SCISPACE
formerly Typeset

Open access • Journal Article • DOI:10.1145/3452917

Higher-order Recursion Schemes and Collapsible Pushdown Automata: Logical Properties — Source link

Christopher H. Broadbent, Arnaud Carayol, C.-H. Luke Ong, Olivier Serre

Institutions: University of Oxford, Centre national de la recherche scientifique, University of Paris

Published on: 15 May 2021 - ACM Transactions on Computational Logic (ACMPUB27New York, NY)

Topics: Pushdown automaton and Recursion

Related papers:

- Higher-Order Recursion Schemes and Collapsible Pushdown Automata: Logical Properties
- The inclusion problem for monadic recursion schemes
- Automata for monadic second-order model-checking
- A contribution to formal methods : games, logic and automata
- Monoidal-closed categories of tree automata

Share this paper:

View more about this paper here: https://typeset.io/papers/higher-order-recursion-schemes-and-collapsible-pushdown-45pll4s1nj

HAL
open science

Higher-Order Recursion Schemes and Collapsible Pushdown Automata: Logical Properties

Christopher H. Broadbent, Arnaud Carayol, C. -H. Luke Ong, Olivier Serre

To cite this version:

Christopher H. Broadbent, Arnaud Carayol, C. -H. Luke Ong, Olivier Serre. Higher-Order Recursion Schemes and Collapsible Pushdown Automata: Logical Properties. ACM Transactions on Computational Logic, Association for Computing Machinery, 2021, 22 (2), 10.1145/3452917 . hal-03014425

HAL Id: hal-03014425

https://hal.archives-ouvertes.fr/hal-03014425

Submitted on 17 May 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Higher-Order Recursion Schemes and Collapsible Pushdown Automata: Logical Properties

CHRISTOPHER H. BROADBENT, Department of Computer Science, University of Oxford, UK
ARNAUD CARAYOL, LIGM, Univ Gustave Eiffel, CNRS, France
C.-H. LUKE ONG, Department of Computer Science, University of Oxford, UK
OLIVIER SERRE, Université de Paris, IRIF, CNRS, France

This paper studies the logical properties of a very general class of infinite ranked trees, namely those generated by higher-order recursion schemes. We consider, for both monadic second-order logic and modal μ-calculus, three main problems: model-checking, logical reflection (aka global model-checking, that asks for a finite description of the set of elements for which a formula holds) and selection (that asks, if exists, for some finite description of a set of elements for which an MSO formula with a second-order free variable holds). For each of these problems we provide an effective solution. This is obtained thanks to a known connection between higher-order recursion schemes and collapsible pushdown automata and on previous work regarding parity games played on transition graphs of collapsible pushdown automata.

CCS Concepts: • **Theory of computation** → **Automata over infinite objects**; **Grammars and context-free languages**; **Automata extensions**; **Verification by model checking**; **Program schemes**.

Additional Key Words and Phrases: Higher-Order Recursion Schemes, Higher-Order (Collapsible) Pushdown Automata, Monadic Second-Order Logic, Modal μ-calculus, Model-Checking, Reflection, Selection, Two-Player Perfect Information Parity Games

ACM Reference Format:
Christopher H. Broadbent, Arnaud Carayol, C.-H. Luke Ong, and Olivier Serre. 2021. Higher-Order Recursion Schemes and Collapsible Pushdown Automata: Logical Properties. 1, 1 (April 2021), 36 pages. https://doi.org/10.1145/nnnnn

1 INTRODUCTION

In this paper we study the logical properties of a very general class of infinite ranked trees, namely those generated by higher-order recursion schemes (equivalently by collapsible pushdown automata). We consider three main problems — model-checking, logical refection (aka global model-checking) and selection — for both monadic second-order logic and modal μ-calculus.

Authors' addresses: Christopher H. Broadbent, Department of Computer Science, University of Oxford, Wolfson Building, Parks Road, Oxford, OX1 3QD, UK, chbroadbent@gmail.com; Arnaud Carayol, LIGM, Univ Gustave Eiffel, CNRS, 5 boulevard Descartes — Champs sur Marne, Marne-la-Vallée Cedex 2, 77454, France, Arnaud.Carayol@univ-mlv.fr; C.-H. Luke Ong, Department of Computer Science, University of Oxford, Wolfson Building, Parks Road, Oxford, OX1 3QD, UK, Luke.Ong@cs.ox.ac.uk; Olivier Serre, Université de Paris, IRIF, CNRS, Bâtiment Sophie Germain, Case courrier 7014, 8 Place Aurélie Nemours, Paris Cedex 13, 75205, France, Olivier.Serre@cnrs.fr.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2021 Association for Computing Machinery.
XXXX-XXXX/2021/4-ART $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

