Higher-order recursion schemes and their automata models

Arnaud Carayol[1] *and Olivier Serre*[2]

[1]LIGM
Université Paris-Est and CNRS
5, boulevard Descartes, Champs-sur-Marne
F-77454 Marne-la-Vallée Cedex 2
email: Arnaud.Carayol@univ-mlv.fr

[2]IRIF
Université Paris Diderot-Paris 7 and CNRS
Case 7014
F-75205 Paris Cedex 13
email: Olivier.Serre@cnrs.fr

2010 Mathematics Subject Classification: 68Q45 or what is appropriate

Key words: Finite automata or what is appropriate

Contents

1 Introduction 440
2 Preliminaries 445
2.1 Trees and terms . . . 445
2.2 Labeled transition systems . . . 446
2.3 Higher-order recursion schemes . . . 447
2.3.1 Simply typed terms . . . 447
2.3.2 Recursion schemes . . . 448
2.3.3 Rewriting system associated with a recursion scheme . . . 449
2.3.4 Value tree of a recursion scheme . . . 450
2.3.5 Labeled recursion schemes . . . 451
2.3.6 Examples of trees defined by labeled recursion schemes . . . 453
2.4 Higher-order pushdown automata . . . 456
2.4.1 Higher-order stack and their operations . . . 456
2.4.2 Stacks with links and their operations . . . 457
2.4.3 Higher-order pushdown automata and collapsible automata . . . 458

3 From CPDA to recursion schemes 459
3.1 Term representation of stacks and configurations 459
3.2 The labeled recursion scheme associated with $\mathcal{A}$ 462
3.3 Correctness of the representation 462
4 From recursion schemes to collapsible pushdown automata 465
4.1 Stacks representing terms. 466
4.2 Simulating the LTS of $\mathcal{S}$ on stacks 471
5 Safe higher-order recursion schemes 475
5.1 Safety and the Translation from Schemes to CPDA 476
5.2 Damm's view of safety 477
References 477

1 Introduction

The main goal of this chapter is to give a self-contained presentation of the equivalence between two models: *higher-order recursion schemes* and *collapsible pushdown automata*. Roughly speaking, a recursion scheme is a finite typed term rewriting system and a natural view of recursion schemes is to be considered generators for (possibly infinite) trees. Collapsible pushdown automata (CPDA) are an extension of deterministic (higher-order) pushdown automata and they naturally induce labeled transition systems (LTS). An LTS is merely a set of relations labeled by a finite alphabet, together with a distinguished element called the root. Hence unfolding an LTS and contracting silent transitions define an infinite tree. Applying this construction to CPDA defines a family of trees that exactly coincides with the family of trees defined by higher-order recursion schemes. This introduction tries to provide the necessary background and motivation for these objects.

Recursive Applicative Program Schemes

Historically, recursion schemes go back to Nivat's *recursive applicative program schemes* [47] that correspond to order-1 recursion schemes in our sense (also see related work by Garland and Luckham on so-called *monadic recursion schemes* [30]). We refer the reader to [24] that, among others things, contains a very detailed and rich history of the topic. For Nivat, a recursive applicative program scheme is a finite system of equations, each of the form $F_i(x_1, \ldots, x_n) = p_i$, where the x_j are order-0 variables and p_i is some order-0 term over the nonterminals (the F_i's), terminals, and the variables $x_1, \ldots, x_k$. In Nivat's work, a *program* is a pair: a program scheme together with an *interpretation* over some domain. An interpretation gives any terminal a function (of the correct rank) over the domain. Taking the least fixed point of the rewriting rules of a program scheme gives a (possibly infinite) term over the terminal alphabet (known as the value of the program in the *free/Hebrand interpretation*); applying the interpretation to this infinite term gives the *value* of the program. Hence, the program scheme gives the uninterpreted syntax tree of some functional program that is then fully specified owing to the interpretation.

Nivat also defined a notion of equivalence for program schemes: two schemes are

equivalent if and only if they compute the same function under *every* interpretation. Later, Courcelle and Nivat [19] showed that two schemes are equivalent if and only if they generate the same infinite term tree. This latter result clearly underscores the importance of studying the trees generated by a scheme. Following the work by Courcelle [16, 17], the equivalence problem for schemes is known to be interreducible to the problem of decidability of language equivalence between deterministic pushdown automata (DPDA). Research on the equivalence for program schemes was halted until Sénizergues [58, 59] established the decidability of DPDA equivalence, which therefore also solved the scheme equivalence problem. Sénizegues' proof was later simplified and improved by Stirling [62, 60]. For more insight about this topic, we refer the reader to [61].

Extension of Schemes to Higher Orders

A recursive function is said to be of higher-order if it takes arguments that are themselves functions. In Nivat's program scheme, both the nonterminals and the variables have order-0. Therefore, they cannot be used to model higher-order recursive programs. In the late 1970s, there was a substantial effort in extending program schemes in order to capture higher-order recursion [35, 20, 21, 27, 28]. Note that evaluation, i.e., computing the value of a scheme in some interpretation, has been a very active topic, in particular because different evaluation policies, e.g., *call by name* (OI) or *call by value* (IO), lead to different semantics [27, 28, 22]. In a very influential paper [22], Damm introduced *order-n λ-schemes* and extended the previously mentioned result of Courcelle and Nivat. Damm's schemes mostly coincide with the *safe* fragment of recursion schemes as we define them later in this chapter. Note that at that time there was no known model of automata equi-expressive with Damm's scheme; in particular, there was no known reduction of the equivalence problem for schemes to a language equivalence problem for (some model of) automata.

Later, Damm and Goerdt [22, 23] considered the word languages generated by level-n λ-schemes and they showed that they coincide with a hierarchy previously defined by Maslov [44, 45]. To define his hierarchy Maslov introduced *higher-order pushdown automata* (higher-order PDA). He also gave an equivalent definition of the hierarchy in terms of *higher-order indexed grammars*. In particular, Maslov's hierarchy offers an attractive classification of the semi-decidable languages: orders 0, 1 and 2 are, respectively, the regular, context-free, and indexed languages, though little is known about languages at higher orders (see [34] for recent results on this topic). Later, Engelfriet [25, 26] considered the characterisation of complexity classes by higher-order pushdown automata. In particular, he showed that alternating pushdown automata characterise deterministic iterated exponential time complexity classes.

Higher-Order Recursion Schemes as Generators of Infinite Structures

Since the late 1990s there has been a strong interest in infinite structures admitting finite descriptions (either internal, algebraic, logical or transformational), mainly motivated by

applications to program verification. See [5] for an overview about this topic. The central question is model-checking: given some presentation of a structure and some formula, decide whether the formula holds. Of course, here decidability is a trade-off between the richness of the structure and the expressivity of the logic.

Of special interest are tree-like structures. Higher-order PDA as a generating device for (possibly infinite) labelled ranked trees were first studied by Knapik, Niwiński and Urzyczyn [37]. As in the case of word languages, an infinite hierarchy of trees is defined according to the order of the generating PDA; lower orders of the hierarchy are well-known classes of trees: orders 0, 1 and 2 are respectively the regular [53], algebraic [18], and hyperalgebraic trees [36]. Knapik *et al.* considered another method of generating such trees, namely by higher-order (deterministic) recursion schemes that satisfy the constraint of *safety*. A major result in their work is the equi-expressivity of both methods as tree generators. In particular, it implies that the equivalence problem for higher-order *safe* recursion schemes is interreducible to the problem of decidability of language equivalence between deterministic higher-order PDA.

An alternative approach was developed by Caucal, who introduced [15] two infinite hierarchies, one made of infinite trees and the other made of infinite graphs, defined by means of two simple transformations: unfolding, which goes from graphs to trees, and inverse rational mapping (or MSO-interpretation [14]), which goes from trees to graphs. He showed that the tree hierarchy coincides with the trees generated by safe schemes.

However the fundamental question open since the early 1980s of finding a class of automata that characterises the expressivity of higher-order recursion schemes was left open. Indeed, the results of Damm and Goerdt, as well as those of Knapik *et al.* may only be viewed as attempts to answer the question, as they have both had to impose the same syntactic constraints on recursion schemes, called of *derived types* and *safety*, respectively, in order to establish their results.

A partial answer was later obtained by Knapik, Niwiński, Urzyczyn, and Walukiewicz, who proved that order-2 homogeneously-typed (but not necessarily safe) recursion schemes are equi-expressive with a variant class of order-2 pushdown automata called *panic automata* [38].

Finally, Hague, Murawski, Ong, and Serre gave a complete answer to the question in [32]. They introduced a new kind of higher-order pushdown automata, which generalizes *pushdown automata with links* [2], or equivalently panic automata, to all finite orders, called *collapsible pushdown automata* (CPDA), in which every symbol in the stack has a link to a (necessarily lower-ordered) stack situated somewhere below it. A major result of their paper is that for every $n \geqslant 0$, order-n recursion schemes and order-n CPDA are equi-expressive as generators of trees.

Decidability of Monadic Second Order Logic

This quest for finding an alternative description of those trees generated by recursion schemes took place in parallel with the study of the decidability of the model-checking problem for monadic second-order logic (MSO) and modal μ-calculus (see [63, 3, 31, 29] for background about these logics and connections with finite automata and games). Decidability of the MSO theories of trees generated by safe schemes was established by

Knapik, Niwiński and Urzyczyn [37] and then Caucal [15] proved a stronger decidability result that holds on graphs as well. The decidability for order-2 unsafe schemes follows from [38] and was obtained thanks to the equi-expressivity with panic automata. This result was independently obtained in [2] with similar techniques.

In 2006, Ong showed the decidability of MSO for arbitrary recursion schemes [48], and established that this problem is n-EXPTIME complete. This result was obtained using tools from innocent game semantics (in the sense of Hyland and Ong [33]) and does not rely on an equivalent automata model for generating trees.

Thanks to their equi-expressivity result, Hague *et al.* provided an alternative proof of the MSO decidability for schemes. Indeed, thanks to the equi-expressivity between schemes and CPDA together with the well-known connections between MSO model-checking (for trees) and parity games, the model-checking problem for schemes is interreducible to the problem of deciding the winner in a two-player perfect information turn-based parity game played over the LTS (i.e., transition graph) associated with a CPDA. They extended the techniques and results of Walukiewicz (for pushdown games) [64], Cachat (for higher-order pushdown) [11] (also see [12] for a more precise study on higher-order pushdown games) and the one from Knapik *et al.* [38]. These techniques were later extended by Broadbent, Carayol, Ong, and Serre to establish stronger results on schemes — in particular closure under MSO marking [8] — and later by Carayol and Serre to prove that recursion schemes enjoy the effective MSO selection property [13].

Some years later, following initial ideas by Aehlig [1], Kobayashi [40], and Kobayashi-Ong [43] gave another proof of the decidability of MSO. The proof consists of showing that one can associate, with any scheme and formula, a typing system (based on intersection types) such that the scheme is typable in this system if and only if the formula holds. Typability is then reduced to solving a parity game.

Using the λY-calculus and Krivine Machines, Salvati and Walukiewicz proposed an alternative approach for the decidability of MSO, as well as, a new proof for the equivalence between schemes and CPDA [55, 56]. In particular, the translation from schemes to CPDA is very similar to the one that we present in this chapter and was independently obtained by the authors in [13].

Recently, Parys established decidability of weak-MSO logic extended by the unbounding quantifier (WMSO+U), for schemes [52].

Verification of Higher-Order Programs

Functional languages such as Haskell, OCaML and Scala strongly encourage the use of higher-order functions. This represents a challenge for software verification, which usually does not model recursion accurately, or models only first-order calls (e.g., SLAM [4] and Moped [57]). However higher-order recursion schemes offer a way of abstracting functional programs in a manner that precisely models higher-order control-flow, and because of the μ-calculus/MSO decidability results for them, it opened a very active line of research toward the verification of higher-order programs.

Even reachability properties (subsumed by the μ-calculus) are very useful in practice: indeed, as a simple example, the safety of incomplete pattern matching clauses could be checked by asking whether the program can reach a state where a pattern match failure

occurs. More complex reachability properties can be expressed using a finite automaton and could, for example, specify that the program respects a certain discipline when accessing a particular resource (see [42] for a detailed overview of the field). Despite even reachability being $(n-1)$-EXPTIME complete, recent research has revealed that useful properties of HORS can be checked in practice.

Kobayashi's TRecS [39] tool, which checks properties expressible by a deterministic trivial Büchi automaton (all states accepting), was the first to demonstrate model-checking of schemes was possible in practice. It works by determining whether a HORS is typable in an intersection type system characterising the property to be checked [42]. In a bid to improve scalability, a number of other algorithms have subsequently been designed and implemented, such as Kobayashi's GTRecS(2) [41] and Neatherway, Ramsay, and Ong's TravMC [46] tools, all based on intersection type inference.

Another approach, providing a fresh set of tools that contrast with previous intersection type techniques, was developed by Broadbent, Carayol, Hague and Serre, relying on an automata-theoretic perspective [9]. Their idea is to start from a recursion scheme and to translate it to an equivalent CPDA, and then perform the verification on the latter. In order to avoid state explosion, they used saturation methods (that were well known to work successfully for pushdown systems [57]) together with an initial forward analysis. This lead to the C-SHORe tool, which is the first model-checking tool for the (direct) analysis of collapsible pushdown systems.

Since C-SHORe was released, two new tools were developed. Broadbent and Kobayashi introduced HorSat (later subsumed by HorSat2), which is an application of the saturation technique and initial forward analysis directly to intersection type analysis of recursion schemes [10]. Secondly, Ramsay, Neatherway and Ong introduced Preface [54], using a type-based abstraction-refinement algorithm that attempts to simultaneously prove and disprove the property of interest. Both HorSat2 and Preface perform significantly better than previous tools.

Structure of this Chapter

Higher-order recursion schemes are a very rich domain and we had to make some choices for both the presentation and the content of this chapter. We decided to devote a large part to the equi-expressivity result between recursion schemes and collapsible pushdown automata. Indeed, it was a longstanding open question in the field; it allowed providing an automata-based proof of the decidability of MSO for recursion schemes; and it gives a tool to who wants to tackle the equivalence problem for recursion schemes (which is interreducible to language equivalence for deterministic CPDA). The presentation of the proof we give is novel and can be thought as a simplification of the original proof in [32]. First, it introduces an alternative definition of schemes called *labeled recursion schemes* by means of labeled transition systems. In these labeled transition systems, the domain is composed of the ground terms built using the non-terminal of the scheme; the relations come from the rewriting rules of the schemes and are labeled by terminals. Second, it presents a transformation from a recursion scheme to a CPDA, which only uses basic automata techniques, and does not appeal to objects from game semantics such as *traversals*. Nevertheless, it is important to stress that, even if concepts like traversals are

no longer present in our proof, the key ideas come from [32] and the CPDA one derives from a scheme is the same as the one defined in [32].

The article is organised as follows. Section 2 introduces the main concepts — schemes and CPDA — together with examples. Then in Section 3 we give a transformation from CPDA to schemes and in Section 4 we provide the converse transformation. Finally, Section 5 is devoted to the notion of safety.

2 Preliminaries

2.1 Trees and terms

Let A be a finite alphabet. We let A^* denote the set of finite words over A, and we refer to a subset of A^* as a language over A. A tree t with directions in A (or simply a *tree* if A is clear from the context) is a non-empty prefix-closed subset of A^*. Elements of t are called *nodes* and ε is called the *root* of t. For a node $u \in t$, the subtree of t rooted at u, denoted t_u, is the tree $\{v \in A^* \mid u \cdot v \in t\}$. We let $\mathrm{Trees}^\infty(A)$ denote the set of trees with directions in A.

A *ranked alphabet* A is an alphabet together with an arity function, $\varrho : A \to \mathbb{N}$. The terms built over a ranked alphabet A are those trees with directions

$$\overrightarrow{A} \stackrel{\text{def}}{=} \bigcup_{f \in A} \overrightarrow{f} \text{ where } \overrightarrow{f} = \{f_1, \ldots, f_{\varrho(f)}\} \text{ if } \varrho(f) > 0 \text{ and } \overrightarrow{f} = \{f\} \text{ if } \varrho(f) = 0.$$

For a tree $t \in \mathrm{Trees}^\infty(\overrightarrow{A})$ 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 if and only if u ends with some $f \in A$ (hence $\varrho(f) = 0$) and if A_u is non-empty, then it is equal to some $\overrightarrow{f}$ for some $f \in A$. We let $\mathrm{Terms}(A)$ denote the set of terms over A.

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

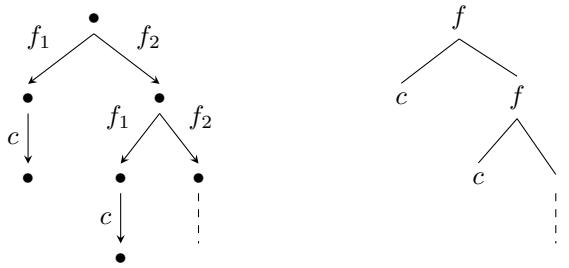


Figure 1. Two representations of the infinite term $f_2^*\{f_1 c, f_1, \varepsilon\} = f(c, f(c, f(\cdots)))$ over the ranked alphabet $\{f, c\}$, assuming that $\varrho(f) = 2$ and $\varrho(c) = 0$.

2.2 Labeled transition systems

A rooted labeled transition system is an edge-labeled directed graph with a distinguished vertex, called the root. Formally, a *rooted labeled transition system* $\mathcal{L}$ (LTS for short) 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*, 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 pair $(s,t) \in D^2$ 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. 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 $w \in L$.

When considering LTS associated with computational models, it is usual to allow silent (or internal) transitions. The symbol for silent transitions is usually ε but here, to avoid confusion with the empty word, we will use λ instead. Following [60, p. 31], we forbid a vertex to be the source of both a silent transition and a non-silent transition. Formally, an LTS *with silent transitions* is an LTS $\langle D, r, \Sigma, (\xrightarrow{a})_{a\in\Sigma} \rangle$ whose set of labels contains a distinguished symbol, denoted $\lambda \in \Sigma$ and such that for all $s \in D$, if s is the source of a λ-transition, then s is not the source of any a-transition with $a \neq \lambda$. We let Σ_λ denote the set $\Sigma \setminus \{\lambda\}$ of non-silent transition labels. For all words $w = a_1 \cdots a_n \in \Sigma_\lambda^*$, we let $\stackrel{w}{\Longrightarrow}$ denote the relation $\xrightarrow{L_w}$, where $L_w \stackrel{\text{def}}{=} \lambda^* a_1 \lambda^* \cdots \lambda^* a_n \lambda^*$ is the set of words over Σ obtained by inserting arbitrarily many occurrences of λ in w.

