}_\ell(collapse(s))]\!]$.

Example 9. *Let us consider the well-formed stacks s_2 and s_3 presented in Example 8. In the representation below the association between variables and their "values" are made explicit by the red arrows.*



$$[\![s_1]\!] = [\![s_2]\!] = F\, G\, (H\, X)\, (H(H(H(H\, X))))$$
$$[\![s_3]\!] = H(H(H(H(H\, X)))$$

The following lemma states the basic properties of the encoding $[\![\cdot]\!]$ and $\mathrm{Arg}_k(\cdot)$.

Lemma 1. *We have the following properties:*

1) *For all φ-stacks $s \in$ WStacks with $\varphi \in V \cup N$ of type $\tau_1 \to \ldots \to \tau_{\varrho(\varphi)} \to o$ and for all $k \in [1, \varrho(\varphi)]$, $Arg_k(s)$ is equal to some $r :: t \in$ CStacks with t of type τ_k.*
2) *For all $s :: t \in$ CStacks with $t : \tau \in \Gamma$, $[\![s :: t]\!]$ is a term in $\mathrm{Terms}_\tau(N)$.*
3) *For all $s \in$ WStacks, $[\![s]\!]$ belongs to $\mathrm{Terms}_o(N)$.*

We conclude with two fundamental properties of $\mathrm{Arg}_k(\cdot)$ that will allow us to simulate the rewriting of the scheme using stack operations and finite memory.

The first property is that the arguments represented by a well-formed stack are not modified when performing a $push_k$ operation. More precisely, for all φ-stacks $s \in$ WStacks with $\varphi \in N \cup V$, $[\![\mathrm{Arg}_\ell(push_k(s))]\!] = [\![\mathrm{Arg}_\ell(s)]\!]$ for all $\ell \in [1, \varrho(\varphi)]$ and all $k \in [2, m]$. This follows (by letting $r = top_k(s)$) from the following slightly more general result.

Lemma 2. *Let $k \in [2, m]$ and let $s = s' + top_k(s) \in$ WStacks. For all non-empty φ-stacks $r \sqsubseteq top_k(s)$, $[\![Arg_\ell(s' + r)]\!] = [\![Arg_\ell(s + r)]\!]$ for all $\ell \in [1, \varrho(\varphi)]$.*

The next property will later be used to prove that any rewriting step can be simulated by a *finite* number of transitions in the automaton.

Lemma 3. *Let s be a φ-stack in* WStacks *for some* $\varphi : \tau_1 \to \ldots \to \tau_{\varrho(\varphi)} \to \mathrm{o}$ *in* $V \cup N$ *and let* $\ell \in [1, \varrho(\varphi)]$ *with* τ_ℓ *of order* $k > 0$. *If* $Arg_\ell(s)$ *is equal to* $r :: t \in \mathrm{CStacks}$ *with* t *starting with* $\psi \in N \cup V$ *then* $pop_{n-k+1}(s) = pop_{n-k+1}(r)$, $|\, top_{n-k+1}(s)\,| > |\, top_{n-k+1}(r)\,|$.

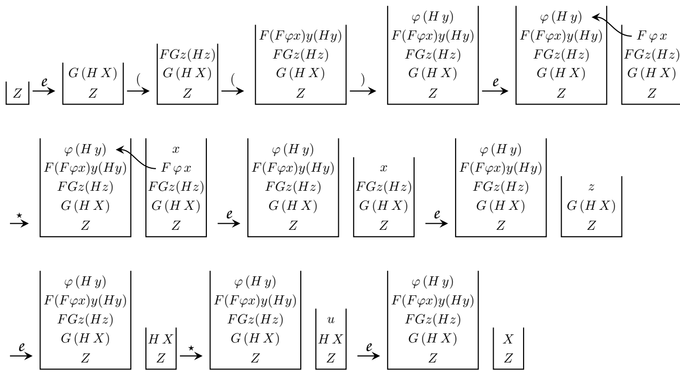
B. Simulating the LTS of $\mathcal{S}$ on Stacks

As an intermediate step, we define an LTS $\mathcal{M}$ over well-formed stacks and we prove that it generates the same tree as $\mathcal{S}$ (*i.e.* $\mathrm{Tree}^\perp(\mathcal{M}) = \mathrm{Tree}^\perp(\mathcal{S})$). From $\mathcal{M}$, a CPDA generating $\mathrm{Tree}^\perp(\mathcal{M})$ is then defined at the end of this section.