Infinite Trees with a Decidable MSO Theory

A fundamental result of Rabin states that, for any formula expressible in monadic second-order (MSO) logic, one can decide whether it holds in the infinite complete binary tree [42]: in other words, the MSO model-checking problem is decidable. Since then, extending this result has been an important field of research. There are three main possible directions for that: the first one is to enrich MSO logic while preserving decidability; the second one is to look for structures richer than the infinite complete binary tree with a decidable MSO theory; the third one is to consider questions subsuming the model-checking problem. In this paper we are following the second and third directions.

Recursion schemes, an old model of computation, were originally designed as a canonical programming calculus for studying program transformation and control structures. In recent years, *higher-order recursion schemes* have received much attention as a method of constructing rich and robust classes of possibly infinite ranked trees with strong algorithmic properties; these are essentially finite typed deterministic term rewriting systems that generate when one applies the rewriting rules *ad infinitum* an infinite tree. The interest was sparked by the discovery of Knapik, Niwiński and Urzyczyn [34] that recursion schemes which satisfy a syntactic constraint called *safety* generate the same class of trees as higher-order pushdown automata. Remarkably these trees have decidable monadic second-order theories, subsuming earlier well-known MSO decidability results for regular (or order-0) trees [42] and algebraic (or order-1) trees [18].

An alternative approach was developed by Caucal who introduced in [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, and as both unfolding and MSO-interpretation preserve MSO decidability, it follows that structures in those hierarchies have MSO decidable theories.

A major step was obtained by Ong who proved in [38] that the modal μ-calculus (local) model checking problem for trees generated by *arbitrary* order-n recursion schemes is n-ExpTime-complete (hence these trees have decidable MSO theories). Note that this result was obtained using tools from innocent game semantics (in the sense of Hyland and Ong [31]) and in particular does not rely on an equivalent automata model for generating trees.

Finding a class of automata that characterises the expressivity of higher-order recursion schemes was left open. Indeed, the results of Damm and Goerdt [19], as well as those of Knapik *et al.* [33, 34] 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* [35]. Finally, Hague, Murawski, Ong and Serre gave a complete answer to the question in [29, 30] (also see [13]). They introduced a new kind of higher-order pushdown automata, which generalises *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 $n \geq 0$, order-n recursion schemes and order-n CPDA are equi-expressive as generators of trees.

Main Results

The equi-expressivity of higher-order recursion schemes and collapsible pushdown automata, as well as the connection between logic and two-player perfect-information parity games (see *e.g.* [45–47]), provide a roadmap to study logical properties of trees generated by recursion schemes: study collapsible pushdown games (*i.e.* parity games played on transition graphs of CPDA) and derive logical consequences on trees generated by recursion schemes. The companion paper [5] gives an in-depth study of collapsible pushdown parity games (following a series of papers [6, 12, 29] by the authors) on top of which we build in the present paper.

Our first straightforward contribution is to note that the decidability of the model-checking problem for MSO (equivalently μ-calculus) against trees generated by recursion schemes is an immediate consequence of the decidability of collapsible pushdown parity games and the equi-expressivity theorem

We then turn to the global version of the model-checking problem. Let $\mathcal{T}$ be a class of finitely-presentable infinite structures (such as trees or graphs) and $\mathcal{L}$ be a logical language for describing correctness properties of these structures. The *global model checking problem* asks, given $t \in \mathcal{T}$ and $\varphi \in \mathcal{L}$, whether the set $[[\varphi]]_t$ of nodes defined by φ and t is finitely describable, and if so, whether it is effective.