An LTS (with silent transitions) 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 2.1. From now on, we always assume that the LTS we consider are deterministic.

We associate a tree with every LTS with silent transitions $\mathcal{L}$, denoted $\mathrm{Tree}(\mathcal{L})$, with directions in Σ_λ, reflecting the possible behaviours of $\mathcal{L}$ starting from the root. For this we let $\mathrm{Tree}(\mathcal{L}) \stackrel{\text{def}}{=} \{w \in \Sigma_\lambda^* \mid \exists s \in D,\ r \stackrel{w}{\Longrightarrow} s\}$. 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 λ-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 $\bot$-child for some fresh symbol $\bot$. This tree, denoted $\mathrm{Tree}^\bot(\mathcal{L})$, is defined by letting

$$\mathrm{Tree}^\bot(\mathcal{L}) \stackrel{\text{def}}{=} \mathrm{Tree}(\mathcal{L}) \cup \{w\bot \in \Sigma_\lambda^*\bot \mid \forall n \geqslant 0, r \stackrel{w\lambda^n}{\Longrightarrow} s_n \text{ for some } s_n\}.$$

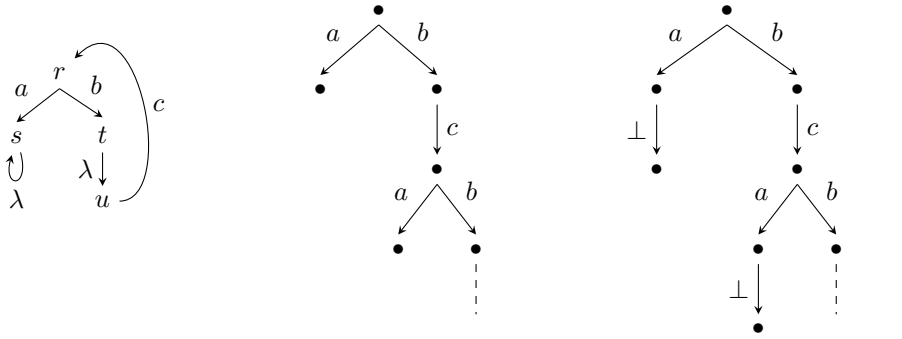


Figure 2. An LTS $\mathcal{L}$ with silent transitions of root r (on the left), the tree $\mathrm{Tree}(\mathcal{L})$ (in the center) and the tree $\mathrm{Tree}^{\perp}(\mathcal{L})$ (on the right).

2.3 Higher-order recursion schemes

Recursion schemes are grammars for simply typed terms, and they are often used to generate a possibly infinite term. Hence before introducing recursion schemes, we start with some necessary definitions about simply typed terms.

Also note that recursion schemes are not traditionally associated with an LTS. Hence we start with the standard definition of recursion schemes as generators for infinite terms, and then we provide an alternative definition based on LTS.

2.3.1 Simply typed 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 adopt 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).

Intuitively, the base type o corresponds to base elements (such as `int` in ML). An arrow type $\tau_1 \to \tau_2$ corresponds to a function taking an argument of type τ_1 and returning an element of type τ_2. Even if there are no specific types for functions taking more than one argument, those functions are represented in their curried form. Indeed, a function taking two arguments of type o and returning a value of type o, in its curried form, has the type $o \to o \to o = o \to (o \to o)$; intuitively, the function only takes its first argument and returns a function expecting the second argument and returning the desired result.

The *order* measures the nesting of a type. Formally one defines $\mathrm{ord}(o) = 0$ and $\mathrm{ord}(\tau_1 \to \tau_2) = \max(\mathrm{ord}(\tau_1) + 1, \mathrm{ord}(\tau_2))$. Alternatively for a type $\tau = (\tau_1, \ldots, \tau_n, o)$ of arity $n > 0$, the order of τ is the maximum of the orders of the arguments plus one, i.e., $\mathrm{ord}(\tau) = 1 + \max\{\mathrm{ord}(\tau_i) \mid 1 \leqslant i \leqslant n\}$.

Example 2.1. The type $o \to (o \to (o \to o))$ has order 1 while the type $((o \to o) \to o) \to o$ has order 3.

Let X be a set of typed symbols. For every symbol $f \in X$, and every type τ, we write

$f : \tau$ to mean that f has type τ. The set of *applicative terms*[1] *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 t to s, 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, and thus we write $t_1 t_2 t_3$ instead of $(t_1 t_2) t_3$.

Example 2.2. Assuming that f and g are two *function* symbols of respective types $(\mathrm{o} \to \mathrm{o}) \to \mathrm{o} \to \mathrm{o}$ and $\mathrm{o} \to \mathrm{o}$ and c is a *constant* symbol of type o, we have

$$g\,c : \mathrm{o}, \qquad f\,g : \mathrm{o} \to \mathrm{o}, \qquad f\,g\,c = (f\,g)\,c : \mathrm{o}, \qquad f\,(f\,g)\,c : \mathrm{o}.$$

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 2.2 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 $|\mathrm{ASubs}(t)| < |t|$.

Fact 1. Any applicative term t over X can be uniquely written as $F\,t_1 \cdots t_n$ where F is a symbol in X of arity $\varrho(F) \geqslant n$ and t_i are applicative terms for all $i \in [1, n]$. Moreover if F has type $(\tau_1, \ldots, \tau_{\varrho(F)}, 0) \in X$, then for all $i \in [1, n]$, t_i has type τ_i and $t : (\tau_{n+1}, \ldots, \tau_{\varrho(F)}, 0)$.

Remark 2.2. In the following, we will simply write "term" instead of "applicative term" and let $\mathrm{Terms}(X)$ denote the set of applicative terms of ground type over X. It should be clear from the context if we are referring to applicative terms over a typed alphabet or terms over a ranked alphabet. Of course, a ranked alphabet A can be seen as a typed alphabet by assigning the type $\underbrace{o \to \cdots \to o}_{\varrho(f)} \to o$ to every symbol f of A. 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).

2.3.2 Recursion schemes 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 $x, y, \varphi, \psi, \chi, \xi, \ldots$ to range over variables.

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

- A is a ranked alphabet of *terminals* and $\bot$ is a distinguished terminal symbol of arity 0 (and hence of ground type) that does not appear in any production rule,
- N is a finite set of typed *non-terminals*; we use upper-case letters $F, G, H, \ldots$ to range over non-terminals,

[1] which should not be confused with terms over a ranked alphabet (cf. Remark 2.2).

- $Z \in N$ is a distinguished *initial symbol* of type o which does not appear in any right-hand side of a production rule,
- $\mathcal{R}$ is a finite set of *production rules*, one for each non-terminal $F : (\tau_1, \cdots, \tau_n, o)$, of the form

$$F\, x_1 \cdots x_n \;\to\; e$$

where the x_i are distinct variables with $x_i : \tau_i$ for $i \in [1, n]$ and e is a ground term in $\mathrm{Terms}((A \setminus \{\bot\}) \cup (N \setminus \{Z\}) \cup \{\, x_1, \ldots, x_n \,\})$. Note that the expressions on both sides of the arrow are terms of *ground* type.

The *order* of a recursion scheme is defined to be the highest order of (the types of) its non-terminals.

2.3.3 Rewriting system associated with a recursion scheme A recursion scheme $\mathcal{S}$ induces a rewriting relation, denoted $\to_{\mathcal{S}}$, over $\mathrm{Terms}(A \cup N)$. Informally, $\to_{\mathcal{S}}$ replaces any ground subterm $F\, t_1 \cdots t_{\varrho(F)}$ starting with a non-terminal F by the right-hand side of the production rule $F\, x_1 \cdots x_n \;\to\; e$ in which the occurrences of the "formal parameter" x_i are replaced by the actual parameter t_i for $i \in [1, \varrho(F)]$.

The term $M[t/x]$ obtained by replacing a variable $x : \tau$ by a term $t : \tau$ over $A \cup N$ in a term M over $A \cup N \cup V$ is defined[2] by induction on M by taking

$$\begin{array}{lcll} (t_1\, t_2)[t/x] & = & t_1[t/x]\, t_2[t/x], & \\ \varphi[t/x] & = & \varphi & \text{for } \varphi \in A \cup N \cup V \text{ if } \varphi \neq x, \\ x[t/x] & = & t. & \end{array}$$

The rewriting system $\to_{\mathcal{S}}$ is defined by induction using the following rules:

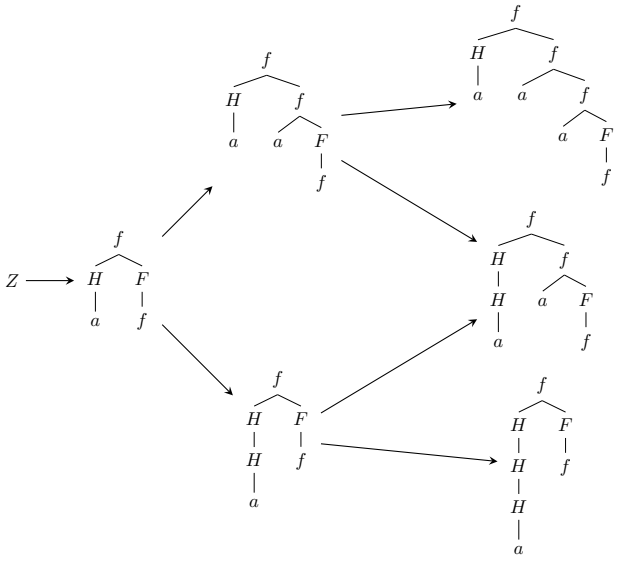
- (*Substitution*) $F t_1 \cdots t_n \to_{\mathcal{S}} e[t_1/x_1, \ldots, t_n/x_n]$ where $F x_1 \cdots x_n \to e$ is a production rule of $\mathcal{S}$.
- (*Context*) If $t \to_{\mathcal{S}} t'$ then $(st) \to_{\mathcal{S}} (st')$ and $(ts) \to_{\mathcal{S}} (t's)$.

Example 2.3. Consider $\mathcal{S}$, the order-2 recursion scheme with the set of non-terminals $\{Z : o,\ H : (o, o),\ F : ((o, o, o), o)\}$, variables $\{z : o,\ \varphi : (o, o, o)\}$, terminals $A = \{f, a\}$ of arity 2 and 0 respectively, and the following rewrite rules:

$$\begin{array}{rcl} Z & \to & f\,(H\, a)\,(F\, f) \\ H\, z & \to & H\,(H\, z) \\ F\, \varphi & \to & \varphi\, a\,(F\, \varphi) \end{array}$$

The figure below depicts the first rewriting steps of $\to_S$, starting from the initial symbol Z.

[2] Note that t does not contain any variables and hence we do not need to worry about capture of variables.



As illustrated above, the relation $\rightarrow_{\mathcal{S}}$ is confluent, i.e., for all ground terms t,t_1 and t_2, if $t \rightarrow_{\mathcal{S}}^* t_1$ and $t \rightarrow_{\mathcal{S}}^* t_2$ (here $\rightarrow_{\mathcal{S}}^*$ denotes the transitive closure of $\rightarrow_{\mathcal{S}}$), then there exists t' such that $t_1 \rightarrow_{\mathcal{S}}^* t'$ and $t_2 \rightarrow_{\mathcal{S}}^* t'$. The proof of this statement is similar to proof of the confluence of the lambda-calculus [6].

2.3.4 Value tree of a recursion scheme Informally the *value tree* of (or the tree *generated* by) a recursion scheme $\mathcal{S}$, denoted $[\![\mathcal{S}]\!]$, is a (possibly infinite) term, *constructed from the terminals in* A, that is obtained as the "limit" of the set of all terms that can obtained by iterative rewriting from the initial symbol Z.

The terminal symbol $\bot : o$ is used to formally restrict terms over $A \cup N$ to their terminal symbols. We define a map $(\cdot)^\bot : \mathrm{Terms}(A \cup N) \longrightarrow \mathrm{Terms}(A)$ that takes an applicative term and replaces each non-terminal, together with its arguments, by $\bot : o$. We define $(\cdot)^\bot$ inductively as follows, where a ranges over A-symbols, and F over non-terminals in N:

$$\begin{aligned} a^\bot &= a, \\ F^\bot &= \bot, \\ (st)^\bot &= \begin{cases} \bot & \text{if } s^\bot = \bot, \\ (s^\bot t^\bot) & \text{otherwise.} \end{cases} \end{aligned}$$

Clearly if $t \in \mathrm{Terms}(A \cup N)$ is of ground type then $t^\bot \in \mathrm{Terms}(A)$ is of ground type as well.

Terms built over A can be partially ordered by the approximation ordering $\preccurlyeq$ defined for all terms t and t' over A by $t \preccurlyeq t'$ if $t \cap (\overrightarrow{A} \setminus \{\bot\})^* \subseteq t'$. In other terms, t' is obtained from t by substituting some occurrences of $\bot$ by arbitrary terms over A.

The set of terms over A together with $\preccurlyeq$ form a *directed complete partial order*, mean-

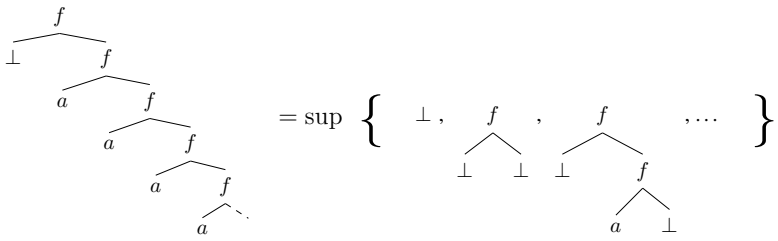
ing that any directed[3] subset D of $\mathrm{Terms}(A)$ admits a *supremum*, denoted $\sup D$.

Clearly if $s \to_{\mathcal{S}} t$ then $s^{\perp} \preccurlyeq t^{\perp}$. The confluence of the relation $\to_{\mathcal{S}}$ implies that the set $\{\, t^{\perp} \mid Z \to_{\mathcal{S}}^{*} t \,\}$ is directed. Hence the *value tree* of (or the tree *generated* by) $\mathcal{S}$ can be defined as its supremum,

$$[\![\, \mathcal{S} \,]\!] \;=\; \sup\{\, t^{\perp} \mid Z \to_{\mathcal{S}}^{*} t \,\}.$$

We write $\mathit{RecTree}_n A$ for the class of value trees $[\![\, \mathcal{S} \,]\!]$, where $\mathcal{S}$ ranges over order-n recursion schemes.

Example 2.4. The value tree of the recursion scheme $\mathcal{S}$ of Example 2.3 is as follows:



Remark 2.3. The relation $\to_{\mathcal{S}}$ is unrestricted, in the sense that any ground subterm starting with a non-terminal can be rewritten. A more constrained rewriting policy referred to as *outermost-innermost (OI)* only allows rewriting a ground non-terminal subterm if it is not below any non-terminal symbols (i.e., it is outermost) [22]. The corresponding rewriting relation is denoted $\to_{\mathcal{S},\mathrm{OI}}$. Note that using $\to_{\mathcal{S},\mathrm{OI}}$ instead of $\to_{\mathcal{S}}$ does not change the value tree of the scheme, i.e., $\sup\{\, t^{\perp} \mid Z \to_{\mathcal{S}}^{*} t \,\} = \sup\{\, t^{\perp} \mid Z \to_{\mathcal{S},\mathrm{OI}}^{*} t \,\}$.

Another rewriting policy referred to as *innermost-outermost (IO)* only allows rewriting a ground non-terminal subterm if this subterm does not contain a ground non-terminal as subterm (i.e., it is innermost) [22]. The corresponding rewriting relation is denoted $\to_{\mathcal{S},\mathrm{IO}}$. Note that using $\to_{\mathcal{S},\mathrm{IO}}$ instead of $\to_{\mathcal{S}}$ may change the value tree of the scheme. Indeed, consider as an example the recursion scheme $\mathcal{S}'$ obtained from the scheme $\mathcal{S}$ in Example 2.3 by replacing its first production rule by the following two rules:

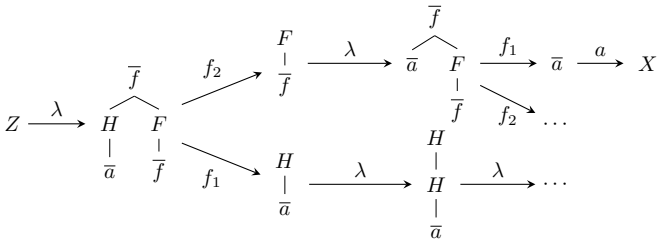
$$\begin{array}{rcl} Z & \to & K\,(H\,a)(F\,f) \\ K\,x\,y & \to & f\,x\,y \end{array}$$

Hence, we just added an intermediate non-terminal K, and one easily checks that $[\![\, \mathcal{S} \,]\!] = [\![\, \mathcal{S}' \,]\!]$. As the non-terminal H is not productive, following the IO policy, the second production rule will never be used, and therefore $\sup\{\, t^{\perp} \mid Z \to_{\mathcal{S}',\mathrm{IO}}^{*} t \,\} = \perp$.

2.3.5 Labeled recursion schemes A labeled recursion scheme is a recursion scheme without terminal symbols but whose productions are labeled by a finite alphabet. This slight variation in the definition allows us to associate a LTS with every labeled recursion scheme.

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

[3] A set D is directed if D is not empty and for all $x, y \in D$, there exists $z \in D$ such that $x \preccurlyeq z$ and $y \preccurlyeq z$)



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

Figure 3. A labeled recursion scheme generating the same term as the scheme of Example 2.3.

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

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

$$F\,t_1 \cdots t_{\varrho(F)} \xrightarrow{a} e[t_1/x_1, \ldots, t_{\varrho(F)}/x_{\varrho(F)}] \text{ if } F\,x_1 \cdots x_n \xrightarrow{a} e \text{ is a production rule.}$$

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 a 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 λ,
- 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 $\overrightarrow{f}$.

Recursion schemes and labeled recursion schemes are equi-expressive for generating terms.