We let $\mathcal{M} = \langle\, \mathrm{WStacks}, [\ldots[\,\perp Z\,]\ldots]_n, \Sigma, (\xrightarrow[\mathcal{M}]{a})_{a\in\Sigma}\,\rangle$ and define the transitions as follows

- $s \xrightarrow[\mathcal{M}]{a} push_1^t(s)$ if s is an A-stack with $A \in N$ and $A\, x_1 \cdots x_{\varrho(A)} \xrightarrow{a} t \in \mathcal{R}$,
- $s \xrightarrow[\mathcal{M}]{e} push_1^t(r)$ if s is a φ-stack with $\varphi : \mathrm{o} \in V$ and $\mathrm{Arg}_{\mathrm{rk}(\varphi)}(pop_1(s)) = r :: t$,
- $s \xrightarrow[\mathcal{M}]{e} push_1^{t,n-k+1}(r)$ if s is a φ-stack with $\varphi : \tau \in V$ of order $k > 0$ and $\mathrm{Arg}_{\mathrm{rk}(\varphi)}(pop_1(push_{n-k+1}(s))) = r :: t$.

Example 10. *In the figure below, we illustrate the definition of $\mathcal{M}$ on the scheme $\mathcal{S}_U$.*



The first line of the definition of $\xrightarrow[\mathcal{M}]{}$ corresponds to the case of an N-stack. To simulate the application of a production rule $A\, x_1 \cdots x_n \xrightarrow{a} e$ on the term encoded by an A-stack s, we simply push the right-hand side e of the production on top of s. The correctness of this rule directly follows from the definition of $[\![\cdot]\!]$. Doing so, a term starting with a variable may be pushed on top of the stack, *e.g.* when applying the production rule $F\,\varphi\, x\, y \xrightarrow{)} \varphi\,(H\,y)$. Indeed, we need to retrieve the value of the head variable in order to simulate the next transition of $\mathcal{S}$: the second and third lines of the definition are normalisation rules that aim at replacing the variable at the head of the top of the stack (for instance, in the 5th stack of Example 10 the variable φ) by its definition (hence not changing the value of the associated term). By iterative application, we eventually end up with an N-stack encoding the same term and we can apply again the first rule.

Proposition 1. $\mathrm{Tree}^\perp(\mathcal{S}) = \mathrm{Tree}^\perp(\mathcal{M})$.

Sketch: One easily concludes after establishing the following soundness result about the definition of $\xrightarrow[\mathcal{M}]{}$.

- Let s be an N-stack in WStacks and $a \in \Sigma$.
  For any $t \in \mathrm{Terms}(N)$, if $[\![\, s\,]\!] \xrightarrow{a} t$ then $\exists s' \in \mathrm{WStacks}$, $s \xrightarrow[\mathcal{M}]{a} s'$ and $[\![\, s'\,]\!] = t$.
  If $\exists s' \in \mathrm{WStacks}$, $s \xrightarrow[\mathcal{M}]{a} s'$ then $[\![\, s\,]\!] \xrightarrow{a} [\![\, s'\,]\!]$.
- Let $s \in \mathrm{WStacks}$ be a φ-stack for $\varphi \in V$ and let $s' \in \mathrm{WStacks}$ be a ψ-stack for $\psi \in V \cup N$.
  If $s \xrightarrow[\mathcal{M}]{e} s'$ then $[\![\, s\,]\!] = [\![\, s'\,]\!]$, $ord(\varphi) \leqslant ord(\psi)$ and $|\, top_{n-ord(\varphi)+1}(s)\,| > |\, top_{n-ord(\varphi)+1}(s')\,|$.
- For all $s \in \mathrm{WStacks}$ there exists a unique N-stack $s' \in \mathrm{WStacks}$ such that $s \xrightarrow[\mathcal{M}]{e*} s'$.