An innovation of our work is a new approach to global model checking, by "internalising" the semantics $[[\varphi]]_t$. Let $\varphi \in \mathcal{L}$, and $\mathcal{S}$ be a recursion scheme over a ranked alphabet Σ (i.e. the node labels of $[[\mathcal{S}]]$, the tree generated by $\mathcal{S}$, are elements of the ranked alphabet Σ). We say that $\mathcal{S}_\varphi$, which is a recursion scheme over $\Sigma \cup \underline{\Sigma}$ (where $\underline{\Sigma}$ consists of a marked copy of each Σ-symbol), is a *φ-reflection*[1] of $\mathcal{S}$ just if $\mathcal{S}$ and $\mathcal{S}_\varphi$ generate the same underlying tree; further, suppose a node u of $[[\mathcal{S}]]$ has label f, then the label of the node u of $[[\mathcal{S}_\varphi]]$ is $\underline{f}$ if u in $[[\mathcal{S}]]$ satisfies φ, and it is f otherwise. Equivalently we can think of $[[\mathcal{S}_\varphi]]$ as the tree that is obtained from $[[\mathcal{S}]]$ by distinguishing the nodes that satisfy φ. Our second contribution is the result that higher-order recursion schemes are (constructively) *reflective* with respect to the modal μ-calculus (Theorem 7.3). *I.e.* we give an algorithm that, given a modal μ-calculus formula φ, transforms a recursion scheme to its φ-reflection.

While modal μ-calculus and MSO are equivalent for expressing properties of a tree at its root, it is no longer true at other nodes (see *e.g.* [32]). Hence, it is natural to ask whether higher-order recursion schemes are reflective with respect to MSO logic. We answer positively (Theorem 7.6) this question by relying on the previous result for μ-calculus.

We derive two consequences of the MSO reflection. The first one (Corollary 8.1) is to show how MSO reflection can be used to construct, starting from a scheme that may have non-productive rules, an equivalent one that does not have such divergent computations. The second application consists in proving (Theorem 8.4) that the class of trees generated by recursion schemes is closed under the operation of MSO interpretation followed by tree unfolding hence, providing a result in the same flavour as the one obtained by Caucal for safe schemes in [15].

Our third main contribution is to consider a more general problem than (both local and global) model-checking, namely the *MSO selection property*. More precisely, we prove (Theorem 9.12) that if $\mathcal{S}$ is a recursion scheme generating a tree t satisfying a formula of the form $\exists X \varphi(X)$ (where X is a second-order free variable ranging over sets of nodes) then one can build another scheme that generates the tree t where a set of nodes U satisfying $\varphi(X)$ is marked. This result is in fact quite surprising as it is known from [9, 10, 25] that there exists a tree generated by an order-3

[1]In programming languages, *reflection* is the process by which a computer program can observe and dynamically modify its own structure and behaviour.

(safe) recursion scheme for which no MSO choice function exists, and that the selection property is closely connected to choice functions.

Note that most of the above mentioned results where previously presented by the authors in two papers at the LiCS conference [6, 12] and that the current paper gives a unify and complete presentation of their proofs.

Related Work

We already discussed the previous work on MSO model-checking again regular trees [42], algebraic trees [18], trees generated by safe recursion schemes [15, 34], trees generated by possibly unsafe order-2 recursion schemes [35] and general recursion schemes [38]. The proof presented in the present paper (*i.e.* going through the connection with collapsible pushdown games) was first presented in [29]. Following initial ideas in [1] and [36], Kobayashi and Ong gave yet another proof of Ong's decidability result: their proof [37] consists in showing that, given a recursion scheme and an MSO formula, one can construct an intersection type system such that the scheme is typable in the type system if and only if the property is satisfied by the scheme; typability is then reduced to solving a parity game.

Piterman and Vardi [41] studied the global model checking problem for regular trees and prefix-recognisable graphs using two-way alternating parity tree automata. Extending their results, Carayol et al. [11] showed that the winning regions of parity games played over the transition graphs of higher-order pushdown automata (a strict subclass of CPDA) are regular. Later, using game semantics, Broadbent and Ong [7] showed that for every order-n recursion scheme $\mathcal{S}$, the set of nodes in $[\![\mathcal{S}]\!]$ that are definable by a given modal μ-calculus formula is recognisable by an order-n (non-deterministic) collapsible pushdown word automaton. The result we prove here (previously presented in [6]) is stronger as μ-calculus reflection implies that the nodes are recognisable by a *deterministic* CPDA.

The MSO selection property was first established in [12]. Alternative proofs were later given by Haddad [27, 28], and by Grellois and Melliès in [23]. Both proofs are very different from the one we give here. Indeed, our proof uses the equi-expressivity theorem to restate the problem as a question on CPDA, and a drawback of this approach is that once the answer is given on the CPDA side one needs to go back to the scheme side, which is not complicated but yields a scheme that is very different from the original one. The advantage of the approaches in [28] (built on top of the intersection types approach by Kobayashi and Ong [37]) and in [23] (based on purely denotational arguments and connections with linear logic) is to work directly on the recursion scheme and to succeed to provide as a selector a scheme obtained from the original one by adding duplicated and annotated versions of the terminals.