Theorem 2.4. *The recursion schemes and the labeled recursion schemes generate the same terms. Moreover the translations are linear and preserves order and arity.*

Proof. Let $\mathcal{S} = \langle A, N, \mathcal{R}, Z, \bot \rangle$ be a recursion scheme. We define a labeled recursion scheme $\mathcal{S}' = \langle \overrightarrow{A}, N', \mathcal{R}', Z, \bot \rangle$ generating the term $[\![\mathcal{S}]\!]$. For each terminal symbol $f \in A$, we introduce a non-terminal symbol, denoted $\overline{f} : \underbrace{o \to \cdots \to o}_{\varrho(f)} \to o$. The set of non-terminal symbols of $\mathcal{S}'$ is $N \cup \{\overline{f} \mid f \in A\} \cup \{X\}$, where X is assumed to be a fresh non-terminal. With a term t over $A \cup N$, we associate the term $\overline{t}$ over N' obtained by replacing every occurrence of a terminal symbol f by its nonterminal counterpart $\overline{f}$. The production rules of $\mathcal{S}'$ are as follows:

$$\begin{array}{ll} & \{F\, x_1 \cdots x_n \xrightarrow{\lambda} \overline{e} \mid F\, x_1 \cdots x_n \longrightarrow e \in \mathcal{R}\} \\ \cup & \{\overline{f}\, x_1 \cdots x_{\varrho(f)} \xrightarrow{f_i} x_i \mid f \in A \text{ with } \varrho(f) > 0 \text{ and } i \in [1, \varrho(f)]\} \\ \cup & \{\overline{c} \xrightarrow{c} X \mid c \in A \text{ with } \varrho(c) = 0\}. \end{array}$$

Conversely, let A be ranked alphabet and let $\mathcal{S} = \langle \overrightarrow{A}, N, \mathcal{R}, Z, \bot \rangle$ be a labeled recursion scheme respecting the syntactic restrictions mentioned above. We define a recursion scheme $\mathcal{S}' = \langle A, N, \mathcal{R}', Z, \bot \rangle$ generating the same term as $\mathcal{S}$. The set of production rules of $\mathcal{S}'$ are defined as follows:

- If $F\, x_1 \cdots x_n \xrightarrow{\lambda} e$ belongs to $\mathcal{R}$ (in this case it is the only rule starting with F) then $F\, x_1 \cdots x_n \to e$ belongs to $\mathcal{R}'$.
- If, for some c of arity 0, $F\, x_1 \cdots x_n \xrightarrow{c} e$ belongs to $\mathcal{R}$ (in this case it is the only rule starting with F and e starts with a non-terminal that has no rule in $\mathcal{R}$) then $F\, x_1 \cdots x_n \to c$ belongs to $\mathcal{R}'$.
- If, for some $f \in A$ of arity $\varrho(f) > 0$, $F\, x_1 \cdots x_n \xrightarrow{f_i} e_i$ belongs to $\mathcal{R}$ for all $1 \leqslant i \leqslant \varrho(f)$, then $F\, x_1 \cdots x_n \to f\, e_1 \cdots e_{\varrho(f)}$ belongs to $\mathcal{R}'$.

□

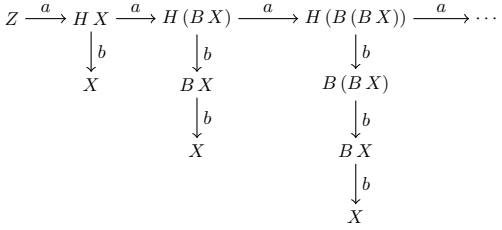
2.3.6 Examples of trees defined by labeled recursion schemes In this section, we provide some examples of trees defined by labeled recursion schemes. Given a language L over Σ, we let $\mathrm{Pref}(L)$ denote the tree in $\mathrm{Trees}^{\infty}(\Sigma)$ containing all prefixes of words in L.

The tree $\mathrm{Pref}(\{a^n b^n \mid n \geqslant 0\})$. Let us start with the tree T_0 corresponding to the deterministic context-free language $\mathrm{Pref}(\{a^n b^n \mid n \geqslant 0\})$. As is the case for all prefix-closed deterministic context-free languages (see [16, 17] or Theorem 4.8 at order 1) , T_0 is generated by an order-1 scheme $\mathcal{S}_0$.

$$\begin{array}{llll} Z & \xrightarrow{a} & H\, X & \qquad H\, x \xrightarrow{a} H\, (B\, x) \\ B\, x & \xrightarrow{b} & x & \qquad H\, x \xrightarrow{b} x \end{array}$$

with $Z, X : o$ and $H, B : o \to o$.

The tree generated by $\mathcal{S}_0$ is given below:



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

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

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

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 $t : o$, $K\,F_1\,F_2 t \xrightarrow{\lambda} F_1(F_2\,t)$). For any term $G : o \to o$, we define G^n for all $n \geqslant 0$ by taking $G^0 = I$ and $G^{n+1} = K\,G\,G^n$. For any ground term t, $G^n\,t$ behaves as $\underbrace{G\,(\cdots(G}_{n}(It))\cdots)$ and, in particular $B^n\,X \overset{b^n}{\Longrightarrow} X$.

For all $n \geqslant 0$, we have

$$Z \xrightarrow{a^n} F\,B^{n-1}\,C^n \xrightarrow{b} C^n(B^{n-1}\,X) \overset{b^{n-1}c^n}{\Longrightarrow} X.$$

The tree $\mathrm{Pref}(\{a^n c b^{2^n} \mid n \geqslant 0\})$**.** Following the same ideas as for $\mathcal{S}_1$, the order-2 scheme $\mathcal{S}_{\exp}$ given below defines the tree $T_{\exp} = \mathrm{Pref}(\{a^n c b^{2^n} \mid n \geqslant 0\})$.

$$\begin{array}{llll} Z & \xrightarrow{\lambda} & F\,B & \qquad F\,\varphi \xrightarrow{a} F\,(D\,\varphi) \qquad D\,\varphi\,x \xrightarrow{\lambda} \varphi\,(\varphi\,x) \\ B\,x & \xrightarrow{b} & x & \qquad F\,\varphi \xrightarrow{c} \varphi\,X \end{array}$$

with $Z, X : o$, $B : o \to o$, $D : (o \to o, o, o)$ and $F : (o \to o, o)$. If we let $\underline{D^n\,B}$ denote the term of type $o \to o$ defined by $\underline{D^0 B} = B$ and $\underline{D^{n+1}\,B} = D\,\underline{(D^n\,B)}$ for $n \geqslant 0$, we have $Z \overset{a^n}{\Longrightarrow} F\,\underline{D^n\,B}$. As, intuitively, D doubles its argument, $\underline{D^n\,B}$ behaves like B^{2^n} for $n \geqslant 0$. In particular, $\underline{D^n\,B}\,X$ reduces by b^{2^n} to X.
For all $n \geqslant 0$, we have

$$Z \overset{a^n}{\Longrightarrow} F\,\underline{D^n\,B} \xrightarrow{c} \underline{D^n\,B}\,X \overset{b^{2^n}}{\Longrightarrow} X.$$

The trees corresponding to the tower of exponentials of height k**.** At order $k+1 \geqslant 1$, we can define the tree $T_{\exp_k} = \mathrm{Pref}(\{a^n\,c\,b^{\exp_k(n)} \mid n \geqslant 0\})$ where we let $\exp_0(n) = n$ and $\exp_{k+1}(n) = 2^{\exp_k(n)}$ for $k \geqslant 0$. We illustrate the idea by giving an order-3 scheme

generating $T_{\exp_2} = \mathrm{Pref}(\{a^n\, c\, b^{2^{2^n}} \mid n \geqslant 0\})$.

$$\begin{array}{llll} Z & \xrightarrow{\lambda} & F\,D_1 & \qquad F\,\psi \xrightarrow{a} F\,(D_2\,\psi) \qquad D_2\,\psi\,\varphi\,x \xrightarrow{\lambda} (\psi(\psi\,\varphi))x \\ B\,x & \xrightarrow{b} & x & \qquad F\,\varphi \xrightarrow{c} \varphi\,B\,X \qquad D_1\,\psi\,x \xrightarrow{\lambda} \psi\,(\psi\,x) \end{array}$$

with $Z, X : o$, $B : o \to o$, $F : ((o \to o, o, o), o)$, $D_1 : (o \to o, o, o)$ and $D_2 : ((o \to o, o, o), o \to o, o, o)$. If we let $\underline{D_2^n\, D_1}$ denote the term of type $(o \to o, o, o)$ defined by $\underline{D_2^0\, D_1} = D_1$ and $\underline{D_2^{n+1}\, D_1} = D_2\, \underline{D_2^n\, D_1}$ for $n \geqslant 0$, we have $Z \overset{a^n}{\Longrightarrow} F\, \underline{D_2^n\, D_1}$. As D_2 intuitively double its argument with each application, $\underline{D_2^n\, D_1}$ behaves as $D_1^{2^n}$ and hence $D_1^{2^n}\, B$ behaves as $B^{2^{2^n}}$.
For all $n \geqslant 0$, we have

$$Z \overset{a^n}{\Longrightarrow} F\, \underline{D_2^n\, D_1} \xrightarrow{c} \underline{D_2^n\, D_1}\, B\, X \overset{b^{2^{2^n}}}{\Longrightarrow} X.$$

The tree of the Urzyczyn language

All schemes presented in this section satisfy a syntactic restriction, called the safety condition, that will be discussed in the last section of this chapter. Paweł Urzyczyn conjectured that (a slight variation) of the tree described below, though generated by a order-2 scheme, could not be generated by any order-2 scheme satisfying the safety condition. This conjecture was proved by Paweł Parys in [49].

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 greater 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 that 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.

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

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 $[\mathbf{n_1} \dots \mathbf{n_\ell}]$ of integers, we associate a term of type $o \to o$ by letting $[\,] = G$ and $[\mathbf{n_1} \dots \mathbf{n_\ell}\, \mathbf{n_{\ell+1}}] = F\,[\mathbf{n_1} \dots \mathbf{n_\ell}]\, \mathbf{n_{\ell+1}}$. Finally we write $([\mathbf{n_1} \dots \mathbf{n_\ell}], \mathbf{n})$ to denote the ground term $[\mathbf{n_1} \dots \mathbf{n_\ell}]\, \mathbf{n}$.

The scheme can be revisited as follows:

$$Z \xrightarrow{\lambda} ([\,],\mathbf{1}) \qquad ([\,],\mathbf{n+1}) \xrightarrow{\star} \mathbf{0} \qquad ([\mathbf{n_1}\dots\mathbf{n_\ell}],\mathbf{n}) \xrightarrow{\star} \mathbf{n_\ell} \qquad \mathbf{n+1} \xrightarrow{\star} \mathbf{n}$$
$$([\mathbf{n_1}\dots\mathbf{n_\ell}],\mathbf{n}) \xrightarrow{(} ([\mathbf{n_1}\dots\mathbf{n_\ell}\,\mathbf{n}],\mathbf{n+1})$$
$$([\mathbf{n_1}\dots\mathbf{n_\ell}],\mathbf{n}) \xrightarrow{)} ([\mathbf{n_1}\dots\mathbf{n_{\ell-1}}],\mathbf{n+1})$$

Let $w = w_0 \dots w_{|w|-1}$ be a prefix of a well-bracketed word. We have $Z \xRightarrow{w} ([\,\mathbf{n_1}\dots\mathbf{n_\ell}\,],|\mathbf{w}|+\mathbf{1})$ where $[\,n_1\dots n_\ell\,]$ is the sequence (in increasing order) of those indices of unmatched opening brackets in w. In turn, $([\,\mathbf{n_1}\dots\mathbf{n_\ell}\,],|\mathbf{w}|) \xrightarrow{\star} \mathbf{n_\ell} \xrightarrow{\star^{n_\ell}} \mathbf{0}$. 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.

2.4 Higher-order pushdown automata

2.4.1 Higher-order stack and their operations Higher-order pushdown automata were introduced by Maslov [45] as a generalisation of pushdown automata. First, recall that a (order-1) pushdown automaton is a machine with a finite control together with an auxiliary storage given by a (order-1) stack whose symbols are taken from a finite alphabet. A higher-order pushdown automaton is defined in a similar way, except that it uses a higher-order stack as auxiliary storage. Intuitively, an order-n stack is a stack whose base symbols are order-$(n-1)$ stacks, with the convention that order-1 stacks are just stacks in the classical sense.

Fix a finite stack alphabet Γ and a distinguished *bottom-of-stack symbol* $\perp \notin \Gamma$. An order-1 stack is a sequence $\perp, a_1, \dots, a_\ell \in \perp\Gamma^*$ which is denoted $[\perp a_1 \cdots a_\ell]_1$. An *order-$k$ stack* (or a *k-stack*), for $k > 1$, is a non-empty sequence $s_1, \dots, s_\ell$ of order-$(k-1)$ stacks which is written $[\,s_1 \cdots s_\ell\,]_k$. For convenience, we may sometimes see an element $a \in \Gamma$ as an order-0 stack, denoted $[\,a\,]_0$. We let Stacks_k denote 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 $\mathrm{ord}(s)$ the order of the stack s.

A *substack* of an order-1 stack $[\perp a_1 \cdots a_h]_1$ is a stack of the form $[\perp a_1 \cdots a_{h'}]_1$ for some $0 \leqslant h' \leqslant h$. A *substack* of an order-k stack $[\,s_1 \dots s_h\,]_k$, for $k > 1$, is either a stack of the form $[\,s_1 \dots s_{h'}\,]_k$ with $0 < h' \leqslant h$ or a stack of the form $[\,s_1 \dots s_{h'}\, s'\,]_k$ 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'.

Example 2.5. The stack

$$s = [\,[\,[\perp baac]_1\,[\perp bb]_1\,[\perp bcc]_1\,[\perp cba]_1\,]_2\,[\,[\perp baa]_1\,[\perp bc]_1\,[\perp bab]_1\,]_2\,]_3$$

is an order-3 stack of height 2.

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 $s = [\, s_1 \cdots s_\ell \,]_n$ 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:

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

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

$$\begin{aligned} \mathrm{top}_i([\, s_1 \cdots s_{\ell+1} \,]_n) &= \begin{cases} s_{\ell+1} & \text{if } i = n, \\ \mathrm{top}_i(s_{\ell+1}) & \text{if } i < n. \end{cases} \\ \mathrm{pop}_i([\, s_1 \cdots s_{\ell+1} \,]_n) &= \begin{cases} [\, s_1 \cdots s_\ell \,]_n & \text{if } i = n \text{ and } \ell \geqslant 1, \\ [\, s_1 \cdots s_\ell \, \mathrm{pop}_i(s_{\ell+1}) \,]_n & \text{if } i < n. \end{cases} \end{aligned}$$

By abuse of notation, we let $\mathrm{top}_{\mathrm{ord}(s)+1}(s) = s$. Note that $\mathrm{pop}_i(s)$ is defined if and only if the height of $\mathrm{top}_{i+1}(s)$ is strictly greater than 1. For example, $\mathrm{pop}_2([\,[\,\bot\, a\, b\,]_1\,]_2)$ 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 $\mathrm{push}_i(s) = s +\!\!+ \mathrm{top}_i(s)$.

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 $\mathrm{push}_1^a(s) = s +\!\!+ [\, a \,]_0$.

Example 2.6. Let s be the order-3 stack of Example 2.5. Then we have

$$\begin{aligned} \mathrm{top}_3(s) &= [\,[\,\bot baa\,]_1\,[\,\bot bc\,]_1\,[\,\bot bab\,]_1\,]_2, \\ \mathrm{pop}_3(s) &= [\,[\,[\,\bot baac\,]_1\,[\,\bot bb\,]_1\,[\,\bot bcc\,]_1\,[\,\bot cba\,]_1\,]_2\,]_3. \end{aligned}$$

Note that $\mathrm{pop}_3(\mathrm{pop}_3(s))$ is undefined.

We also have that

$$\begin{aligned} \mathrm{push}_2(\mathrm{pop}_3(s)) &= [\,[\,[\,\bot baac\,]_1\,[\,\bot bb\,]_1\,[\,\bot bcc\,]_1\,[\,\bot cba\,]_1\,[\,\bot cba\,]_1\,]_2\,]_3, \\ \mathrm{push}_1^c(\mathrm{pop}_3(s)) &= [\,[\,[\,\bot baac\,]_1\,[\,\bot bb\,]_1\,[\,\bot bcc\,]_1\,[\,\bot cbac\,]_1\,]_2\,]_3. \end{aligned}$$

2.4.2 Stacks with links and their operations We 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 the alphabet Γ are defined as order-n stacks[4]

[4]Note that we therefore slightly generalise our previous definition, as we implicitly use an infinite stack

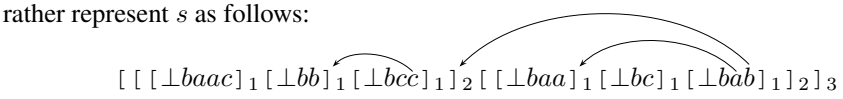
over alphabet $\Gamma \cup \Gamma \times \{2, \cdots, n\} \times \mathbb{N}$.

Example 2.7. The stack s equals to

$$[\,[\,[\,\bot baac\,]_1\,[\,\bot bb\,]_1\,[\,\bot bc(c,2,2)\,]_1\,]_2\,[\,[\,\bot baa\,]_1\,[\,\bot bc\,]_1\,[\,\bot b(a,2,1)(b,3,1)\,]_1\,]_2\,]_3$$

is an order-3 stack with links.

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:



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 $\mathrm{push}_1^{a,\ell}(s) = \mathrm{push}_1^{(a,\ell,h)}$ where we let $h = |\mathrm{top}_\ell(s)| - 1$ and require that $h > 1$.

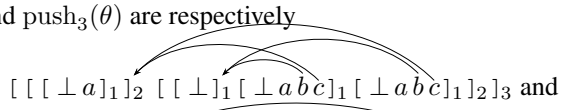
The collapse operation is defined only when the topmost symbol is the source of an ℓ-link, and results in truncating the topmost ℓ stack to only keep the component below the target of the link. Formally, if $\mathrm{top}_1(s) = (a, \ell, h)$ and $s = s' +\!\!+ [t_1 \cdots t_k]_\ell$ with $k > h$ we let $\mathrm{collapse}(s) = s' +\!\!+ [t_1 \cdots t_h]_\ell$.