■

From $\mathcal{M}$ we now define an n-CPDA $\mathcal{A} = \langle\, \Sigma, \Gamma, Q, \delta, q_0\,\rangle$ generating the same tree as $\mathcal{M}$. The set of states Q is equal to $\{\, q_0, q_1, \ldots, q_{\varrho(\mathcal{S})}, q_*\,\}$ where $\varrho(\mathcal{S})$ denotes the maximal arity appearing in $\mathcal{S}$. Intuitively the initial state q_0 is only used to go from $(q_0, [\ldots[\,\perp\,]_1\ldots]_n)$ to $(q_*, [\ldots[\,\perp Z\,]_1\ldots]_n)$; the state q_* is used to mark N-stacks; for $k \in [1, \varrho(\mathcal{S})]$, the state q_k is used to the compute $\mathrm{Arg}_k(\cdots)$. The transitions are given below.

- $\delta(q_0, \perp, e) = (q_*, push_1^Z)$,
- If t starts with $F \in N$ and $F\, x_1 \cdots x_{\varrho(F)} \xrightarrow{a} e \in \mathcal{R}$:
  - $\delta(q_*, t, a) = (q_*, push_1^e)$ if e starts with a symbol in N,
  - $\delta(q_*, t, a) = (q_{\mathrm{rk}(x)}, id)$ if e is a variable $x : o$ (here id is the identity function),
  - $\delta(q_*, t, a) = (q_{\mathrm{rk}(x)}, push_1^e; push_{n-k+1}; pop_1)$ if e starts with a variable x of order $k > 0$.
- If t is a term of the form $\varphi\, t_1 \cdots t_\ell$ for some $\varphi \in V \cup N$:
  - $\delta(q_k, t, e) = (q_{\mathrm{rk}(t_k)}, pop_1; push_1^{t_k})$ if $k \leqslant \ell$ and $t_k : o$,
  - $\delta(q_k, t, e) = (q_{\mathrm{rk}(t_k)}, pop_1; push_1^{t_k, n-h+1})$ if $k \leqslant \ell$ and t_k has order $h > 0$,
  - $\delta(q_k, t, e) = (q_{k-\ell}, collapse)$ if $k > \ell$.

where, for all $t \in \Gamma$, $q_{\mathrm{rk}(t)}$ designates the state $q_{\mathrm{rk}(x)}$ if t starts with a variable x and q_* otherwise, and $op_1; op_2$ means applying op_1 followed by op_2. An equivalent CPDA using only one operation per transition may be obtained by adding intermediary states.

Theorem 1. *For every labeled recursion scheme $\mathcal{S}$ of order-n, there is an n-CPDA $\mathcal{A}$ that generates the same tree. Moreover, the number of states in $\mathcal{A}$ is linear in the maximal arity appearing in $\mathcal{S}$, and its alphabet is of size linear in*

the one of $\mathcal{S}^3$.

Remark 3. *In [18], the authors use Krivine machines [14] as an abstract model to represent the sequence of rewriting of a scheme[4]. A Krivine machine computes the weak head normal form of a λY-term, using explicit substitutions (called here environments). Environments are functions assigning closures to variables, and closures themselves are pairs consisting of a term and an environment. This mutually recursive definition is schematically represented by the grammar $C := (t, \rho)$ and $\rho := \varnothing \mid \rho[x \to C]$ where t is an term of the λY-calculus with free-variable and $\varnothing$ designates the empty environment. The λY-term t_C represented by a closure $C = (t, \rho)$ is inductively defined as t in which every occurrence of a free variable x is replaced by the term $t_{\rho(x)}$.*

A pair $s :: t$ (cf. Notation 2) can be seen as a closure[5] (t, ρ) where $\rho(x)$ is defined for all variables x occurring in t by $\rho(x) = Arg_{\mathrm{rk}(x)}(s)$. With this view in mind and up to the translation of schemes into equivalent λY-terms, the LTS *$\mathcal{M}$ faithfully simulates the Krivine machine presented in [18]. Note that the correspondence is facilitated by the use of labeled schemes.*

This remark also allows us to inherit the simplifications of [18] for the decidability of CPDA parity games.