In a recent work [40], Parys considered the logic WMSO+U, an extension of weak monadic second-order logic (*i.e.* MSO logic where second-order quantification is limited to range on *finite* sets) by the unbounding quantifier, expressing the fact that there exist arbitrarily large finite sets satisfying a given property its extension. This logic is incomparable with MSO logic. He showed that model-checking is decidable and that both reflection and selection hold for trees generated by recursion schemes.

Structure of This Paper

The article is organised as follows. Section 2 introduces the main concepts and some classical results. In Section 3 we introduce higher-order recursion schemes and in Section 4 collapsible pushdown automata; we then give in Section 5 few known results that we build on in the rest

of the paper. Section 6 briefly discusses the (local) model-checking problem. The global model-checking and the consequences the refection properties are respectively studied in Section 7 and Section 8. Finally the selection problem is solved in Section 9.

2 PRELIMINARIES

2.1 Basic Notations

When f is a (partial) mapping, we let $\mathsf{dom}(f)$ denote its domain.

2.2 Words

An ***alphabet*** Σ is a (possibly infinite) set of letters. In the sequel Σ^* denotes the set of ***finite words*** over Σ, and Σ^ω the set of ***infinite words*** over Σ. The empty word is written ε and the length of a word u is denoted by $|u|$. Let u be a finite word and v be a (possibly infinite) word. Then $u \cdot v$ (or simply uv) denotes the concatenation of u and v; the word u is a prefix of v iff there exists a word w such that $v = u \cdot w$. A subset $X \subseteq \Sigma^*$ is ***prefix-closed*** if, for every $v \in X$ one has $u \in X$ for any prefix u of v.

2.3 Trees

Let D be a finite set of *directions*. A ***D-tree*** is just a prefix-closed subset T of D^* whose elements are called *nodes*. For a node $u \in T$, an element of the form $u \cdot d$ for some $d \in D$ is called the d-child (or simply a child if d does not matter) of u. A node with no child is called a *leaf* while the node ε is the *root* of T.

Let Σ be a finite alphabet. A ***Σ-labelled tree*** is a function $t : \mathsf{Dom}(t) \to \Sigma$ such that $\mathsf{Dom}(t)$ is a D-tree for some set of directions D; for a node node $u \in \mathsf{Dom}(t)$, we refer to $t(u)$ as the *label* of u in t.

If Σ is a ***ranked alphabet*** *i.e.* each Σ-symbol a has an arity $ar(a) \geq 0$, a ***Σ-labelled ranked and ordered*** tree (or simply a Σ-labelled tree if the context is clear) $t : \mathsf{Dom}(t) \to \Sigma$ is a Σ-labelled tree such that the following holds (meaning that the label of a node determines its number of children):

- $\mathsf{Dom}(t)$ is a $\{1, \ldots, m\}$-tree where $m = \max\{ar(a) \mid a \in A\}$;
- for every node $u \in \mathsf{Dom}(t)$, $\{i \mid 1 \leq i \leq m \text{ and } u \cdot i \in \mathsf{Dom}(t)\} = \{1, \ldots, ar(t(u))\}$.

We write $\mathcal{T}^\infty(\Sigma)$ for the set of (finite and infinite) Σ-labelled trees.

2.4 Graphs

Let A be a finite alphabet containing a distinguished symbol $\mathfrak{e}$ standing for silent transition; we let $A_\mathfrak{e} = A \setminus \{\mathfrak{e}\}$. An ***A-labelled graph*** is a pair $G = (V, E)$ where V is a set of vertices and $E \subseteq V \times A \times V$ is a set of edges. For any $(u, a, v) \in E$ we write $u \xrightarrow{a} v$ and we refer to it as an a-edge (*resp.* silent edge if $a = \mathfrak{e}$) with source u and target v. Moreover, we require that for all $u \in V$, if u is the source of a silent transition then u is not the source of any a-transition with $a \neq \mathfrak{e}$.

For a word $w = a_1 \cdots a_n \in A^*$, we define a binary relation $\xrightarrow{w}$ on V by letting $u \xrightarrow{w} v$ if there exists a sequence $v_0, \ldots, v_n$ of elements in V such that $v_0 = u$, $v_n = v$, and for all $i \in [1, n]$, $v_{i-1} \xrightarrow{a_i} v_i$. These definitions are extended to languages over A by taking, for all $L \subseteq A^*$, the relation $\xrightarrow{L}$ to be the union of all $\xrightarrow{w}$ for $w \in L$.