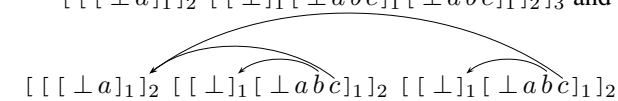
For any n, we let $\mathrm{Op}_n(\Gamma)$ denote the set of all operations over order-n stacks with links.

Example 2.8. Take the 3-stack $s = [\,[\,[\,\bot a\,]_1\,]_2\ [\,[\,\bot\,]_1\,[\,\bot a\,]_1\,]_2\,]_3$. We have

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

Then $\mathrm{push}_2(\theta)$ and $\mathrm{push}_3(\theta)$ are respectively





We have $\mathrm{collapse}\,(\mathrm{push}_2(\theta)) = \mathrm{collapse}\,(\mathrm{push}_3(\theta)) = \mathrm{collapse}(\theta) = [\,[\,[\,\bot a\,]_1\,]_2\,]_3$.

2.4.3 Higher-order pushdown automata and collapsible automata 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 λ, the set Γ is a stack alphabet, Q is a finite set of control states, $q_0 \in Q$ is the initial state, and

alphabet, but this does not introduce any technical change in the definition.

$\delta : Q \times \Gamma \times \Sigma \to Q \times \mathrm{Op}_n(\Gamma)$ is a (partial) transition function such that, for all $q \in Q$ and $\gamma \in \Gamma$, if $\delta(q, \gamma, \lambda)$ is defined then for all $a \neq \lambda$, the value $\delta(q, \gamma, a)$ is undefined, i.e., if some λ-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.

In the special case where $\delta(q, \gamma, \lambda)$ is undefined for all $q \in Q$ and $\gamma \in \Gamma$, we refer to $\mathcal{A}$ as a λ-free n-CPDA.

In the special case where $\mathrm{collapse} \notin \delta(q, \gamma, a)$ for all $q \in Q$, $\gamma \in \Gamma$ and $a \in \Sigma$, $\mathcal{A}$ is called a *higher-order pushdown automaton*.

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 let $\mathrm{Config}(\mathcal{A})$ denote the set of configurations of $\mathcal{A}$ and 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 $\delta(q, \mathrm{top}_1(s), a) = (q', op)$ and $s' = op(s)$.

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

3 From CPDA to recursion schemes

In this section, we argue that, for any CPDA $\mathcal{A}$, one can construct a labeled recursion scheme (of the same order) that generates the same tree. For this, we first introduce a representation of stacks and configurations of $\mathcal{A}$ by applicative terms. Then we define a labeled recursion scheme $\mathcal{S}$ and finally we show that the LTS associated with $\mathcal{S}$ is the same as the one associated with $\mathcal{A}$, which shows that $\mathcal{S}$ and $\mathcal{A}$ define the same tree.

For the rest of this section we fix an order-n CPDA $\mathcal{A} = \langle\, \Sigma, \Gamma, Q, \delta, q_1 \,\rangle$ and we let the state-set of $\mathcal{A}$ be $Q = \{q_1, \ldots, q_m\}$ where $m \geqslant 1$. In order to treat in a uniform way those stack symbols that come with a link and those that do not, we will attach fake links, which we refer to as 1-links (recall that so far all links were ℓ-links with $\ell > 1$) to those symbols that have no link; moreover $\mathrm{collapse}(s)$ will be undefined for any stack s such that $\mathrm{top}_1(s)$ has a 1-link. In the following, we therefore write $\mathrm{push}_1^{a,1}$ instead of push_1^a.

3.1 Term representation of stacks and configurations

We start by defining some useful types. First we identify the base type o with a new type denoted $\mathbf{n}$. Inductively, for each $0 \leqslant k < n$ we define a type

$$\mathbf{k} \quad = \quad (\mathbf{k+1})^m \to (\mathbf{k+1})$$

where, for types A and B, we write $A^m \to B$ as a shorthand for $\underbrace{A \to \cdots \to A}_{m} \to B$. In particular, for every $0 \leqslant k \leqslant n$, we have

$$\mathbf{k} \;=\; (\mathbf{k+1})^m \to (\mathbf{k+2})^m \to \cdots \to \mathbf{n}^m \to \mathbf{n}.$$

We also introduce, for every $1 \leqslant k \leqslant n$ a non-terminal Void_k of type $\mathbf{k}$.

Assume s is an order-n stack and p is a control state of $\mathcal{A}$. In the sequel, we will define, for every $0 \leqslant k \leqslant n$, a term $[|s|]_k^p : \mathbf{k}$ that represents the *behaviour* of the top_k stack in s. To understand why $[|s|]_k^p$ is of type $\mathbf{k}$ one can view an order-k stack as acting on order-$(k+1)$ stacks: for every order-$(k+1)$ stack we can build a new order-$(k+1)$ stack by pushing an order-k stack on top of it. This behaviour corresponds to the type $(\mathbf{k+1}) \to (\mathbf{k+1})$. However, for technical reasons, when dealing with control states and configurations, we need to work with m copies of each stack (one per control state). Hence we view a k-stack as mapping m copies of an order-$(k+1)$ stack to a single order-$(k+1)$ stack. This explains why $\mathbf{k}$ is defined to be $(\mathbf{k+1})^m \to (\mathbf{k+1})$.

For every stack symbol a, every $1 \leqslant \ell \leqslant n$ and every state $p \in Q$, we introduce a non-terminal

$$\mathcal{F}_p^{a,\ell} \ : \ \ell^m \to \mathbf{1}^m \to \cdots \to \mathbf{n}^m \to \mathbf{n}$$

For every $0 \leqslant k \leqslant n$, every state p and every order-n stack s whose top_1 symbol is some a with an ℓ-link, we define (inductively) the following term of order $\mathbf{k} = (\mathbf{k+1})^m \to \cdots \to \mathbf{n}^m \to \mathbf{n}$:

$$[|s|]_k^p = \mathcal{F}_p^{a,\ell}[|\mathrm{collapse}(s)|]_\ell^{q_1\cdots q_m} \\ [|\mathrm{pop}_1(s)|]_1^{q_1\cdots q_m}[|\mathrm{pop}_2(s)|]_2^{q_1\cdots q_m}\cdots[|\mathrm{pop}_k(s)|]_k^{q_1\cdots q_m}$$

where

- $[|t|]_h^{q_1\cdots q_m}$ is a shorthand for (the sequence) $[|t|]_h^{q_1}[|t|]_h^{q_2}\cdots[|t|]_h^{q_m}$
- $[|\mathrm{pop}_i(s)|]_i^{q_j} = \mathrm{Void}_i$ for all $j \in [1,m]$ if $\mathrm{pop}_i(s)$ is undefined.
- $[|\mathrm{collapse}(s)|]_1^{q_j} = \mathrm{Void}_1$ for all $j \in [1,m]$; note that it corresponds to the case where $\mathrm{top}_1(s)$ has a 1-link (i.e., a fake link); hence $\mathrm{collapse}(s)$ is undefined.

Note that the previous definition is well-founded, as every stack in the definition of $[|s|]_k^p$ has fewer symbols than s. Intuitively, $[|s|]_k^p$ represents the top k-stack of the configuration (p,s), i.e., $\mathrm{top}_{(k+1)}(s)$.

Example 3.1. Consider the following order-2 stack

$$s = [\,[\,\bot\, a\,]_1\,[\,\bot\, b\,]_1\,[\,\bot\, b\, c\,]_1\,]_2$$

and assume (for simplicity) that we have a unique control state p. Then one has

$$[|s|]_2^p = \mathcal{F}_p^{c,1}\ \ \mathrm{Void}_1\ \ (\mathcal{F}_p^{b,2}\ \zeta\ (\mathcal{F}_p^{\bot,1}\mathrm{Void}_1\mathrm{Void}_1))\ \ (\mathcal{F}_p^{b,2}\zeta\ (\mathcal{F}_p^{\bot,1}\mathrm{Void}_1\mathrm{Void}_1)\ \zeta))$$

where

$$\zeta = [|\,[\,[\,\bot\, a\,]_1\,]_2|]_2^p = \mathcal{F}_p^{a,1}\ \mathrm{Void}_1\ (\mathcal{F}_p^{\bot,1}\mathrm{Void}_1\mathrm{Void}_1)\ \mathrm{Void}_2.$$

Let s and t be two order-n stacks with links and let $k \geqslant 1$. We shall say that s and t are top_k-*identical* iff the following holds:

- s and t are top_1-identical if and only s and t have the same top_1 symbol with an ℓ-link (for some ℓ) and (if defined) $\mathrm{collapse}(s)$ and $\mathrm{collapse}(t)$ are $\mathrm{top}_{\ell+1}$-identical;
- and for $k > 1$, s and t are top_k-identical if and only for all $j \geqslant 0$, $\mathrm{pop}^j_{k-1}(s)$ is defined iff $\mathrm{pop}^j_{k-1}(t)$ is defined, and when defined, $\mathrm{pop}^j_{k-1}(s)$ and $\mathrm{pop}^j_{k-1}(t)$ are $\mathrm{top}_{(k-1)}$-identical.

Note that the previous definition is well founded, as it always refer to stacks with fewer symbols than s or t.

Lemma 3.1. *Let s and t be order-n stacks with links, and let $k \geqslant 0$. If s and t are $\mathrm{top}_{(k+1)}$-identical then $[|s|]^p_k = [|t|]^p_k$ for every state p.*

Proof. The proof is by induction on the maximal size (i.e., the number of stack symbols) of s and t, and once the maximal size is fixed we reason by induction on k.

The base case of s and t containing only the bottom-of-stack symbol is trivial. Hence assume that the property is established for any pair of stacks with less than N symbols for some $N > 0$, and consider two stacks s and t whose maximal size is $N + 1$. Assume that s and t are $\mathrm{top}_{(k+1)}$-identical for some $k \geqslant 0$.

If s and t are top_1-identical, then, by definition, we have that $\mathrm{top}_1(s) = (a, \ell, k)$ and $\mathrm{top}_1(t) = (a, \ell, k')$ for some $a \in \Gamma$, $1 \leqslant e \leqslant n$ and $k, k' \in \mathbb{N}$, and that (when defined) $\mathrm{collapse}(s)$ and $\mathrm{collapse}(t)$ are $\mathrm{top}_{\ell+1}$-identical. As $\mathrm{collapse}(s)$ and $\mathrm{collapse}(t)$ are both of size $\leqslant N$, we have, by induction hypothesis, that $[|\mathrm{collapse}(s)|]^{q_1 \cdots q_m}_\ell = [|\mathrm{collapse}(t)|]^{q_1 \cdots q_m}_\ell$. Thus it immediately follows that $[|s|]^p_0 = [|t|]^p_0$.

We now consider some $k \geqslant 0$ and assume that the property is established for any $h \leqslant k$. We consider the case $(k + 1)$ and thus assume that s and t are $\mathrm{top}_{(k+2)}$-identical: in particular $\mathrm{pop}_{(h-1)}(s)$ and $\mathrm{pop}_{(h-1)}(t)$ are also top_h-identical for any $h \leqslant (k + 1)$, and by induction hypothesis, we have, for any $h \leqslant k$ and any state q, that $[|s|]^q_h = [|t|]^q_h$. By definition, we also have that $\mathrm{top}_1(s) = (a, \ell, k)$ and $\mathrm{top}_1(t) = (a, \ell, k')$ for some $a \in \Gamma$, $1 \leqslant e \leqslant n$ and $k, k' \in \mathbb{N}$, and that (when defined) $\mathrm{collapse}(s)$ and $\mathrm{collapse}(t)$ are $\mathrm{top}_{(\ell+1)}$-identical. As $\mathrm{collapse}(s)$ and $\mathrm{collapse}(t)$ are both of size $\leqslant N$, we have, by induction hypothesis, that $[|\mathrm{collapse}(s)|]^{q_1 \cdots q_m}_\ell = [|\mathrm{collapse}(t)|]^{q_1 \cdots q_m}_\ell$.

We let $j_s = j_t$ be the maximal j such that $\mathrm{pop}^j_{(k+1)}(s)$ (equiv. $\mathrm{pop}^j_{(k+1)}(t)$) is defined. By definition

$$[|s|]^p_{(k+1)} = \mathcal{F}^{a,\ell}_p [|\mathrm{collapse}(s)|]^{q_1 \cdots q_m}_\ell [|\mathrm{pop}_1(s)|]^{q_1 \cdots q_m}_1 \cdots [|\mathrm{pop}_{(k+1)}(s)|]^{q_1 \cdots q_m}_{(k+1)},$$

and

$$[|t|]^p_{(k+1)} = \mathcal{F}^{a,\ell}_p [|\mathrm{collapse}(t)|]^{q_1 \cdots q_m}_\ell [|\mathrm{pop}_1(t)|]^{q_1 \cdots q_m}_1 \cdots [|\mathrm{pop}_{(k+1)}(t)|]^{q_1 \cdots q_m}_{(k+1)}.$$

Now if $j_s = 0$, we have

$$[|\mathrm{pop}_{(k+1)}(s)|]^{q_1 \cdots q_m}_{(k+1)} = [|\mathrm{pop}_{(k+1)}(t)|]^{q_1 \cdots q_m}_1 = \mathrm{Void}_{(k+1)} \cdots \mathrm{Void}_{(k+1)},$$

and thus $[|s|]^p_{(k+1)} = [|t|]^p_{(k+1)}$. If $j_s > 0$, we note that $j_{\mathrm{pop}_{(k+1)}(s)} = j_{\mathrm{pop}_{(k+1)}(t)} = j_s - 1$ and $\mathrm{pop}_{(k+1)}(s)$ and recall that $\mathrm{pop}_{(k+1)}(t)$ are $\mathrm{top}_{(k+2)}$-identical. Thus, by induction on j_s, we have that $[|\mathrm{pop}_{(k+1)}(s)|]^q_{(k+1)} = [|\mathrm{pop}_{(k+1)}(t)|]^q_{(k+1)}$ for any state q, and we conclude that $[|s|]^p_{(k+1)} = [|t|]^p_{(k+1)}$. □

3.2 The labeled recursion scheme associated with $\mathcal{A}$

We let $\mathcal{S} = \langle \Sigma, N, \mathcal{R}, Z, \bot \rangle$ where

$$N = \{\mathcal{F}_p^{a,\ell} \mid p \in Q,\ a \in \Gamma,\ \text{and } 1 \leqslant \ell \leqslant n\} \cup \{\mathrm{Void}_k \mid 0 \leqslant k \leqslant n\}.$$

The set of productions $\mathcal{R}$ contains the production $Z \xrightarrow[\mathcal{S}]{\lambda} [|\,[\cdots[\bot]_1\cdots]_n|]_n^{q_1}$ and the production $\mathcal{F}_p^{a,\ell}\,\overline{\Phi}\,\overline{\Psi_1}\cdots\overline{\Psi_n} \xrightarrow[\mathcal{S}]{a} \Xi_{q,op}$ if $\delta(p,a,a) = (q, op)$ and the term $\Xi_{q,op}$ is equal to:

$$\begin{array}{ll}
\mathcal{F}_q^{a',\ell'}\,\overline{\Psi_{\ell'}}\,\langle \mathcal{F}_\star^{a,\ell}\,\overline{\Phi}\,\overline{\Psi_1}\rangle\,\overline{\Psi_2}\cdots\overline{\Psi_n} & \text{if } op = \mathrm{push}_1^{a',\ell'} \text{ for } \ell' > 1,\\
\mathcal{F}_q^{a',1}\,\mathrm{Void}_1^m\,\langle \mathcal{F}_\star^{a,\ell}\,\overline{\Phi}\,\overline{\Psi_1}\rangle\,\overline{\Psi_2}\cdots\overline{\Psi_n} & \text{if } op = \mathrm{push}_1^{a',1},\\
\mathcal{F}_q^{a,\ell}\,\overline{\Phi}\,\overline{\Psi_1}\cdots\overline{\Psi_{(k-1)}}\langle \mathcal{F}_\star^{a,\ell}\,\overline{\Phi}\,\overline{\Psi_1}\cdots\overline{\Psi_k}\rangle\overline{\Psi_{(k+1)}}\cdots\overline{\Psi_n} & \text{if } op = \mathrm{push}_k,\\
\Psi_{k,q}\,\overline{\Psi_{k-1}}\cdots\overline{\Psi_n} & \text{if } op = \mathrm{pop}_k,\\
\Phi_q\,\overline{\Psi_{\ell-1}}\cdots\overline{\Psi_n} & \text{if } op = \mathrm{collapse} \text{ and } \ell > 1.
\end{array}$$

where $\langle \mathcal{F}_\star^{a,\ell}\,\overline{\Phi}\,\overline{\Psi_1}\cdots\overline{\Psi_k}\rangle$ as a shorthand for the sequence

$$\mathcal{F}_{q_1}^{a,\ell}\,\overline{\Phi}\,\overline{\Psi_1}\cdots\overline{\Psi_k}\quad \mathcal{F}_{q_2}^{a,\ell}\,\overline{\Phi}\,\overline{\Psi_1}\cdots\overline{\Psi_k}\quad\cdots\quad \mathcal{F}_{q_m}^{a,\ell}\,\overline{\Phi}\,\overline{\Psi_1}\cdots\overline{\Psi_k}$$

and Void_1^m is a a shorthand for $\underbrace{\mathrm{Void}_1\cdots\mathrm{Void}_1}_{m}$.

3.3 Correctness of the representation

The following proposition relates the LTS defined by $\mathcal{A}$ with the one defined by $\mathcal{S}$.

Proposition 3.2. *Let (p,s) be a configuration of $\mathcal{A}$ and let $a \in \Sigma$. Then*

$$(p,s) \xrightarrow[\mathcal{A}]{a} (q,t) \text{ if and only if } [|s|]_n^p \xrightarrow[\mathcal{S}]{a} [|t|]_n^q$$

Proof. Let a be the top symbol in s and let $0 \leqslant \ell \leqslant n$ be such that a has an $(\ell+1)$ link. By definition, the head non-terminal symbol of $[|s|]_n^p$ is $\mathcal{F}_p^{a,\ell}$.

Remark that $\delta(p,a,a)$ is defined, i.e., there exists some (q,t) with $(p,s) \xrightarrow[\mathcal{A}]{a} (q,t)$, iff there is some term ζ such that $[|s|]_n^p \xrightarrow[\mathcal{S}]{a} \zeta$. Hence it suffices to show, when $\delta(p,a,a) = (q, op)$ is defined, that $\zeta = [|op(s)|]_n^q$, and for this we do a case analysis.