IV. Safe Higher-order Recursion Schemes

In this section, we consider a syntactic subfamily of recursion schemes called the *safe recursion schemes*. The *safety* constraint was introduced in [10] but was already implicit in the work of Damm [6] (see also [7, p. 44] for a detailed presentation). This restriction constrains the way variables are used to form argument subterms of the rules' right-hand sides.

Definition 5 ([10])**.** *A recursion scheme is* safe *if no right-hand side contains an argument-subterm of order k containing a variable of order strictly less than k.*

For instance, the scheme in Example 3 is safe. On the other hand, the scheme $\mathcal{S}_U$ of Example 4 is not because the production $F\,\varphi\,x\,y \xrightarrow{(} F(F\varphi x)y(Hy)$ contains in its right-hand side the argument subterm $F\varphi x : \mathrm{o} \to \mathrm{o}$ of order-1 which contains the variable $x : \mathrm{o}$ of order-0. Urzyczyn conjectured that (a slight variation of) the tree T_U generated by $\mathcal{S}_U$, though generated by a order-2 scheme, could not be generated by any *safe* scheme. This conjecture was recently proved by Parys [16].

Remark 4. *In [10], [11], the notion of safety is only defined for* homogeneous *schemes. A type is said to be* homogeneous *if it is either ground or equal to $\tau_1 \to \cdots \to \tau_n \to \mathrm{o}$ where the τ_i's are homogeneous and $\mathrm{ord}(\tau_1) \geqslant \cdots \geqslant \mathrm{ord}(\tau_n)$. By extension, a scheme is homogeneous if all its non-terminal symbols have homogeneous types. For instance $(\mathrm{o} \to \mathrm{o}) \to \mathrm{o} \to \mathrm{o}$ is an homogeneous type whereas $\mathrm{o} \to (\mathrm{o} \to \mathrm{o}) \to \mathrm{o}$ is not. We will see in Proposition 2 that dropping the homogeneity constraint in the definition of safety does not change the family of generated trees.*

A. Safety and the Translation from Schemes to CPDA

In [10], [11], the motivation for considering the safety constraint was that safe schemes can be translated into a subfamily of the collapsible automata, namely higher-order pushdown automata. An order-k pushdown automaton is an order-k CPDA that does not use the collapse operation (hence, links are useless).

Theorem 2 below shows that the translation of recursion schemes into collapsible automata presented in Section III, when applied to a safe scheme, yields an automaton in which links are not really needed. Obviously the automaton performs the collapse operations but whenever it is applied to an order-k link its target is the $(k-1)$-stack below the top $(k-1)$-stack. Hence any collapse operation can safely be replaced by a pop_k operation. In doing so, we re-obtain the translation of safe (homogeneous) schemes into higher-order pushdown automata presented in [11].

Definition 6. *A CPDA is* link-free *if for every configuration (p, s) reachable from the initial configuration and for every transition $\delta(p, top_1(s), a) = (q, collapse)$, we have $collapse(s) = pop_\ell(s)$ where ℓ is the order of the link attached to $top_1(s)$.*

Theorem 2. *The translation of Section III applied to a safe recursion scheme yields a link-free collapsible automaton.*

Sketch: We present the ingredients of the proof only at order-2. For the general case, the ideas are similar but lead to more technicalities.

Let us first introduce some notations. Let $(q, s = [s_1 \dots s_m]_2)$ be a configuration of $\mathcal{A}$ reachable from the initial configuration. For $i \in [1, m]$ and $j \in [1, |s_i|]$, we denote by $r(i,j)$, $t(i,j)$ and $o(i,j)$ respectively the j-th symbol of stack s_i, the target (if defined) in $[1, i-1]$ of its link and the order (if defined) of this link. By definition of $\mathcal{A}$, $t(i,j)$ and $o(i,j)$ are defined iff $r(i,j)$ is a term of order $k > 0$ and in this case $o(i,j)$ is equal to $2 - k + 1$

Moreover for $i \in [2, m]$, we let ℓ_i be the smallest index at which s_{i-1} and s_i have a different symbol (or $|s_i| + 1$ if no such index exists).