For a word $w = a_1 \cdots a_k \in A_\mathfrak{e}^*$, we denote by $\overset{w}{\Longrightarrow}$ the relation $\xrightarrow{L_w}$ where $L_w = \mathfrak{e}^* a_1 \mathfrak{e}^* \cdots \mathfrak{e}^* a_k \mathfrak{e}^*$ is the set of words over A obtained by inserting arbitrarily many occurrences of $\mathfrak{e}$ in w.

The graph G is said to be ***deterministic*** if for all u, v_1 and v_2 in V and all a in A, if $u \xrightarrow{a} v_1$ and $u \xrightarrow{a} v_2$ then $v_1 = v_2$. From now on we always assume that the graphs are deterministic.

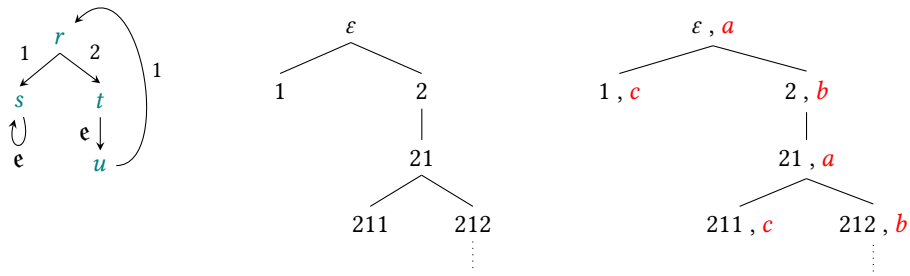


Fig. 1. A deterministic $\{1, 2, \mathfrak{e}\}$-labelled graph G with root r (on the left) together with its associated tree $\mathrm{Tree}(G)$ (in the middle) and its associated $\{a, b, c\}$-labelled ranked and order tree when one lets $ar(a) = 2$, $ar(b) = 1$, $ar(c) = 0$, $\rho(r) = a$, $\rho(s) = c$ and $\rho(u) = b$ (on the right; node labels are written in red).

Consider a deterministic A-labelled graph $G = (V, E)$ together with a distinguished vertex $r \in V$ called its *root*. We associate with it a tree, denoted $\mathrm{Tree}(G)$, with directions in $A_{\mathfrak{e}}$, reflecting the possible behaviours in G starting from the root. For this we let

$$\mathrm{Tree}(G) = \{w \in A_{\mathfrak{e}}^* \mid \exists v \in V,\ r \overset{w}{\Longrightarrow} v\},$$

i.e. $\mathrm{Tree}(G)$ is obtained by unfolding G from its root and contracting all $\mathfrak{e}$-transitions. Figure 1 presents a deterministic $\{1, 2, \mathfrak{e}\}$-labelled graph G together with its associated tree.

In case $G = (V, E)$ is equipped with a vertex-labelling function $\rho : V \to \Sigma$ where Σ is a finite alphabet, one can define a Σ-labelled tree from a pair (G, r) by considering the tree $t : \mathrm{Tree}(G) \to \Sigma$ where $t(w) = \rho(v_w)$ where v_w is the unique vertex in G such that $r \overset{w}{\Longrightarrow} v_w$ and that is not the source of an $\mathfrak{e}$-labelled edge other than an $\mathfrak{e}$-labelled loop (recall that we assumed that a vertex that is the source of a silent transition cannot be the source of any a-transition with $a \neq \mathfrak{e}$). Note that, if Σ is ranked and if $A_{\mathfrak{e}} = \{1, \cdots m\}$ with $m = \max\{ar(a) \mid a \in \Sigma\}$, if we have $\{i \mid i \neq \mathfrak{e} \text{ and } \exists v',\ v \overset{i}{\longrightarrow} v'\} = \{1, \ldots, ar(\rho(v))\}$ for every $v \in V$, then the tree t is a Σ-labelled ranked and ordered tree. An example is given in Figure 1.