First, we let

$$[|s|]_n^p = \mathcal{F}_p^{a,\ell}\quad C^{q_1}\cdots C^{q_m}\quad T_1^{q_1}\cdots T_1^{q_m}\quad\cdots\quad T_n^{q_1}\cdots T_n^{q_m}$$

where $C^{q_i} = [|\mathrm{collapse}(s)|]_\ell^{q_i} : \ell$ and $T_k^{q_i} = [|\mathrm{pop}_k(s)|]_k^{q_i} : \mathbf{k}$ for every $1 \leqslant i \leqslant m$ and every $1 \leqslant k \leqslant n$.

Then we distinguish the five possible cases for op.

- Assume that $op = \mathrm{push}_1^{a',\ell'}$, with $\ell' > 1$. Then, by definition we have

$$[|\mathrm{push}_1^{a',\ell'}(s)|]_n^q = \mathcal{F}_q^{a',\ell'}[|\mathrm{collapse}(\mathrm{push}_1^{a',\ell'}(s))|]_{\ell'}^{q_1\cdots q_m} \\ [|\mathrm{pop}_1(\mathrm{push}_1^{a',\ell'}(s))|]_1^{q_1\cdots q_m}\cdots[|\mathrm{pop}_n(\mathrm{push}_1^{a',\ell'}(s))|]_n^{q_1\cdots q_m}.$$

For every $j > 1$, one has $\mathrm{pop}_j(\mathrm{push}_1^{a',\ell'}(s)) = \mathrm{pop}_j(s)$, and therefore, we have

$$[|\mathrm{pop}_j(\mathrm{push}_1^{a',\ell'}(s))|]_j^{q_1\cdots q_m} = T_j^{q_1}\cdots T_j^{q_m}.$$

One has $\mathrm{collapse}(\mathrm{push}_1^{a',\ell'}(s)) = \mathrm{pop}_{\ell'}(s)$, and therefore, we have

$$[|\mathrm{collapse}(\mathrm{push}_1^{a',\ell'}(s))|]_{\ell'}^{q_1\cdots q_m} = T_{\ell'}^{q_1}\cdots T_{\ell'}^{q_m}.$$

Finally, we have that $\mathrm{pop}_1(\mathrm{push}_1^{a',\ell'}(s)) = s$, and therefore, we have

$$[|\mathrm{pop}_1(\mathrm{push}_1^{a',\ell'}(s))|]_1^{q_i} = \mathcal{F}_{q_i}^{a,\ell}C^{q_1}\cdots C^{q_m}T_1^{q_1}\cdots T_1^{q_m}.$$

Hence, it follows that

$$[|\mathrm{push}_1^{a',\ell'}(s)|]_n^q = \mathcal{F}_q^{a',\ell'}\quad T_{\ell'}^{q_1}\cdots T_{\ell'}^{q_m} \\ \mathcal{F}_{q_1}^{a,\ell}C^{q_1}\cdots C^{q_m}T_1^{q_1}\cdots T_1^{q_m}\quad\cdots\quad \mathcal{F}_{q_m}^{a,\ell}C^{q_1}\cdots C^{q_m}T_1^{q_1}\cdots T_1^{q_m} \\ T_2^{q_1}\cdots T_2^{q_m}\quad\cdots\quad T_n^{q_1}\cdots T_n^{q_m}.$$

On the other hand, it follows syntactically from the definition of $\mathcal{S}$ that the right hand side of the previous expression is the term ζ such that $[|s|]_n^p \xrightarrow[\mathcal{S}]{a} \zeta$.

- Assume that $op = \mathrm{push}_1^{a',1}$. Then, by definition we have

$$[|\mathrm{push}_1^{a',1}(s)|]_n^q = \mathcal{F}_q^{a',1}\mathrm{Void}_1\cdots\mathrm{Void}_1 \\ [|\mathrm{pop}_1(\mathrm{push}_1^{a',1}(s))|]_1^{q_1\cdots q_m}\cdots[|\mathrm{pop}_n(\mathrm{push}_1^{a',1}(s))|]_n^{q_1\cdots q_m}.$$

For every $j > 1$, one has $\mathrm{pop}_j(\mathrm{push}_1^{a',1}(s)) = \mathrm{pop}_j(s)$, and therefore

$$[|\mathrm{pop}_j(\mathrm{push}_1^{a',\ell'}(s))|]_j^{q_1\cdots q_m} = T_j^{q_1}\cdots T_j^{q_m}.$$

Finally, we have that $\mathrm{pop}_1(\mathrm{push}_1^{a',1}(s)) = s$, and therefore

$$[|\mathrm{pop}_1(\mathrm{push}_1^{a',1}(s))|]_1^{q_i} = \mathcal{F}_{q_i}^{a,\ell}C^{q_1}\cdots C^{q_m}T_1^{q_1}\cdots T_1^{q_m}.$$

Hence, it follows that

$$[|\mathrm{push}_1^{a',\ell'}(s)|]_n^q = \mathcal{F}_q^{a',\ell'}\quad \mathrm{Void}_1\cdots\mathrm{Void}_1 \\ \mathcal{F}_{q_1}^{a,\ell}C^{q_1}\cdots C^{q_m}T_1^{q_1}\cdots T_1^{q_m}\quad\cdots\quad \mathcal{F}_{q_m}^{a,\ell}C^{q_1}\cdots C^{q_m}T_1^{q_1}\cdots T_1^{q_m} \\ T_2^{q_1}\cdots T_2^{q_m}\quad\cdots\quad T_n^{q_1}\cdots T_n^{q_m}.$$

On the other hand, it follows syntactically from the definition of $\mathcal{S}$ that the right hand side of the previous expression is the term ζ such that $[|s|]_n^p \xrightarrow[\mathcal{S}]{a} \zeta$.

- Assume that $op = \mathrm{push}_k$. Then, by definition we have

$$[|\mathrm{push}_k(s)|]^q_n = \mathcal{F}^{a,\ell}_q [|\mathrm{collapse}(\mathrm{push}_k(s))|]^{q_1\cdots q_m}_\ell \\ [|\mathrm{pop}_1(\mathrm{push}_k(s))|]^{q_1\cdots q_m}_1 \cdots [|\mathrm{pop}_n(\mathrm{push}_k(s))|]^{q_1\cdots q_m}_n.$$

Note that we used the fact that the top_1 element in $\mathrm{push}_k(s)$ is a a and has an $(\ell+1)$-link. Now, note that for every $j > k$, one has $\mathrm{pop}_j(\mathrm{push}_k(s)) = \mathrm{pop}_j(s)$, and therefore

$$[|\mathrm{pop}_j(\mathrm{push}_k(s))|]^{q_1\cdots q_m}_j = T^{q_1}_j \cdots T^{q_m}_j.$$

We also have that $\mathrm{pop}_k(\mathrm{push}_k(s)) = s$, and therefore, for every $1 \leqslant i \leqslant m$,

$$[|\mathrm{pop}_k(\mathrm{push}_k(s))|]^{q_i}_k = \mathcal{F}^{a,\ell}_{q_i} C^{q_1} \cdots C^{q_m} T^{q_1}_1 \cdots T^{q_m}_1 \cdots T^{q_1}_k \cdots T^{q_m}_k.$$

Now, for $j < k$, $\mathrm{pop}_j(\mathrm{push}_k(s))$ and $\mathrm{pop}_j(s)$ are top_{j+1}-identical and, thanks to Lemma 3.1, we have that $[|\mathrm{pop}_j(\mathrm{push}_k(s))|]^{q_i}_j = [|s|]^{q_i}_j = T^{q_i}_j$.
If $\ell = 1$, both $\mathrm{collapse}(s)$ and $\mathrm{collapse}(\mathrm{push}_k(s))$ are undefined, hence we have $[|\mathrm{collapse}(\mathrm{push}_k(s))|]^{q_i}_\ell = \mathrm{Void}_1 = C^{q_i}$.
If $1 < \ell \leqslant k$, $\mathrm{collapse}(\mathrm{push}_k(s))$ and s are $\mathrm{top}_{\ell+1}$-identical and, thanks to Lemma 3.1 we have that $[|\mathrm{collapse}(\mathrm{push}_k(s))|]^{q_i}_\ell = [|\mathrm{collapse}(s)|]^{q_i}_\ell = C^{q_i}$.
If $\ell > k$, $\mathrm{collapse}(s) = \mathrm{collapse}(\mathrm{push}_k(s))$ hence $[|\mathrm{collapse}(\mathrm{push}_k(s))|]^{q_i}_\ell = C^{q_i}$.
Therefore, it follows that

$$[|\mathrm{push}_k(s)|]^q_n = \mathcal{F}^{a,\ell}_q C^{q_1} \cdots C^{q_m} T^{q_1}_1 \cdots T^{q_m}_1 \ \cdots\ T^{q_1}_{(k-1)} \cdots T^{q_m}_{(k-1)} \\ \mathcal{F}^{a,\ell}_{q_1} C^{q_1} \cdots C^{q_m} T^{q_1}_1 \cdots T^{q_m}_1 \cdots T^{q_1}_k \cdots T^{q_m}_k \cdots \\ \mathcal{F}^{a,\ell}_{q_m} C^{q_1} \cdots C^{q_m} T^{q_1}_1 \cdots T^{q_m}_1 \cdots T^{q_1}_k \cdots T^{q_m}_k \\ T^{q_1}_{(k+1)} \cdots T^{q_m}_{(k+1)} \ \cdots\ T^{q_1}_n \cdots T^{q_m}_n.$$

On the other hand, it follows syntactically from the definition of $\mathcal{S}$ that the right hand side of the previous expression is the term ζ such that $[|s|]^p_n \xrightarrow[\mathcal{S}]{a} \zeta$.

- Assume that $op = \mathrm{pop}_k$. Then, by definition we have

$$[|\mathrm{pop}_k(s)|]^q_n = \mathcal{F}^{a',\ell'}_q [|\mathrm{collapse}(\mathrm{pop}_k(s))|]^{q_1\cdots q_m}_\ell \\ [|\mathrm{pop}_1(\mathrm{pop}_k(s))|]^{q_1\cdots q_m}_1 \cdots [|\mathrm{pop}_n(\mathrm{pop}_k(s))|]^{q_1\cdots q_m}_n$$

where the top_1 element in $\mathrm{pop}_k(s)$ is a a' and has an $(\ell'+1)$-link. Equivalently, we have

$$[|\mathrm{pop}_k(s)|]^q_n = [|\mathrm{pop}_k(s)|]^q_k \\ [|\mathrm{pop}_{(k+1)}(\mathrm{pop}_k(s))|]^{q_1\cdots q_m}_{(k+1)} \cdots [|\mathrm{pop}_n(\mathrm{pop}_k(s))|]^{q_1\cdots q_m}_n.$$

For every $j > k$, one has $\mathrm{pop}_j(\mathrm{pop}_k(s)) = \mathrm{pop}_j(s)$, and therefore, we have $[|\mathrm{pop}_j(\mathrm{pop}_k(s))|]^{q_1\cdots q_m}_j = T^{q_1}_j \cdots T^{q_m}_j$. Moreover, by definition we have that $T^q_k = [|\mathrm{pop}_k(s)|]^q_k$. Hence, it follows that

$$[|\mathrm{pop}_k(s)|]^q_n = T^q_k T^{q_1}_{k+1} \cdots T^{q_m}_{k+1} \cdots T^{q_1}_n \cdots T^{q_m}_n.$$

On the other hand, it follows syntactically from the definition of $\mathcal{S}$ that the right hand side of the previous expression is the term ζ such that $[|s|]_n^p \xrightarrow[\mathcal{S}]{a} \zeta$.

- Assume that $op = \mathrm{collapse}$. Then, by definition we have

$$[|\mathrm{collapse}(s)|]_n^q = \mathcal{F}_q^{a',\ell'}[|\mathrm{collapse}(\mathrm{collapse}(s))|]_\ell^{q_1\cdots q_m} \\ [|\mathrm{pop}_1(\mathrm{collapse}(s))|]_1^{q_1\cdots q_m}\cdots[|\mathrm{pop}_n(\mathrm{collapse}(s))|]_n^{q_1\cdots q_m}$$

where the top_1 element in $\mathrm{collapse}(s)$ is a' and has an $(\ell'+1)$-link. Equivalently, one has

$$[|\mathrm{collapse}(s)|]_n^q = [|\mathrm{collapse}(s)|]_\ell^q \\ [|\mathrm{pop}_{(\ell+1)}(\mathrm{collapse}(s))|]_{(k+1)}^{q_1\cdots q_m}\cdots[|\mathrm{pop}_n(\mathrm{collapse}(s))|]_n^{q_1\cdots q_m}.$$

For every $j > e$, one has $\mathrm{pop}_j(\mathrm{collapse}(s)) = \mathrm{pop}_j(s)$, and therefore we have $[|\mathrm{pop}_j(\mathrm{collapse}(s))|]_j^{q_1\cdots q_m} = T_j^{q_1}\cdots T_j^{q_m}$. Moreover, by definition we have that $C^q = [|\mathrm{collapse}(s)|]_\ell^q$.
On the other hand, it follows syntactically from the definition of $\mathcal{S}$ that the right-hand side of the previous expression is the term ζ such that $[|s|]_n^p \xrightarrow[\mathcal{S}]{a} \zeta$.

□

Corollary 3.3. *The* LTS *defined by $\mathcal{A}$ is isomorphic to the one defined by $\mathcal{S}$. In particular, $\mathcal{A}$ and $\mathcal{S}$ generate the same tree.*

Proof. Immediate from Proposition 3.2. □

4 From recursion schemes to collapsible pushdown automata

In this section, 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., $\mathrm{Tree}^\perp(\mathcal{S}) = \mathrm{Tree}^\perp(\mathcal{A})$. Recall that a silent production rule is a production rule labeled by λ. To simplify the presentation we assume that $\mathcal{S}$ does not contain any such production rule. If $\mathcal{S}$ were to contain silent transitions, we would treat the symbol λ as any other symbol[5] 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 we associate a ground term over N denoted by $[\![\, s \,]\!]$ with a configuration of the form $(q_\star, s)$. Other configurations correspond to internal steps of the simulation and are only the source of silent transitions.

[5]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 λ is such that $\mathrm{Tree}^\perp(\mathcal{S}) = \mathrm{Tree}^\perp(\mathcal{A})$.

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

Hence, the main ingredient of the construction is the partial mapping $[\![\cdot]\!]$ associating a ground term over N with an order-n stack. 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 4.1, we present the mapping $[\![\cdot]\!]$ together with its basic properties; in Section 4.2, 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., $x = x_{\mathrm{rk}(x)}$).

Example 4.1. Throughout the whole section, we will illustrate definitions and constructions using the order-2 scheme $\mathcal{S}_U$ generating the tree T_U presented at the end of Section 2.3.6 as a running example. We recall its definition below.

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

with $Z, X : o, G, H : o \to o$ and $F : (o \to o, o, o)$. We have $\mathrm{rk}(\varphi) = \mathrm{rk}(z) = \mathrm{rk}(u) = 1$, $\mathrm{rk}(x) = 2$ and $\mathrm{rk}(y) = 3$.

4.1 Stacks representing terms.

The stack alphabet Γ consists of the initial symbol and of the right-hand sides of the production rules in $\mathcal{R}$ and their argument subterms (cf. Section 2.3.1), i.e., $\Gamma \stackrel{\text{def}}{=} \{ Z \} \cup \bigcup_{F\, x_1 \cdots x_{\varrho(x)} \xrightarrow{a} e} \{ e \} \cup \mathrm{ASubs}(e)$.

Example 4.2. For the scheme $\mathcal{S}_U$, one gets the following stack alphabet:

$$
\begin{array}{rcl}
\Gamma & = & \{\, Z\,,\, G\,(H\,X)\,,\, H\,X\,,\, X\,,\, F\,(F\,\varphi\,x)\,y\,(Hy)\,,\, F\,\varphi\,x\,,\, Hy\,,\, F\,G\,z\,(Hz)\,, \\
 & & \quad G\,,\, Hz\,,\, \varphi\,(Hy)\} \cup \{x, y, z, u, \varphi\}.
\end{array}
$$

Notation 4.1. 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., $\varphi \in V$).

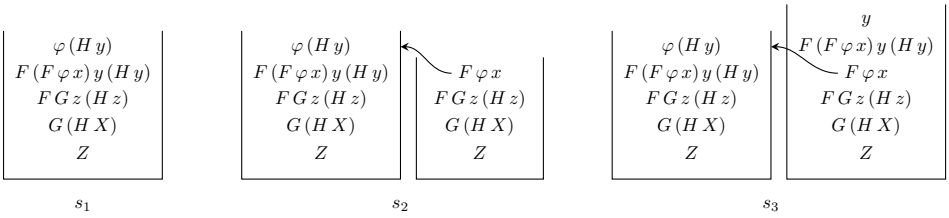
In order to represent a term in $\mathrm{Terms}(N)$, a stack over Γ must be *well-formed*, i.e., it must satisfy syntactic conditions given in the following definition.

Definition 4.2 (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 then $\mathrm{pop}_1(r)$ 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 $\mathrm{collapse}(r)$ is a φ-stack for some variable $\varphi \in V$ of type τ.

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

Example 4.3. For the scheme $\mathcal{S}_U$, the following order-2 stacks are well-formed.



Notation 4.3. 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 $\mathrm{pop}_1(s) :: top_1(s)$. We let $\mathrm{CStacks}$ denote 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 4.5, we will associate a ground term over N with any well-formed stack s that we refer to as the *value* of s. To define this value, we first associate, with any element $s :: t$ in $\mathrm{CStacks}$, 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.