The stack s satisfies the following properties:

1) for all $i \in [1, |s_1|]$, $t(1, i)$ is undefined;
2) for all $i \in [2, m]$, $\ell_i \leqslant |s_{i-1}|$ and for all $i \in [2, m-1]$, $\ell_i \leqslant |s_i|$;
3) for all $i \in [2, m]$ and $1 \leqslant j < \ell_i$, $t(i,j) = t(i-1, j)$;

[3] The size of a scheme is defined as the sum of the sizes of the left and right hand sides of the rewriting rules. In particular it is larger than the sum of the sizes of all argument subterms of right hand sides of the rules.

[4] The authors work with the equivalent formalism of the λY-calculus.

[5] to represent applicative terms over N instead of λY-terms.

4) for all $i \in [2,m]$ with $\ell_i \leqslant |s_i|$, $r(i,\ell_i)$ does not contain a variable of order 0 and is an argument subterm of $r(i-1,\ell_i)$ and if $r(i,\ell_i)$ is of order 1 then $t(i,\ell_i) = i-1$;
5) for all $i \in [2,m]$ with $j \in [\ell_i+1, |s_i|]$, $t(i,j)$ is undefined;
6) if $m \geqslant 2$ then $\ell_m = |s_m|+1$ iff $top_1(s) = \varphi t_1 \dots t_h$, $q = q_k$ for some $k \in [1,h]$ such that $ord(t_k) = 1$.

These properties are proved by induction on the length of the shortest path in the LTS from the initial configuration to (q,s) and by inspection of the transitions of $\mathcal{A}$.

Inspecting the transitions of $\mathcal{A}$, a $collapse$ operation can only be performed if $q = q_k$ and $top_1(s) = \varphi\, t_1 \dots t_h$ with $k > h$ and $\varphi : (\tau_1, \dots, \tau_m, o)$. Thanks to Definition 1, $\varphi\, t_1 \dots t_h$ is of order-1. Property 5 implies that ℓ_m is either equal to $|s_m|$ or to $|s_m|+1$. Property 6 implies $\ell_m \neq |s_m|+1$ as otherwise we would have $k \leqslant h$. Thus, we have $\ell_m = |s_m|$ and by Property 4, $collapse(s) = pop_2(s)$. ∎

We get the following corollary extending (by dropping the homogeneity assumption) a previous result from [11].

Corollary 1. *Order-k safe schemes and order-k pushdown automata generate the same trees.*

B. Damm's View of Safety

The safety constraint may seem unnatural and purely *ad-hoc*. Inspired by the constraint of derived types of Damm, we introduce a more natural constraint, *Damm-safety*, which leads the same family of trees [6].

Damm-safety syntactically restricts the use of partial application: in any argument subterm of a right-hand side if one argument of some order-k is provided then all arguments of order-k must also be provided. For instance if $\varphi : (o \to o) \to (o \to o) \to o \to o \to o$, $f : o \to o$ and $c : o$, the terms φ, $\varphi\, f\, f$ and $\varphi\, f\, f\, c\, c$ can appear as argument subterms in a Damm-safe scheme but $\varphi\, f$ and $\varphi\, f\, f\, c$ are forbidden.

Definition 7 ([6]). *A recursion scheme is* Damm-safe *if it is homogeneous and all argument-subterms appearing in a right hand-side are of the form $\varphi\, t_1 \cdots t_k$ with $\varphi : \tau_1 \to \cdots \to \tau_n \to o$ and either $k = 0$, $k = n$ or $ord(\tau_k) > ord(\tau_{k+1})$.*

As in Damm-safe scheme all argument subterms of an argument subterm of order-k appearing in a right-hand side have at least order-k, it is easy to see that Damm-safety implies the safety constraint. However, the safety constraint, even when restricted to homogeneous schemes, is less restrictive than Damm-safety. Consider for instance a variable $x : o$ and non-terminals $G : o \to o \to o$ and $C : o$, then $G\, x$ cannot appear as an argument-subterm in a safe scheme but $G\, C$ can. As $G\, C$ does not satisfy Damm-safety constraint, safety is syntactically more permissive than Damm-safety. However unsurprisingly, any safe scheme can be transformed into an equivalent Damm-safe scheme of the same order. The transformation consists in converting the safe scheme into a higher-order pushdown automaton (Corollary 1) and then converting this automaton back to a scheme using the translation of [11]. In fact, this translation of higher-order pushdown automata into safe schemes produces Damm-safe schemes.