2.5 Types

Types are generated by the grammar $\tau ::= o \mid \tau \to \tau$. Every type $\tau \neq o$ can be written uniquely as $\tau_1 \to (\tau_2 \to \cdots \to (\tau_n \to o) \cdots)$, for some $n \geq 1$ which is called its *arity*; the *ground* type o has arity 0. We follow the convention that arrows associate to the right, and simply write $\tau_1 \to \tau_2 \to \cdots \to \tau_n \to o$, which we sometimes abbreviate to $(\tau_1, \ldots, \tau_n, o)$. The ***order*** of a type measures the nesting depth on the left of $\to$. We define $ord(o) = 0$ and $ord(\tau_1 \to \tau_2) = \max(ord(\tau_1) + 1, ord(\tau_2))$. Thus $ord(\tau_1 \to \ldots \to \tau_n \to o) = 1 + \max\{ord(\tau_i) \mid 1 \leq i \leq n\}$. For example, $ord(o \to o \to o \to o) = 1$ and $ord(((o \to o) \to o) \to o) = 3$.

2.6 Terms

Let Υ be a set of typed symbols. Let $f \in \Upsilon$ and τ be a type, we write $f : \tau$ to mean that f has type τ.

The set of ***(applicative) terms of type τ generated from*** Υ, written $\mathcal{T}_\tau(\Upsilon)$, is defined by induction over the following rules. If $f : A$ is an element of Υ then $f \in \mathcal{T}_\tau(\Upsilon)$; if $s \in \mathcal{T}_{\tau_1 \to \tau_2}(\Upsilon)$ and

$t \in \mathcal{T}_{\tau_1}(\Upsilon)$ then $s\ t \in \mathcal{T}_{\tau_2}(\Upsilon)$. For simplicity we write $\mathcal{T}(\Upsilon)$ to mean $\mathcal{T}_o(\Upsilon)$, the set of terms of ground type. Let t be a term, we write $t : \tau$ to mean that t is an term of type τ.

A ranked alphabet Σ can be seen as a set of typed symbols: for each Σ-symbol a its arity $ar(a) \geq 0$ determines its type $\underbrace{o \to \cdots \to o \to}_{ar(a)} o$; in particular every symbol has an order-0 or order-1 type. In this setting, terms in $\mathcal{T}(\Sigma)$ can be identified with Σ-labelled ranked and ordered trees.

2.7 Logic

We now give some brief background on logic (mainly for trees and graphs). More thorough introductions to the topic can be found in many textbooks and survey, *e.g.* in [20, 45].

A ***relational structure*** $\mathfrak{A} = (D, R_1, \ldots, R_k)$ is given by a (possibly *infinite*) set D, called the ***domain*** of $\mathfrak{A}$, and ***relations*** R_i (if we let r_i be the arity of relation R_i, then $R_i \subseteq D^{r_i}$).

Many classical objects can be represented as relational structures. For instance, a Σ-labelled tree $t : \mathsf{Dom}(t) \to \Sigma$ over a ranked alphabet Σ whose symbols have arity at most m corresponds to the relational structure $\mathfrak{t} = (\mathsf{Dom}(t), (S_i)_{1 \leq i \leq m}, \sqsubset, (Q_a)_{a \in \Sigma})$ where S_i is the i-child relation defined by $S_i = \{(u, ui) \mid u, ui \in \mathsf{Dom}(t)\}$, $\sqsubset$ is the strict prefix ordering, and $Q_a = t^{-1}(a)$ for each label $a \in \Sigma$.

In a similar manner a vertex- and edge-labelled graph can be represented as a relational structure: the domain coincides with the set of vertices and one has a binary relation for each edge label and a unary relation for each vertex label.

Properties of relational structures (here trees or graphs) can be expressed thanks to logical formalisms. We start with ***first-order logic (FO)***. First-order formulas on the relational structure $\mathfrak{A}$ may use variables $x, y, \ldots$ ranging over elements in the domain and are built up from atomic formulas of the form

- $x = y$ where both x and y are variables, and
- $R(x_1, \ldots, x_k)$ where R is any relation in $\mathfrak{A}$

by means of the usual Boolean connectives $\neg$, $\vee$, $\wedge$, $\Rightarrow$, $\Leftrightarrow$ and the quantifiers $\exists$ and $\forall$. The semantics is as expected and we do not give it here (see *e.g.* [20]). A formula can contain *free* variables, *i.e.* variables that are not under the scope of a quantifier. In that case the semantics of the formula should be understood relatively to some interpretation of the free variables.

Monadic second-order logic (MSO) extends first-order logic by allowing second-order variables $X, Y, \ldots$ which range over *sets* of elements of the domain (*e.g.* sets of nodes in trees or set of vertices in graphs). In the syntax of MSO formulas one can use the atomic formulas $x \in X$, where x is a first-order variable and X is a second-order variable, whose meaning is that element x belongs to set X.