For all $\varphi \in V \cup N$, all $k \in [1, \varrho(\varphi)]$ and all φ-stack $s \in \mathrm{WStacks}$, we define an element of $\mathrm{CStacks}$, denoted $\mathrm{Arg}_k(s)$, 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

$$\begin{cases} \mathrm{Arg}_k(s) & = & \mathrm{pop}_1(s) :: t_k & \text{if } k \leqslant \ell, \\ \mathrm{Arg}_k(s) & = & \mathrm{Arg}_{k-\ell}(\mathrm{collapse}(s)) & \text{otherwise.} \end{cases}$$

Definition 4.4. For all $s :: t \in \mathrm{CStacks}$, we take:

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

Let us provide some intuition regarding the definition of $[\![s :: t]\!]$. Unsurprisingly, $[\![s :: t]\!]$ is defined by structural induction on t, and the induction 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 the $\mathrm{rk}(x)$-th position in the left-hand side $A\,x_1 \cdots x_{\varrho(A)}$. As $s :: t \in \mathrm{CStacks}$, $top_1(s)$ is of the form $At_1 \cdots t_\ell$ 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 — corresponding to the two cases in the definition of $\mathrm{Arg}_k(s)$ — depending on whether x references 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 $\mathrm{pop}_1(s)$, i.e., $[\![s :: x]\!] = [\![\mathrm{pop}_1(s) :: t_{\mathrm{rk}(x)}]\!]$.
- 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, $\mathrm{collapse}(s)$, the stack obtained by following the link, has a top-symbol of the form $\varphi\, t'_1 \cdots t'_m$ 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 $[\![\mathrm{pop}_1(\mathrm{collapse}(s)) :: t'_{\mathrm{rk}(x)-\ell}]\!]$. If $\mathrm{rk}(x)$ is greater than $\ell + m$ then the link attached to the top symbol of $\mathrm{collapse}(s)$ 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 all its $\varrho(\varphi)$ arguments to the value of φ in the context of $\mathrm{pop}_1(s)$. This leads to the following formal definition.

Definition 4.5. The term associated with a well-formed φ-stack $s \in \mathrm{WStacks}$ with $\varphi \in N \cup V$ is

$$[\![s]\!] \stackrel{\mathrm{def}}{=} [\![\mathrm{pop}_1(s) :: \varphi]\!][\![\mathrm{Arg}_1(s)]\!] \cdots [\![\mathrm{Arg}_{\varrho(\varphi)}(s)]\!].$$

Fact 2. Let s be a well-formed φ-stack. If $top_1(s) : o$ then

$$[\![s]\!] = [\![\mathrm{pop}_1(s) :: top_1(s)]\!].$$

If $top_1(s) : \tau_1 \to \cdots \to \tau_\ell \to \mathrm{o}$ then

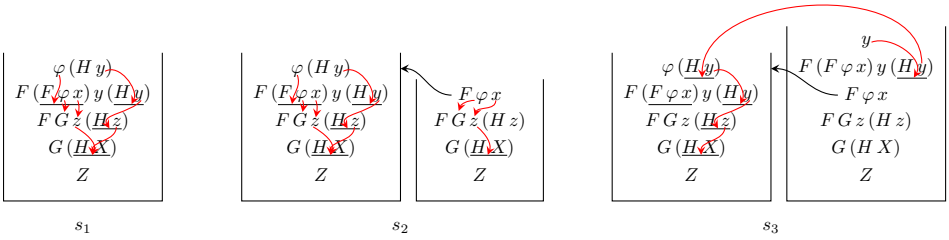
$$[\![s]\!] = [\![\mathrm{pop}_1(s) :: top_1(s)]\!] [\![\mathrm{Arg}_1(\mathrm{collapse}(s))]\!] \cdots [\![\mathrm{Arg}_\ell(\mathrm{collapse}(s))]\!].$$

Proof. The first case (i.e., $top_1(s) : o$) is immediate. Assume that $top_1(s)$ is equal to $\varphi\, t_1 \cdots t_n$ with $\varphi \in N \cup V$ of type $\tau_1 \to \cdots \to \tau_{\varrho(\varphi)} \to \mathrm{o}$ and $t_i \in \Gamma$ of type τ_i, for all $i \in [1, n]$. Note that $\ell = \varrho(\varphi) - n$. We have

$$\begin{aligned} [\![s]\!] &\stackrel{\mathrm{def}}{=} \underbrace{[\![\mathrm{pop}_1(s) :: \varphi]\!][\![\mathrm{Arg}_1(s)]\!] \cdots [\![\mathrm{Arg}_n(s)]\!]}_{[\![\mathrm{pop}_1(s) :: \varphi\, t_1 \cdots t_n]\!]} [\![\mathrm{Arg}_{n+1}(s)]\!] \cdots [\![\mathrm{Arg}_{\varrho(\varphi)}(s)]\!] \\ &= [\![\mathrm{pop}_1(s) :: top_1(s)]\!] [\![\mathrm{Arg}_1(\mathrm{collapse}(s))]\!] \cdots [\![\mathrm{Arg}_\ell(\mathrm{collapse}(s))]\!]. \end{aligned}$$

□

Example 4.4. Let us consider the well-formed stacks s_1, s_2, and s_3 presented in Example 4.3. In the representation below, the association between variables and their "values" are made explicit by the red arrows.



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

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

Lemma 4.1. *We have the following properties:*

(1) For all φ-stacks $s \in \mathrm{WStacks}$ with $\varphi \in V \cup N$ of type $\tau_1 \to \cdots \to \tau_{\varrho(\varphi)} \to \mathrm{o}$ and for all $k \in [1, \varrho(\varphi)]$, the stack $\mathrm{Arg}_k(s)$ is equal to some $r :: t \in \mathrm{CStacks}$ with t of type τ_k.

(2) For all $s :: t \in \mathrm{CStacks}$ with $t : \tau \in \Gamma$, the term $[\![\, s :: t \,]\!]$ belongs to $\mathrm{Terms}_\tau(N)$.

(3) For all $s \in \mathrm{WStacks}$, the term $[\![\, s \,]\!]$ belongs to $\mathrm{Terms}(N)$.

Proof. We start proving the first point and then use it to obtain the second one. Combining them, we finally prove the last point.

(1) We proceed by induction on the size of $s \in \mathrm{WStacks}$. The base case considers the stack $[\cdots[\bot Z]_1 \cdots]_n$. As $\varrho(Z) = 0$, there is nothing to prove.

Fix some stack s and assume that the property holds for all stacks smaller than $s \in \mathrm{WStacks}$. Let $\varphi\, t_1 \cdots t_\ell : \tau$ be the top symbol of s with $\varphi \in N \cup V$ and $t_i \in \Gamma$ for all $i \in [1, \ell]$. If φ is of type $\tau_1 \to \cdots \to \tau_{\varrho(\varphi)} \to \mathrm{o}$ then for all $i \in [1, \ell]$, t_i is of type τ_i and τ is the type $\tau_{\ell+1} \to \cdots \to \tau_{\varrho(\varphi)} \to \mathrm{o}$.

If $k \leqslant \ell$, then $\mathrm{Arg}_k(s) \stackrel{\mathrm{def}}{=} \mathrm{pop}_1(s) :: t_k$ and there is nothing to prove. If $\varrho(\varphi) \geqslant k > \ell$, then $\mathrm{Arg}_k(s) \stackrel{\mathrm{def}}{=} \mathrm{Arg}_{k-\ell}(\mathrm{collapse}(s))$. To conclude the result by induction, the only thing we have to prove is that $\mathrm{Arg}_{k-\ell}(\mathrm{collapse}(s))$ is well defined. As $\mathrm{ord}(\tau) > 0$, we have by definition of WStacks that $\mathrm{collapse}(s)$ is well-defined and that its top symbol starts with a symbol ψ of type τ. As $|\mathrm{collapse}(s)| < |s|$ and as $\varrho(\psi) = \varrho(\varphi) - \ell \geqslant k - \ell \geqslant 1$, we have by the induction hypothesis that $\mathrm{Arg}_{k-\ell}(\mathrm{collapse}(s))$ is well-defined and is equal to some $r :: t \in \mathrm{CStacks}$ with $t \in \Gamma$ of type $\tau_{k-\ell+\ell} = \tau_k$.

(2) We proceed by induction on the size of $s :: t$. The base case deals with the stack $[\cdots[\,\bot\,]_1 \cdots]_n :: Z$. As $[\![\,[\,]_n :: Z\,]\!] \stackrel{\mathrm{def}}{=} Z$, the property holds.

Assume that the property holds for all elements of CStacks smaller than some $s :: t \in \mathrm{CStacks}$ with $t : \tau$. Let us show that $[\![\, s :: t \,]\!]$ is of type τ. The case where $t \in N$ is trivial. The one where $t = t_1 t_2$ is immediate by induction, as both $[\![\, s :: t_2 \,]\!]$ and $[\![\, s :: t_1 \,]\!]$ have a size smaller than $[\![\, s :: t \,]\!]$. The last case is when t is a variable $x \in V$. Assume that the variable x appears in an A-production for some $A : \tau = \tau_1 \to \cdots \to \tau_{\varrho(A)} \to \mathrm{o}$ in N. In particular, the variable x is of type $\tau_{\mathrm{rk}(x)}$. We have $[\![\, s :: x \,]\!] \stackrel{\mathrm{def}}{=} [\![\, \mathrm{Arg}_{\mathrm{rk}(x)}(s) \,]\!]$. By definition of CStacks, s is an A-stack and using point (1), $\mathrm{Arg}_{\mathrm{rk}(x)}(s)$ is equal to

$r :: t'$ with $r \in Stacks$ and $t' : \tau_{\mathrm{rk}(x)} \in \Gamma$. Thus $[\![s :: x]\!] = [\![r :: t']\!]$ for some r smaller than s, and using the induction hypothesis, one concludes that $[\![s :: x]\!]$ is a term in $\mathrm{Terms}_{\tau_{\mathrm{rk}(x)}}(N)$.

(3) Let $s \in \mathrm{WStacks}$ whose top symbol starts with $\varphi : \tau = \tau_1 \to \cdots \to \tau_{\varrho(\varphi)} \to \mathrm{o}$. Clearly $\mathrm{pop}_1(s) :: \varphi$ belongs to CStacks and by point (2), $[\![\mathrm{pop}_1(s) :: \varphi]\!]$ is of type τ. Points (1) and (2) imply that, $[\![\mathrm{Arg}_k(s)]\!]$ is of type τ_k, for all $k \in [1, \varrho(\varphi)]$. Hence, from Definition 4.5 it directly follows that $[\![s]\!]$ is of type o. □

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 $s \in \mathrm{WStacks}$ with $\varphi \in N \cup V$, we have $[\![\mathrm{Arg}_\ell(\mathrm{push}_k(s))]\!] = [\![\mathrm{Arg}_\ell(s)]\!]$ for all $\ell \in [1, \varrho(\varphi)]$ and all $k \in [2, m]$. This follows (by letting $r = \mathrm{top}_k(s)$) from the following slightly more general result.

Lemma 4.2. *Let $k \in [2, m]$ and let $s = s' +\!\!+ \mathrm{top}_k(s) \in \mathrm{WStacks}$. For all non-empty φ-stacks $r \sqsubseteq \mathrm{top}_k(s)$, we have $[\![Arg_\ell(s' +\!\!+ r)]\!] = [\![Arg_\ell(s +\!\!+ r)]\!]$ for all $\ell \in [1, \varrho(\varphi)]$.*

Proof. We show, by induction on the size of r, that $s +\!\!+ r$ and $s' +\!\!+ r$ are well-formed and $[\![\mathrm{Arg}_\ell(s' +\!\!+ r)]\!] = [\![\mathrm{Arg}_\ell(s +\!\!+ r)]\!]$ for all $\ell \in [1, \varrho(\varphi)]$, where $\varphi \in N \cup V$ denotes the head symbol of $top_1(r)$.

The base case (which considers $[\cdots [\bot Z]_1 \cdots]_k$) is immediate. Assume that the property holds for all substacks of $\mathrm{top}_k(s)$ smaller than some φ-stack $r \sqsubseteq \mathrm{top}_k(s)$. We will show that it holds for r.

The key observation is that: $\mathrm{top}_2(s +\!\!+ r) = \mathrm{top}_2(s' +\!\!+ r)$ and either

$$\mathrm{collapse}(s +\!\!+ r) = \mathrm{collapse}(s +\!\!+ r)$$

if the link attached to topmost symbol of r is order greater than k, or

$$\mathrm{collapse}(s +\!\!+ r) = s +\!\!+ \mathrm{collapse}(r) \text{ and } \mathrm{collapse}(s' +\!\!+ r) = s' +\!\!+ \mathrm{collapse}(r)$$

otherwise.

As $s' +\!\!+ r$ is a substack of s (which is well-formed), $s' +\!\!+ r$ is well-formed as well. To prove that $s +\!\!+ r$ is well-formed, we need to show that every non-empty substack of $s +\!\!+ r$ satisfies the two properties expressed in Definition 4.2. The case of a proper substack immediately follows the induction hypothesis. We can deduce that $s +\!\!+ r$ satisfies these two properties from the above observations. Indeed the first property only depends on the top most order-1 stack (and $\mathrm{top}_2(s +\!\!+ r) = \mathrm{top}_2(s' +\!\!+ r)$) and the second property follows from the fact that $top_1(s +\!\!+ r) = top_1(s' +\!\!+ r)$ and $top_1(\mathrm{collapse}(s +\!\!+ r)) = top_1(\mathrm{collapse}(s' +\!\!+ r))$.

Assume that the top symbol of r is equal to $\varphi\, t_1 \cdots t_n$. Let $\ell \in [1, \varrho(\varphi)]$ and let us show that $[\![\mathrm{Arg}_\ell(s +\!\!+ r)]\!] = [\![\mathrm{Arg}_\ell(s' +\!\!+ r)]\!]$.

If $\ell \leqslant n$ then we have $[\![\mathrm{Arg}_\ell(s +\!\!+ r)]\!] = [\![s +\!\!+ \mathrm{pop}_1(r) :: t_\ell]\!] = [\![s' +\!\!+ \mathrm{pop}_1(r) :: t_\ell]\!]$. By the induction hypothesis, we have that $[\![s +\!\!+ r' :: t]\!] = [\![s' +\!\!+ r' :: t]\!]$ for any proper substack r' of r, and in particular for $r' = \mathrm{pop}_1(r)$.

If $\ell > n$ then $[\![\mathrm{Arg}_\ell(s +\!\!+ r)]\!]$ is equal to both $[\![\mathrm{Arg}_{\ell-n}(\mathrm{collapse}(s +\!\!+ r))]\!]$ and $[\![\mathrm{Arg}_{\ell-n}(\mathrm{collapse}(s +\!\!+ r))]\!]$. From the above observation, we either have that the stack $\mathrm{collapse}(s +\!\!+ r)$ is equal to $\mathrm{collapse}(s' +\!\!+ r)$ and the equality trivially holds, or we have $\mathrm{collapse}(s +\!\!+ r) = s +\!\!+ \mathrm{collapse}(r)$ and $\mathrm{collapse}(s' +\!\!+ r) = s' +\!\!+ \mathrm{collapse}(r)$ in which case the equality follows by the induction hypothesis as $|\,\mathrm{collapse}(r)\,| < |\,r\,|$. □

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

Lemma 4.3. *Let s be a φ-stack in* WStacks *for some $\varphi : \tau_1 \to \cdots \to \tau_{\varrho(\varphi)} \to \mathrm{o}$ in $V \cup N$ and let $\ell \in [1, \varrho(\varphi)]$ with τ_ℓ of order $k > 0$. If $\mathrm{Arg}_\ell(s)$ is equal to $r :: t \in$ CStacks with t starting with $\psi \in N \cup V$ then*

$$\mathrm{pop}_{n-k+1}(s) = \mathrm{pop}_{n-k+1}(r) \text{ and } |\,\mathrm{top}_{n-k+1}(s)\,| > |\,\mathrm{top}_{n-k+1}(r)\,|.$$

Proof. We proceed by induction of the size of s. The base case, which considers the stack $[\cdots[\,\bot Z\,]_1\cdots]_n$, is immediate as $\varrho(Z) = 0$.

Assume that the property holds for all stacks in WStacks smaller than some stack $s \in$ WStacks. Let $\varphi\, t_1 \dots t_m$ be the top symbol of s with $\varphi : \tau_1 \to \cdots \to \tau_{\varrho(\varphi)} \to \mathrm{o}$ in $V \cup N$ and $m \in [0, \varrho(\varphi)]$. Let $\ell \in [1, \varrho(\varphi)]$ and let k be the order of τ_ℓ. Assume that $\mathrm{Arg}_\ell(s) = r :: t$.

If $\ell \leqslant m$, then $\mathrm{Arg}_\ell(s) = \mathrm{pop}_1 s :: t_\ell$. In particular, r is equal to $\mathrm{pop}_1(s)$ and the property holds because $\mathrm{pop}_{n-k+1}(r) = \mathrm{pop}_{n-k+1}(\mathrm{pop}_1(s)) = \mathrm{pop}_{n-k+1}(s)$ as $n - k + 1 \geqslant 2$ (indeed $k < n$ by definition of n).

If $\ell > m$, $\mathrm{Arg}_\ell(s) = \mathrm{Arg}_{\ell-m}(\mathrm{collapse}(s))$. By the induction hypothesis, we have

$$\mathrm{pop}_{n-k+1}(\mathrm{collapse}(s)) = \mathrm{pop}_{n-k+1}(r).$$

To conclude the result, it is enough to show that $\mathrm{pop}_{n-k+1}(\mathrm{collapse}(s)) = \mathrm{pop}_{n-k+1}(s)$. Let k' be the order of $top_1(s)$. As $top_1(s) = \varphi\, t_1 \cdots t_m$ is of type $\tau_{m+1} \to \cdots \to \tau_{\varrho(\varphi)} \to \mathrm{o}$, we have $k' > k$. By definition of well-formed stacks, the order of the link attached to top symbol is equal to $n - k' + 1$. In particular, $\mathrm{pop}_{n-k+1}(\mathrm{collapse}(s)) = \mathrm{pop}_{n-k+1}(s)$. □

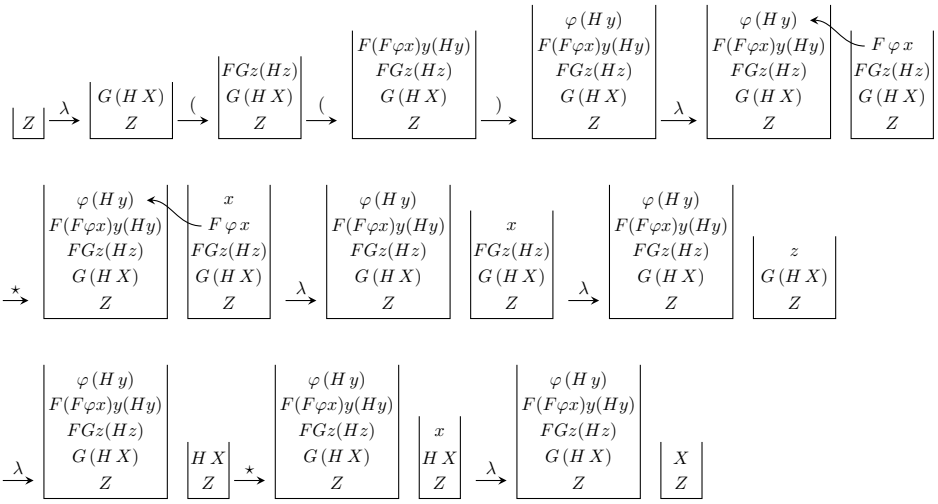
4.2 Simulating the LTS of $\mathcal{S}$ on stacks

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., $\mathrm{Tree}^\bot(\mathcal{M}) = \mathrm{Tree}^\bot(\mathcal{S})$). From $\mathcal{M}$, a CPDA generating $\mathrm{Tree}^\bot(\mathcal{M})$ is then easily defined.