Proposition 2. *Damm-safe schemes are safe and for every safe scheme, there exists a Damm-safe scheme of the same order generating the same tree.*

V. Effective Selection

Let $\varphi(X_1, \cdots, X_\ell)$ be a monadic second order (MSO) formula with ℓ second-order free variables, and let t be a term over a ranked alphabet Σ. The *MSO selection problem* is to decide whether the formula $\exists X_1 \dots \exists X_\ell\ \varphi(X_1, \cdots, X_\ell)$ holds in t, and in this case to give a term t_φ over the ranked alphabet $\Xi = \Sigma \times \{0,1\}^\ell$ (we take $\varrho(a,(b_1,\dots,b_\ell)) = \varrho(a)$) such that the following holds:

1) $t = \pi(t_\varphi)$ where π is the alphabetical morphism from $\overrightarrow{\Xi}$ to $\overrightarrow{\Sigma}$ defined by $\pi((a,\overline{b})) = a$ for $a \in \Sigma$ with $\varrho(a) = 0$ and $\pi((a,\overline{b})_i) = a_i$ for $a \in \Sigma$ with $\varrho(a) > 0$ and $i \in [1, \varrho(a)]$. Intuitively, t_φ is obtained by marking every node in t by a vector of ℓ booleans. Indeed for all non-leaf node u, there exists a unique element $(c,\overline{b}) \in \Xi$ such that for all $x \in \overrightarrow{(c,\overline{b})}$, ux is in t_φ. The tuple $\overline{b} \in \{0,1\}^\ell$ is the label of the node u of t. The label of a non-leaf node u of t is denoted b_u
2) The formula $\varphi(X_1 \leftarrow U_1, \dots, X_\ell \leftarrow U_\ell)$ holds in t where $\forall 1 \leqslant i \leqslant \ell$, $U_i = \{u \in t \mid b_u(i) = 1\}$.

Intuitively, the second point states that this marking exhibits a valuation of the X_i for which φ holds in t. We refer to t_φ as a *selector for φ in t*.

Let $\mathcal{R}$ be a class of generators of terms. We say that $\mathcal{R}$ has the *effective MSO selection property* if there is an algorithm that transforms any pair $(R, \varphi(X_1, \dots, X_\ell))$ with $R \in \mathcal{R}$ into some $R_\varphi \in \mathcal{R}$ (if exists) such that the term generated by R_φ is a selector for φ in the term generated by R.

Theorem 3. *Labeled recursion schemes as well as CPDA have the effective MSO selection property.*

The proof of Theorem 3 is highly non-trivial and requires a precise analysis of winning strategies in parity games played over terms generated by CPDA (the key argument is that winning strategies can be embedded into the CPDA generating the term). We do not believe that a proof of the statement for labeled recursion schemes can be obtained without using an automaton model, and we think that it shows the usefulness of CPDA in the study of logical properties of schemes.

Remark 5. *A similar statement for* safe *schemes can be deduced from [8], [3], [5]. However the machinery for general schemes is much more involved.*

In [2] a much weaker notion, *MSO-reflectivity*, was considered. A class of generators of terms is MSO-reflective if it has the effective MSO selection property for those formula $\varphi(X)$ of the form $\varphi(X) \equiv x \in X \Leftrightarrow \psi(x)$ where $\psi(x)$ is an MSO formula with a single first-order free variable (note that in this case, there is a unique valuation of X that makes $\varphi(X)$ holds). The main result of [2] follows from Theorem 3.