We will also consider a fixpoint modal logic called ***(modal) μ-calculus***. As we will write only few formulas from μ-calculus and mostly rely on the fact that there exists a strong connection between μ-calculus and parity games, we do not give any definition regarding to this logic and refer the reader to [4, 47].

For a given structure $\mathfrak{A}$ and a formula φ, one writes $\mathfrak{A} \models \varphi$ to mean that φ is true in $\mathfrak{A}$, and $(\mathfrak{A}, p_1 \ldots, p_k) \models \varphi(x_1, \ldots, x_k)$ to mean that φ is true in $\mathfrak{A}$ when one interprets free variables x_i as p_i for each $i = 1, \ldots, k$.

The ***local model-checking problem*** is to decide for a given structure $\mathfrak{A}$ and a formula φ without free variable, whether $\mathfrak{A} \models \varphi$ holds.

The ***global model-checking*** is, for a given structure $\mathfrak{A}$ and a formula $\varphi(x)$ with a first-order free variable x, to provide a finite description of the set $[[\varphi]]_{\mathfrak{A}} = \{p \in D \mid (\mathfrak{A}, p) \models \varphi(x)\}$. In case of μ-calculus (that is interpreted in pointed relational structures, *i.e.* relational structure together

with a distinguished root element), the set $[[\varphi]]_{\mathfrak{A}}$ is intended to be the set of roots that make the formula φ true.

2.8 Games

An ***arena*** is a triple $\mathcal{G} = (G, V_{\mathrm{E}}, V_{\mathrm{A}})$ where $G = (V, E)$ is a graph and $V = V_{\mathrm{E}} \uplus V_{\mathrm{A}}$ is a partition of the vertices among two players, Éloïse and Abelard. For simplicity in the definitions, we assume that G has no dead-end.

Éloïse and Abelard play in $\mathcal{G}$ by moving a pebble along edges. A ***play*** from an initial vertex v_0 proceeds as follows: the player owning v_0 (*i.e.* Éloïse if $v_0 \in V_{\mathrm{E}}$, Abelard otherwise) moves the pebble to a vertex $v_1 \in E(v_0)$. Then the player owning v_1 chooses a successor $v_2 \in E(v_1)$ and so on. As we assumed that there is no dead-end, a play is an infinite word $v_0 v_1 v_2 \cdots \in V^\omega$ such that for all $0 \leq i$ one has $v_{i+1} \in E(v_i)$. A ***partial play*** is a prefix of a play, *i.e.* it is a finite word $v_0 v_1 \cdots v_\ell \in V^*$ such that for all $0 \leq i < \ell$ one has $v_{i+1} \in E(v_i)$.

A ***strategy*** for Éloïse is a function $\varphi_{\mathrm{E}} : V^* V_{\mathrm{E}} \to V$ assigning, to every partial play ending in some vertex $v \in V_{\mathrm{E}}$, a vertex $v' \in E(v)$. Strategies of Abelard are defined likewise, and usually denoted φ_{A}. In a given play $\lambda = v_0 v_1 \cdots$ we say that Éloïse (*resp.* Abelard) ***respects a strategy*** φ_{E} (*resp.* φ_{A}) if whenever $v_i \in V_{\mathrm{E}}$ (*resp.* $v_i \in V_{\mathrm{A}}$) one has $v_{i+1} = \varphi_{\mathrm{E}}(v_0 \cdots v_i)$ (*resp.* $v_{i+1} = \varphi_{\mathrm{A}}(v_0 \cdots v_i)$).

A ***winning condition*** is a subset $\Omega \subseteq V^\omega$ and a (two-player perfect information) ***game*** is a pair $\mathbb{G} = (\mathcal{G}, \Omega)$ consisting of an arena and a winning condition. A game is finite if it is played on a finite arena.

A play λ is ***won*** by Éloïse if and only if $\lambda \in \Omega$; otherwise λ is won by Abelard. A strategy φ_{E} is ***winning*** for Éloïse in $\mathbb{G}$ from a vertex v_0 if any play starting from v_0 where Éloïse respects φ_{E} is won by her. Finally a vertex v_0 is ***winning*** for Éloïse in $\mathbb{G}$ if she has a winning strategy φ_{E} from v_0. Winning strategies and winning vertices for Abelard are defined likewise.