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

$$\begin{cases} s \xrightarrow[\mathcal{M}]{a} \mathrm{push}_1^t(s) & \text{if } s \text{ is an } A\text{-stack with } A \in N \\ & \text{and } A\,x_1 \cdots x_{\varrho(A)} \xrightarrow{a} t \in \mathcal{R}, \\ s \xrightarrow[\mathcal{M}]{\lambda} \mathrm{push}_1^t(r) & \text{if } s \text{ is a } \varphi\text{-stack with } \varphi : \mathrm{o} \in V \\ & \text{and } \mathrm{Arg}_{\mathrm{rk}(\varphi)}(\mathrm{pop}_1(s)) = r :: t, \\ s \xrightarrow[\mathcal{M}]{\lambda} \mathrm{push}_1^{t,n-k+1}(r) & \text{if } s \text{ is a } \varphi\text{-stack with } \varphi : \tau \in V \text{ of order } k > 0 \\ & \text{and } \mathrm{Arg}_{\mathrm{rk}(\varphi)}(\mathrm{pop}_1(\mathrm{push}_{n-k+1}(s))) = r :: t. \end{cases}$$

Example 4.5. 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 $[\![\cdot]\!]$ (cf. Lemma 4.4 below). 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 normalization rules that aim at replacing the variable at the head of the top of the stack (in Example 4.5 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.

The following lemma states the soundness of the first line of the definition of $\xrightarrow[\mathcal{M}]{}$.

Lemma 4.4. *Let s be an N-stack in* WStacks *and $a \in \Sigma$.*

$$\begin{cases} \exists t \in \mathrm{Terms}(N),\ [\![s]\!] \xrightarrow{a} t & \Rightarrow \quad \exists s' \in \mathrm{WStacks},\ s \xrightarrow[\mathcal{M}]{a} s' \text{ and } [\![s']\!] = t \\ \exists s' \in \mathrm{WStacks},\ s \xrightarrow[\mathcal{M}]{a} s' & \Rightarrow \quad [\![s]\!] \xrightarrow{a} [\![s']\!] \end{cases}$$

Proof. Let $s \in \mathrm{WStacks}$ be an A-stack for some $A \in N$ and let $a \in \Sigma$. By definition of $[\![s]\!]$, $[\![s]\!]$ is equal to $A\,[\![\mathrm{Arg}_1(s)]\!] \cdots [\![\mathrm{Arg}_{\varrho(A)}(s)]\!]$.

Assume that $[\![s]\!] \xrightarrow{a} t$ for some $t \in \mathrm{Terms}(N)$. By definition of $\xrightarrow{a}$, there exists a production $A\,x_1 \cdots x_{\varrho(A)} \xrightarrow{a} t'$ in $\mathcal{R}$ such that t is equal to

$$t'[x_1/[\![\mathrm{Arg}_1(s)]\!], \ldots, x_{\varrho(A)}/[\![\mathrm{Arg}_{\varrho(A)}(s)]\!]].$$

By definition of $\xrightarrow[\mathcal{M}]{a}$, we have $s \xrightarrow[\mathcal{M}]{a} \mathrm{push}_1^{t'}(s)$ hence we only need to note that the term $[\![\mathrm{push}_1^{t'}(s)]\!]$ is equal to $t'[x_1/[\![\mathrm{Arg}_1(s)]\!], \ldots, x_{\varrho(A)}/[\![\mathrm{Arg}_{\varrho(A)}(s)]\!]]$. Indeed, as t' is of ground type, $[\![\mathrm{push}_1^{t'}(s)]\!]$ is equal to $[\![s :: t']\!]$ which is by definition equal to $t'[x_1/[\![\mathrm{Arg}_1(s)]\!], \ldots, x_{\varrho(A)}/[\![\mathrm{Arg}_{\varrho(A)}(s)]\!]]$.

Now, assume that $s \xrightarrow[\mathcal{M}]{a} s'$ for some $s' \in \mathrm{WStacks}$. By definition of $\xrightarrow[\mathcal{M}]{a}$, there exists a production $A\,x_1 \cdots x_{\varrho(A)} \xrightarrow{a} t' \in \mathcal{R}$ such that $s' = \mathrm{push}_1^{t'}(s)$. As s is an A-stack, we have $[\![s]\!] = A\,[\![\mathrm{Arg}_1(s)]\!] \cdots [\![\mathrm{Arg}_{\varrho(A)}(s)]\!]$. Furthermore $[\![s']\!]$ is equal to $t'[x_1/\mathrm{Arg}_1(s), \ldots, x_{\varrho(A)}/\mathrm{Arg}_{\varrho(A)}(s)]$. Hence by definition of $\xrightarrow{a}$, $[\![s]\!] \xrightarrow{a} [\![s']\!]$. □

The next lemma states the soundness of the second and third lines of the definition of $\mathcal{M}$. It also permits concluding that there are no infinite paths labeled by λ in $\mathcal{M}$.

Lemma 4.5. *We have the following properties:*

(1) Let $s \in \mathrm{WStacks}$ be a φ-stack with $\varphi \in V$ and $s' \in \mathrm{WStacks}$ be a ψ-stack with $\psi \in V \cup N$. If $s \xrightarrow[\mathcal{M}]{\lambda} s'$ then $\mathrm{ord}(\varphi) \leqslant \mathrm{ord}(\psi)$ and $[\![s]\!] = [\![s']\!]$ with $|\,\mathrm{top}_{n-\mathrm{ord}(\varphi)+1}(s)\,| > |\,\mathrm{top}_{n-\mathrm{ord}(\varphi)+1}(s')\,|$.

(2) For all stack $s \in \mathrm{WStacks}$ there exists a unique N-stack $s' \in \mathrm{WStacks}$ such that $s \xrightarrow[\mathcal{M}]{\lambda^*} s'$.

Proof. (1) Let φ be a variable in V and let s be a φ-stack in WStacks.We distinguish two cases depending on the order of the φ.

Assume that φ is of ground type and that $\mathrm{Arg}_{\mathrm{rk}(\varphi)}(\mathrm{pop}_1(s))$ is some $r :: t \in \mathrm{CStacks}$.

We have by definition of $\mathcal{M}$ that $s \xrightarrow[\mathcal{M}]{\lambda} s' = \mathrm{push}_1^t(r)$. To show that $[\![s]\!]$ is equal to $[\![s']\!]$, we simply unfold the definitions.

$$[\![s]\!] \overset{\mathrm{def}}{=} [\![\mathrm{pop}_1(s) :: \varphi]\!] \overset{\mathrm{def}}{=} [\![\mathrm{Arg}_{\mathrm{rk}(\varphi)}(\mathrm{pop}_1(s))]\!] \overset{\mathrm{def}}{=} [\![r :: t]\!] \overset{\mathrm{Def}\ 4.5}{=} [\![\mathrm{push}_1^t(r)]\!] \overset{\mathrm{def}}{=} [\![s']\!].$$

Assume that $s' = \mathrm{push}_1^t(r)$ is a ψ-stack for some $\psi \in N \cup V$. We have $\mathrm{ord}(\psi) \geqslant \mathrm{ord}(\varphi) = 0$. As $|\,\mathrm{Arg}_k(\mathrm{pop}_1(s))\,| \leqslant |\,s\,| - 2$, we have that $|\,\mathrm{top}_{n+1}(s) = s\,| > |\,\mathrm{top}_{n+1}(s') = s'\,|$.

Assume that φ is of type $\tau = \tau_1 \to \cdots \to \tau_{\varrho(\varphi)} \to \mathrm{o}$ of order $k > 0$. Assume that $\mathrm{Arg}_{\mathrm{rk}(\varphi)}(\mathrm{pop}_1(\mathrm{push}_{n-k+1}(s)))$ is equal to $r :: t \in \mathrm{CStacks}$. First recall that, from Lemma 4.1, we have that $t : \tau$. We have by definition that $s \xrightarrow[\mathcal{M}]{} s' = \mathrm{push}_1^{t,n-k+1}(r)$. Let us show that $[\![s]\!] = [\![s']\!]$. Using Fact 2, we have that:

$$\begin{aligned}\llbracket s' \rrbracket &= \underbrace{\llbracket \mathrm{pop}_1(s') :: top_1(s') \rrbracket}_{=\llbracket \mathrm{pop}_1(s)::\varphi \rrbracket \ (4.1)} \underbrace{\llbracket \mathrm{Arg}_1(\mathrm{collapse}(s')) \rrbracket}_{=\llbracket \mathrm{Arg}_1(s) \rrbracket \ (4.2)} \cdots \underbrace{\llbracket \mathrm{Arg}_{\varrho(\varphi)}(\mathrm{collapse}(s')) \rrbracket}_{=\llbracket \mathrm{Arg}_{\varrho(\varphi)}(s) \rrbracket \ (4.2)} \\ &= \llbracket \mathrm{pop}_1(s) :: \varphi \rrbracket \llbracket \mathrm{Arg}_1(s) \rrbracket \cdots \llbracket \mathrm{Arg}_{\varrho(\varphi)}(s) \rrbracket = \llbracket s \rrbracket.\end{aligned}$$

The equalities denoted (4.1) and (4.2) are proven below:

$$\begin{aligned}\llbracket \mathrm{pop}_1(s') :: top_1(s') \rrbracket &\stackrel{\text{def}}{=} \llbracket r :: t \rrbracket = \llbracket \mathrm{Arg}_{\mathrm{rk}(\varphi)}(\mathrm{pop}_1(\mathrm{push}_{n-k+1}(s))) \rrbracket \\ &\stackrel{\text{Lemma 4.2}}{=} \llbracket \mathrm{Arg}_{\mathrm{rk}(\varphi)}(\mathrm{pop}_1(s)) \rrbracket = \llbracket \mathrm{pop}_1(s) :: \varphi \rrbracket \quad (4.1)\end{aligned}$$

and for all $i \in [1, \varrho(\varphi)]$,

$$\begin{aligned}\llbracket \mathrm{Arg}_i(\mathrm{collapse}(s')) \rrbracket &= \llbracket \mathrm{Arg}_i(\mathrm{collapse}(\mathrm{push}_1^{t,n-k+1}(r))) \rrbracket \\ &= \llbracket \mathrm{Arg}_i(\mathrm{pop}_{n-k+1}(r)) \rrbracket \\ &\stackrel{\text{Lemma 4.3}}{=} \llbracket \mathrm{Arg}_i(\mathrm{pop}_{n-k+1}(\mathrm{pop}_1(\mathrm{push}_{n-k+1}(s)))) \rrbracket \\ &= \llbracket \mathrm{Arg}_i(s) \rrbracket.\end{aligned} \tag{4.2}$$

As both φ and t have type τ, and as t is of the form $\psi\ t_1 \cdots t_\ell$ for some $\ell \geqslant 0$, it directly follows that $\mathrm{ord}(\varphi) \leqslant \mathrm{ord}(\psi)$.

The inequality $|\,\mathrm{top}_{n-\mathrm{ord}(\varphi)+1}(s)\,| > |\,\mathrm{top}_{n-\mathrm{ord}(\varphi)+1}(s')\,|$ follows from Lemma 4.3.
(2) Assume, to get a contradiction, that there exists an infinite sequence $(s_i)_{i\geqslant 0}$ of stacks in WStacks such that for all $i \geqslant 0$, $s_i \xrightarrow[\mathcal{M}]{\lambda} s_{i+1}$. For all $i \geqslant 0$, we let t_i denote the top symbol of s_i and φ_i the head symbol of t_i. According to (1), the order of the φ_i increases and hence is ultimately constant. Let j and k be such that, for all $i \geqslant j$, $\mathrm{ord}(\varphi_i)$ is equal to k. Using (1), the size of the $\mathrm{top}_{n-k+1}(s_i)$ is strictly decreasing starting from j, which provides the contradiction. □

From Lemma 4.4 and 4.5, $\mathcal{M}$ and $\mathcal{S}$ generate the same trees.

Proposition 4.6.

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

Proof. By definition of $\mathcal{M}$, only well-formed N-stacks can be the source of non-silent transitions. Let s be a well-formed N-stack. If $\llbracket s \rrbracket \xrightarrow[\mathcal{S}]{a} t$ for some $a \in \Sigma$, then the N-stack s' such that $s \xrightarrow[\mathcal{M}]{a\lambda^*} s'$ is such that $\llbracket s' \rrbracket = t$. Conversely if $s \xrightarrow[\mathcal{M}]{a\lambda^*} s'$ for some N-stack s', then $\llbracket s \rrbracket \xrightarrow[\mathcal{S}]{a} \llbracket s' \rrbracket$. □

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 $\{\, q_0, q_1, \ldots, q_{\varrho(\mathcal{S})}, q_*, q_V \,\}$ 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 state q_V is used to signal stacks that appear in the derivation of system $\mathcal{M}$ that are V-stacks. The transitions are given below.

- $\delta(q_0, \bot, \lambda) = (q_*, \mathrm{push}_1^Z)$,
- If t starts with $F \in N$ and $F\, x_1 \cdots x_{\varrho(F)} \xrightarrow{a} e \in \mathcal{R}$:
  - $\delta(q_*, t, a) = (q_*, \mathrm{push}_1^e)$ if e starts with a symbol in N,
  - $\delta(q_*, t, a) = (q_a, \mathrm{push}_1^e)$ if e starts with a variable.
- If t is a term of the form $\varphi\, t_1 \cdots t_\ell$ for some $\varphi \in V$:
  - $\delta(q_V, t, \lambda) = (q_{\mathrm{rk}(\varphi)}, \mathrm{pop}_1)$ if φ is an order-0 variable,
  - $\delta(q_V, t, \lambda) = (q_{\mathrm{rk}(\varphi)}, \mathrm{push}_{n-k+1}; \mathrm{pop}_1)$ if φ is a variable of order $k > 0$.
- If t is a term of the form $\varphi\, t_1 \cdots t_\ell$ for some $\varphi \in V \cup N$:
  - $\delta(q_k, t, \lambda) = (q_{\mathrm{rk}(t_k)}, \mathrm{pop}_1; \mathrm{push}_1^{t_k})$ if $k \leqslant \ell$ and $t_k : o$,
  - $\delta(q_k, t, \lambda) = (q_{\mathrm{rk}(t_k)}, \mathrm{pop}_1; \mathrm{push}_1^{t_k, n-h+1})$ if $k \leqslant \ell$ and t_k has order $h > 0$,
  - $\delta(q_k, t, \lambda) = (q_{k-\ell}, \mathrm{collapse})$ 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.

Remark 4.7. The previously given CPDA uses several operations per transition. An equivalent CPDA using only one operation per transition may be obtained by adding intermediary states.

Theorem 4.8. *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 $\mathcal{S}$.*

Proof (sktech). Let s be a well-formed stack. We denote by $\langle\!\langle\, s \,\rangle\!\rangle$ the configuration of $\mathcal{A}$ defined by $\langle\!\langle\, s \,\rangle\!\rangle = (q_*, s)$ if s is an N-stack and $\langle\!\langle\, s \,\rangle\!\rangle = (q_{\mathrm{rk}(x)}, s)$ if s is a V-stack whose topmost symbol starts with a variable x.

Clearly for any well-formed N-stack s, $s \xrightarrow[\mathcal{M}]{a} s'$ if and only if $\langle\!\langle\, s \,\rangle\!\rangle \xrightarrow[\mathcal{A}]{a} \langle\!\langle\, s' \,\rangle\!\rangle$.

For any V-stack s, if $s \xrightarrow[\mathcal{M}]{\lambda} s'$ then $\langle\!\langle\, s \,\rangle\!\rangle \xrightarrow[\mathcal{A}]{\lambda^*} \langle\!\langle\, s' \,\rangle\!\rangle$ as intuitively $\xrightarrow[\mathcal{A}]{}$ combines the definition of both $\xrightarrow[\mathcal{M}]{}$ and $\mathrm{Arg}_k(\,\cdot\,)$. Conversely, for all V-stacks, if $s \xrightarrow[\mathcal{M}]{\lambda} s'$ and $\langle\!\langle\, s \,\rangle\!\rangle \xrightarrow[\mathcal{A}]{\lambda} \langle\!\langle\, s_2 \,\rangle\!\rangle$ then $\langle\!\langle\, s_2 \,\rangle\!\rangle \xrightarrow[\mathcal{A}]{\lambda^*} \langle\!\langle\, s' \,\rangle\!\rangle$. □

5 Safe higher-order recursion schemes

In this last section, we consider a syntactic subfamily of recursion schemes called *safe recursion schemes*. The *safety* constraint was introduced in [36], but was already implicit in the work of Damm [22] (also see [24, 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 5.1 ([36]). A recursion scheme is *safe* if none of its right-hand sides contains an argument-subterm of order k containing a variable of order strictly less than k.

Other than the scheme $\mathcal{S}_U$ generating the tree of the Urzyczyn language, all examples we gave are safe schemes. The scheme $\mathcal{S}_U$ is not safe, as the production

$$F\,\varphi\,x \xrightarrow{(} F(F\varphi x)y(Hy)$$

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 [49, 50].

Remark 5.1. In [36], 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. In Proposition 5.5, we will see that dropping the homogeneity constraint in the definition of safety does not change the family of generated trees.