Corollary 2. *Labeled recursion schemes as well as CPDA have the effective MSO-reflectivity property.*

Remark 6. *A variant of selection [17] ask for existence of a formula $\psi(X_1, \dots X_\ell)$ that is a* selector *for $\varphi(X_1, \dots X_\ell)$ in t in the following sense. Either neither of the formulas $\exists X_1 \dots \exists X_\ell \ \varphi(X_1, \cdots, X_\ell)$ and $\exists X_1 \dots \exists X_\ell \, \psi(X_1, \cdots, X_\ell)$ holds in t or ψ defines a unique tuple $(U_1, \cdots, U_\ell)$ and this tuple also satisfies φ. In [4] it is shown that a selector does not always exist in general, and the counter-example is for a tree generated by a (safe) recursion scheme.*

A degenerated version of selection is model-checking. Theorem 1 together with a careful analysis of the complexity of parity games on CPDA lead the same complexity as in [13].

Corollary 3. *The μ-calculus model-checking of trees generated by recursion schemes is* polynomial *under the assumption that the arity of types and the formula are bounded above by a constant.*

Acknowledgements: This work was supported by the following projects: AMIS (ANR 2010 JCJC 0203 01 AMIS) and FREC (ANR 2010 BLAN 0202 02 FREC).

References

[1] C. Broadbent. *On Collapsible Pushdown Automata, their Graphs and the Power of Links*. PhD thesis, University of Oxford, Forthcoming.

[2] C. Broadbent, A. Carayol, C.-H. L. Ong, and O. Serre. Recursion schemes and logical reflexion. In *Proc. of LICS'10*, pages 120–129. IEEE, 2010.

[3] A. Carayol. *Automates infinis, logiques et langages*. PhD thesis, Université de Rennes 1, 2006.

[4] A. Carayol, C. Löding, D. Niwiński, and I. Walukiewicz. Choice functions and well-orderings over the infinite binary tree. *Central European Journal of Mathematics*, 8(4):662–682, 2010.

[5] A. Carayol and M. Slaats. Positional strategies for higher-order pushdown parity games. In *Proc. of MFCS'08*, volume 5162 of *LNCS*, pages 217–228. Springer, 2008.

[6] W. Damm. The IO- and OI-hierarchies. *Theoret. Comput. Sci.*, 20:95–207, 1982.

[7] J. de Miranda. *Structures generated by higher-order grammars and the safety constraint*. PhD thesis, University of Oxford, 2006.

[8] S. Fratani. *Automates à piles de piles ... de piles*. PhD thesis, Université de Bordeaux, 2006.

[9] M. Hague, A. S. Murawski, C.-H. L. Ong, and O. Serre. Collapsible pushdown automata and recursion schemes. In *Proc. of LICS'08*, pages 452–461. IEEE, 2008.

[10] T. Knapik, D. Niwiński, and P. Urzyczyn. Deciding monadic theories of hyperalgebraic trees. In *Proc. of TLCA'01*, volume 2044 of *LNCS*, pages 253–267. Springer, 2001.

[11] T. Knapik, D. Niwiński, and P. Urzyczyn. Higher-Order Pushdown Trees Are Easy. In *Proc. of FoSSaCS'02*, volume 2303 of *LNCS*, pages 205–222. Springer, 2002.

[12] T. Knapik, D. Niwiński, P. Urzyczyn, and I. Walukiewicz. Unsafe grammars and panic automata. In *Proc. of ICALP'05*, volume 3580 of *LNCS*, pages 1450–1461. Springer, 2005.

[13] N. Kobayashi and C.-H. L. Ong. A type system equivalent to the modal mu-calculus model checking of higher-order recursion schemes. In *Proc. of LICS'09*, pages 179–188. IEEE, 2009.

[14] J.-L. Krivine. A call-by-name lambda-calculus machine. *Higher-Order and Symbolic Computation*, 20(3):199–207, 2007.

[15] C.-H. L. Ong. On model-checking trees generated by higher-order recursion schemes. In *Proc. of LICS'06*, pages 81–90. IEEE, 2006.

[16] P. Parys. On the Significance of the Collapse Operation. In *Proc. of LiCS'12*. IEEE, 2012.

[17] A. Rabinovich and A. Shomrat. Selection and uniformization problems in the monadic theory of ordinals: A survey. In *Pillars of Computer Science*, volume 4800 of *LNCS*, pages 571–588. Springer, 2008.

[18] S. Salvati and I. Walukiewicz. Krivine machines and higher-order schemes. In *Proc. of ICALP'11*, volume 6756 of *LNCS*, pages 162–173. Springer, 2011.