5.1 Safety and the Translation from Schemes to CPDA

In [36, 37], 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. Recall that an order-k pushdown automaton is an order-k CPDA that does not use the collapse operation (hence, links are useless).

Theorem 5.2 below shows that the translation of recursion schemes into collapsible automata presented in Section 4, 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. This notion is captured by the notion of link-free CPDA.

Definition 5.2. A CPDA is *link-free* if for every configuration (p, s) reachable from the initial configuration and for every transition $\delta(p, top_1(s), a) = (q, \mathrm{collapse})$, we have $\mathrm{collapse}(s) = \mathrm{pop}_\ell(s)$, where ℓ is the order of the link attached to $top_1(s)$.

Theorem 5.2. *The translation of Section 4 applied to a safe recursion scheme yields a link-free collapsible automaton.*

We get the following corollary extending a previous result from [36], by dropping the homogeneity assumption.

Corollary 5.3. *Order-k safe schemes and order-k pushdown automata generate the same trees.*

5.2 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 to the same family of trees [22].

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 $f : \mathrm{o} \to \mathrm{o}$, $c : \mathrm{o}$ and $\varphi : (\mathrm{o} \to \mathrm{o}) \to (\mathrm{o} \to \mathrm{o}) \to \mathrm{o} \to \mathrm{o} \to \mathrm{o}$, 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 5.3 ([22]). 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 $\varphi : \tau_1 \to \cdots \to \tau_n \to \mathrm{o}$ and either $k \in \{0, n\}$ or $\mathrm{ord}(\tau_k) > \mathrm{ord}(\tau_{k+1})$.

Remark 5.4. The second constraint in the definition of Damm safety can be reformulated as follows: all argument subterms of an argument subterm of order-k appearing in a right-hand side have at least order-k.

Using Remark 5.4, 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 $x : \mathrm{o}$ and non-terminals $G : \mathrm{o} \to \mathrm{o} \to \mathrm{o}$ and $C : \mathrm{o}$. Then $G\, x$ cannot appear as an argument-subterm in a safe scheme, but $G\, C$ can. As $G\, C$ does not satisfy the 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 of converting the safe scheme into a higher-order pushdown automaton (Corollary 5.3) and then converting this automaton back to a scheme using the translation of [36]. In fact, this translation of higher-order pushdown automata into safe schemes produces Damm-safe schemes.

Proposition 5.5. *Damm-safe schemes are safe and for every safe scheme, there exists a Damm-safe scheme of the same order generating the same tree.*

This proposition in particular shows that any safe scheme can be transformed into an equivalent homogeneous one. Broadbent, using the translation from schemes into CPDA, showed that any scheme (possibly unsafe) can be converted into an equivalent one that is homogeneous [7]. Recently, Parys gave a new proof of this result by directly manipulating the scheme; he also provided another construction that preserves safety [51].

References

[1] K. Aehlig. A finite semantics of simply-typed lambda terms for infinite runs of automata. In Z. Ésik, editor, *Proc. 20th Workshop on Computer Science Logic*, volume 4207 of *Lecture Notes in Comput. Sci.*, pages 104–118. Springer-Verlag, 2006. 443

[2] K. Aehlig, J. de Miranda, and L. Ong. Safety is not a restriction at level 2 for string languages. In V. Sassone, editor, *Proc. 8th Int. Conf. on Foundations of Software Science and Computation Structures*, volume 3411 of *Lecture Notes in Comput. Sci.*, pages 490–501. Springer-Verlag, 2005. 442, 443

[3] A. Arnold and D. Niwiński. *Rudiments of mu-calculus*, volume 146 of *Studies in Logic and the Foundations of Mathematics*. Elsevier, 2001. 442

[4] T. Ball and S. K. Rajamani. The SLAM project: Debugging system software via static analysis. In J. Launchbury and J. C. Mitchell, editors, *Proc. 29th ACM Symp. on Principles of Programming Languages*, pages 1–3. ACM, 2002. 443

[5] V. Bárány, E. Grädel, and S. Rubin. Automata-based presentations of infinite structures. In *Finite and Algorithmic Model Theory*, volume 379 of *London Math. Soc. Lect. Note Ser.*, pages 1–76. Cambridge University Press, 2011. 442

[6] H. P. Barendregt. *The lambda Calculus: its syntax and semantics*, volume 103. North Holland, revised edition, 1984. 450

[7] C. Broadbent. *On collapsible pushdown automata, their graphs and the power of links*. PhD thesis, University of Oxford, 2011. 477

[8] C. Broadbent, A. Carayol, C.-H. L. Ong, and O. Serre. Recursion schemes and logical reflexion. In *Proc. 25th IEEE Symp. on Logic in Computer Science*, pages 120–129. IEEE Computer Society, 2010. 443

[9] C. H. Broadbent, A. Carayol, M. Hague, and O. Serre. C-SHORe: a collapsible approach to higher-order verification. In G. Morrisett and T. Uustalu, editors, *Proc. 18th ACM SIGPLAN Int. Conf. on Functional Programming*, pages 13–24. ACM, 2013. 444

[10] C. H. Broadbent and N. Kobayashi. Saturation-based model checking of higher-order recursion schemes. In S. R. D. Rocca, editor, *Proc. 22nd Annual Conf. of the European Association for Computer Science Logic*, volume 23 of *LIPIcs*, pages 129–148. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2013. 444

[11] T. Cachat. Higher order pushdown automata, the Caucal hierarchy of graphs and parity games. In J. C. M. Baeten, J. K. Lenstra, J. Parrow, and G. J. Woeginger, editors, *Proc. 30th Int. Conf. on Automata, Languages, and Programming (ICALP)*, volume 2719 of *Lecture Notes in Comput. Sci.*, pages 556–569. Springer-Verlag, 2003. 443

[12] A. Carayol, A. Meyer, M. Hague, C.-H. L. Ong, and O. Serre. Winning regions of higher-order pushdown games. In *Proc. 23rd IEEE Symp. on Logic in Computer Science*, pages 193–204. IEEE Computer Society, 2008. 443

[13] A. Carayol and O. Serre. Collapsible pushdown automata and labeled recursion schemes: Equivalence, safety and effective selection. In *Proc. 27th IEEE Symp. on Logic in Computer Science*, pages 165–174. IEEE Computer Society, 2012. 443

[14] A. Carayol and S. Wöhrle. The Caucal hierarchy of infinite graphs in terms of logic and higher-order pushdown automata. In P. K. Pandya and J. Radhakrishnan, editors, *Proc. 23rd Conf. on Foundations of Software Technology and Theoretical Computer Science*, volume 2914 of *Lecture Notes in Comput. Sci.*, pages 112–123. Springer-Verlag, 2003. 442

[15] D. Caucal. On infinite terms having a decidable monadic theory. In K. Diks and W. Rytter, editors, *Proc. 27th Symp., Mathematical Foundations of Computer Science 2002*, volume 2420 of *Lecture Notes in Comput. Sci.*, pages 165–176. Springer-Verlag, 2002. 442, 443

[16] B. Courcelle. A representation of trees by languages I. *Theoret. Comput. Sci.*, 6:255–279, 1978. 441, 453

[17] B. Courcelle. A representation of trees by languages II. *Theoret. Comput. Sci.*, 7:25–55, 1978. 441, 453

[18] B. Courcelle. The monadic second-order logic of graphs IX: machines and their behaviours. *Theoret. Comput. Sci.*, 151:125–162, 1995. 442

[19] B. Courcelle and M. Nivat. The algebraic semantics of recursive program schemes. In J. Winkowski, editor, *Proc. 7th Symp., Mathematical Foundations of Computer Science 1978*, volume 64 of *Lecture Notes in Comput. Sci.*, pages 16–30. Springer-Verlag, 1978. 441

[20] W. Damm. Higher type program schemes and their tree languages. In H. Tzschach, H. Waldschmidt, and H. K. Walter, editors, *Theoretical Computer Science, Proc. 3rd GI Conf.*, volume 48 of *Lecture Notes in Comput. Sci.*, pages 51–72. Springer-Verlag, 1977. 441

[21] W. Damm. Languages defined by higher type program schemes. In A. Salomaa and M. Steinby, editors, *Proc. 4th Colloq. on Automata, Languages, and Programming (ICALP)*, volume 52 of *Lecture Notes in Comput. Sci.*, pages 164–179. Springer-Verlag, 1977. 441

[22] W. Damm. The IO- and OI-hierarchies. *Theoret. Comput. Sci.*, 20:95–207, 1982. 441, 451, 475, 477

[23] W. Damm and A. Goerdt. An automata-theoretical characterization of the OI-hierarchy. *Inform. Comput.*, 71:1–32, 1986. 441

[24] J. de Miranda. *Structures generated by higher-order grammars and the safety constraint*. PhD thesis, University of Oxford, 2006. 440, 475

[25] J. Engelfriet. Iterated pushdown automata and complexity classes. In D. S. Johnson, R. Fagin, M. L. Fredman, D. Harel, R. M. Karp, N. A. Lynch, C. H. Papadimitriou, R. L. Rivest, W. L. Ruzzo, and J. I. Seiferas, editors, *Proc. 15th Ann. ACM Symp. Theor. Comput., STOC 1983*. ACM, 1983. 441

[26] J. Engelfriet. Iterated stack automata and complexity classes. *Inform. Comput.*, 95(1):21–75, 1991. 441

[27] J. Engelfriet and E. M. Schmidt. IO and OI. I. *J. Comput. System Sci.*, 15(3):328–353, 1977. 441

[28] J. Engelfriet and E. M. Schmidt. IO and OI. II. *J. Comput. System Sci.*, 16(1):67–99, 1978. 441

[29] J. Flum, E. Grädel, and T. Wilke. *Logic and automata: history and perspectives*. Amsterdam University Press, 2007. 442

[30] S. Garland and D. Luckham. Program schemes, recursion schemes and formal languages. *J. Comput. System Sci.*, 7(2):119–160, 1973. 440

[31] E. Grädel, W. Thomas, and T. Wilke, editors. *Automata, Logics, and Infinite Games: A Guide to Current Research [outcome of a Dagstuhl seminar, February 2001]*, volume 2500 of *Lecture Notes in Comput. Sci.* Springer-Verlag, 2002. 442

[32] M. Hague, A. S. Murawski, C.-H. L. Ong, and O. Serre. Collapsible pushdown automata and recursion schemes. In *Proc. 23rd IEEE Symp. on Logic in Computer Science*, pages 452–461. IEEE Computer Society, 2008. 442, 444, 445

[33] J. M. E. Hyland and C.-H. L. Ong. On Full Abstraction for PCF: I. Models, observables and the full abstraction problem, II. Dialogue games and innocent strategies, III. A fully abstract and universal game model. *Inform. Comput.*, 163:285–408, 2000. 443

[34] K. Inaba and S. Maneth. The complexity of tree transducer output languages. In R. Hariharan, M. Mukund, and V. Vinay, editors, *Proc. 28th Conf. on Foundations of Software Technology and Theoretical Computer Science*, volume 2 of *LIPIcs*, pages 244–255. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2008. 441

[35] K. Indermark. Schemes with recursion on higher types. In A. W. Mazurkiewicz, editor, *Proc. 5th Symp., Mathematical Foundations of Computer Science 1976*, volume 45 of *Lecture Notes in Comput. Sci.*, pages 352–358. Springer-Verlag, 1976. 441

[36] T. Knapik, D. Niwiński, and P. Urzyczyn. Deciding monadic theories of hyperalgebraic trees. In S. Abramsky, editor, *5th Int. Conf. on Typed Lambda Calculi and Applications, TLCA'01*, volume 2044 of *Lecture Notes in Comput. Sci.*, pages 253–267. Springer-Verlag, 2001. 442, 475, 476, 477

[37] T. Knapik, D. Niwiński, and P. Urzyczyn. Higher-order pushdown trees are easy. In M. Nielsen and U. Engberg, editors, *Proc. 5th Int. Conf. on Foundations of Software Science and Computation Structures*, volume 2303 of *Lecture Notes in Comput. Sci.*, pages 205–222. Springer-Verlag, 2002. 442, 443, 476

[38] T. Knapik, D. Niwiński, P. Urzyczyn, and I. Walukiewicz. Unsafe grammars and panic automata. In L. Caires, G. F. Italiano, L. Monteiro, C. Palamidessi, and M. Yung, editors, *Proc. 32nd Int. Conf. on Automata, Languages, and Programming (ICALP)*, volume 3580 of *Lecture Notes in Comput. Sci.*, pages 1450–1461. Springer-Verlag, 2005. 442, 443

[39] N. Kobayashi. Model-checking higher-order functions. In A. Porto and F. J. López-Fraguas, editors, *Proc. 11th Int. ACM SIGPLAN Conf. on Principles and Practice of Declarative Programming*, pages 25–36. ACM, 2009. 444

[40] N. Kobayashi. Types and higher-order recursion schemes for verification of higher-order programs. In Z. Shao and B. C. Pierce, editors, *Proc. 36th ACM Symp. on Principles of Programming Languages*, pages 416–428. ACM, 2009. 443

[41] N. Kobayashi. A practical linear time algorithm for trivial automata model checking of higher-order recursion schemes. In M. Hofmann, editor, *Proc. 14th Int. Conf. on Foundations of Software Science and Computation Structures*, volume 6604 of *Lecture Notes in Comput. Sci.*, pages 260–274. Springer-Verlag, 2011. 444

[42] N. Kobayashi. Model checking higher-order programs. *J. Assoc. Comput. Mach.*, 60(3):20:1–20:62, 2013. 444

[43] 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. 24th IEEE Symp. on Logic in Computer Science*, pages 179–188. IEEE Computer Society, 2009. 443

[44] A. N. Maslov. The hierarchy of indexed languages of an arbitrary level. *Sov. math. Dokl.*, 15:1170–1174, 1974. 441

[45] A. N. Maslov. Multilevel stack automata. *Probl. Inf. Transm.*, 12:38–43, 1976. 441, 456

[46] R. P. Neatherway, S. J. Ramsay, and C.-H. L. Ong. A traversal-based algorithm for higher-order model checking. In P. Thiemann and R. B. Findler, editors, *Proc. 17th ACM SIGPLAN Int. Conf. on Functional Programming*, pages 353–364. ACM, 2012. 444

[47] M. Nivat. Langages algébriques sur le magma libre et sémantique des schémas de programme. In M. Nivat, editor, *Automata, Languages, and Programming: Proceedings of a Symposium*, pages 293–308. North-Holland, 1972. 440

[48] C.-H. L. Ong. On model-checking trees generated by higher-order recursion schemes. In *Proc. 21st IEEE Symp. on Logic in Computer Science*, pages 81–90. IEEE Computer Society, 2006. 443

[49] P. Parys. Collapse operation increases expressive power of deterministic higher order pushdown automata. In T. Schwentick and C. Dürr, editors, *STACS 2011, Proc. 28th Symp. Theoretical Aspects of Comp. Sci.*, volume 9 of *LIPIcs*, pages 603–614. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2011. 455, 476

[50] P. Parys. On the significance of the collapse operation. In *Proc. 27th IEEE Symp. on Logic in Computer Science*, pages 521–530. IEEE Computer Society, 2012. 476

[51] P. Parys. Homogeneity without loss of generality. In H. Kirchner, editor, *Proc. 21st Int. Conf. on Foundations of Software Science and Computation Structures*, volume 108 of *LIPIcs*, pages 27:1–27:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. 477

[52] P. Parys. Recursion schemes and the WMSO+U logic. In R. Niedermeier and B. Vallée, editors, *STACS 2018, Proc. 35th Symp. Theoretical Aspects of Comp. Sci.*, volume 96 of *LIPIcs*, pages 53:1–53:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. 443

[53] M. Rabin. Decidability of second-order theories and automata on infinite trees. *Trans. Amer. Math. Soc.*, 141:1–35, 1969. 442

[54] S. J. Ramsay, R. P. Neatherway, and C. L. Ong. A type-directed abstraction refinement approach to higher-order model checking. In S. Jagannathan and P. Sewell, editors, *Proc. 41st ACM Symp. on Principles of Programming Languages*, pages 61–72. ACM, 2014. 444

[55] S. Salvati and I. Walukiewicz. Krivine machines and higher-order schemes. In L. Aceto, M. Henzinger, and J. Sgall, editors, *Proc. 38th Int. Conf. on Automata, Languages, and Programming (ICALP)*, volume 6756 of *Lecture Notes in Comput. Sci.*, pages 162–173. Springer-Verlag, 2011. 443

[56] S. Salvati and I. Walukiewicz. Recursive schemes, Krivine machines, and collapsible pushdown automata. In A. Finkel, J. Leroux, and I. Potapov, editors, *Reachability Problems, Proc. 6th Int. Workshop, RP 2012*, volume 7550 of *Lecture Notes in Comput. Sci.*, pages 6–20. Springer-Verlag, 2012. 443

[57] S. Schwoon. *Model-checking pushdown systems*. PhD thesis, Technical University of Munich, 2002. 443, 444

[58] G. Sénizergues. The equivalence problem for deterministic pushdown automata is decidable. In P. Degano, R. Gorrieri, and A. Marchetti-Spaccamela, editors, *Proc. 24th Int. Conf. on Automata, Languages, and Programming (ICALP)*, volume 1256 of *Lecture Notes in Comput. Sci.*, pages 671–681. Springer-Verlag, 1997. 441

[59] G. Sénizergues. L(A)=L(B)? a simplified decidability proof. *Theoret. Comput. Sci.*, 281(1-2):555–608, 2002. 441

[60] C. Stirling. Decidability of bisimulation equivalence for pushdown processes. Technical Report EDI-INF-RR-0005, School of Informatics, University of Edinburgh, 2000. 441, 446

[61] C. Stirling. Schema revisited. In P. Clote and H. Schwichtenberg, editors, *Proc. 14th Workshop on Computer Science Logic*, volume 1862 of *Lecture Notes in Comput. Sci.*, pages 126–138. Springer-Verlag, 2000. 441

[62] C. Stirling. Decidability of DPDA equivalence. *Theoret. Comput. Sci.*, 255(1-2):1–31, 2001. 441

[63] W. Thomas. Languages, automata, and logic. In G. Rozenberg and A. Salomaa, editors, *Handbook of formal language theory*, volume III, pages 389–455. Springer-Verlag, 1997. 442

[64] I. Walukiewicz. Pushdown processes: games and model-checking. *Inform. Comput.*, 157:234–263, 2001. 443