A ***parity winning condition*** is defined by a ***colouring function*** ρ that is a mapping $\rho : V \to C \subset \mathbb{N}$ where C is a *finite* set of ***colours***. The parity winning condition associated with ρ is the set $\Omega_\rho = \{v_0 v_1 \cdots \in V^\omega \mid \liminf(\rho(v_i))_{i \geq 0} \text{ is even}\}$, *i.e.* a play is winning if and only if the smallest colour infinitely often visited is even. A ***parity game*** is a game of the form $\mathbb{G} = (\mathcal{G}, \Omega_\rho)$ for some colouring function.

2.9 Tree Automata

We now introduce the usual model of automata to recognise languages of (possibly infinite) ranked trees.

Let Σ be a ranked alphabet. A ***tree automaton*** is a tuple $\mathcal{A} = (Q, \Sigma, q_0, \Delta, \rho, \mathrm{Acc})$ where Q is a finite set of control states, $q_0 \in Q$ is the initial state, $\Delta \subseteq \bigcup_{a \in \Sigma} Q \times \{a\} \times Q^{ar(a)}$ is a transition relation, $\rho : Q \to C \subset \mathbb{N}$ is a colouring function and $\mathrm{Acc} \subseteq Q \times \Sigma$ is an acceptance condition to handle leaves.

Let $t : \mathrm{Dom}(t) \to \Sigma$ be a Σ-labelled tree. A ***run*** of $\mathcal{A}$ over t is a tree $r : \mathrm{Dom}(t) \to Q$ such that the following holds:

- $r(\varepsilon) = q_0$, *i.e.* the root is labelled by the initial state;
- for every node $u \in \mathrm{Dom}(t)$ one has $(r(u), t(u), r(u \cdot 1), \cdots r(u \cdot ar(t(u)))) \in \Delta$, *i.e.* the local constraints imposed by the transition relation are respected.

The run r is accepting if and only if the following two conditions are satisfied:

- for every leaf $u \in \mathrm{Dom}(t)$ one has $(r(u), t(u)) \in \mathrm{Acc}$;
- for every infinite branch, the smallest colour of a state appearing infinitely often along it is even; formally if $u_0, u_1, u_2, \ldots$ denotes the infinite sequence of nodes read along a branch one has that $\liminf(\rho(r(u_i))_{i \geq 0}$ is even.

Finally, a tree is ***accepted*** by $\mathcal{A}$ if and only if there is an accepting run over it, and we refer to the set of accepted trees as the language recognised by $\mathcal{A}$.

There are tight connections between model-checking MSO logic against ranked trees, solving parity games, and checking whether the language recognised by a tree automaton is empty. We refer the reader to [45] for details on those connections and we only recall here the ones we use in the present paper.

Let Σ be a ranked alphabet, let t be a Σ-labelled ranked tree and let $u_1, \cdots, u_k$ be nodes in t for some $k \geq 0$. Then we write $t_{u_1,\cdots,u_k}$ for the $\Sigma \times \{0,1\}^k$-labelled ranked tree such that $\mathrm{Dom}(t_{u_1,\cdots,u_k}) = \mathrm{Dom}(t)$ and for every $u \in \mathrm{Dom}(t)$, $t_{u_1,\cdots,u_k}(u) = (t(u), (\iota_1(u), \ldots, \iota_k(u)))$ where, for $i = 1, \cdots, k$, one let $\iota_i(u) = 1$ if $u = u_i$ and $\iota_i(u) = 0$ otherwise. In other words $t_{u_1,\cdots,u_k}$ is obtained from t by marking nodes $u_1, \ldots, u_k$.

The first connection is the famous result by Rabin [42].

Theorem 2.1. *Let Σ be a ranked alphabet. The following holds.*

- *For every MSO formula $\varphi(x_1, \ldots, x_k)$ with possibly first-order variables over Σ-labelled ranked trees, one can build an automaton $\mathcal{A}_{\varphi(x_1,\ldots,x_k)}$ such that the trees accepted by $\mathcal{A}_{\varphi(x_1,\ldots,x_k)}$ are exactly those trees $t_{u_1,\ldots,u_k}$ such that $(t, u_1, \ldots, u_k) \models \varphi(x_1, \ldots, x_k)$.*
- *For every tree automaton $\mathcal{A}$ one can build an MSO-formula $\varphi_{\mathcal{A}}$ such that the trees accepted by $\mathcal{A}$ are exactly those where $\varphi_{\mathcal{A}}$ holds.*

The second connection used in this paper is the game-approach to the acceptance problem for tree automata. Let $\mathcal{A} = (Q, \Sigma, q_0, \Delta, \rho, \mathrm{Acc})$ be a tree automaton and let t be a Σ-labelled tree. Consider the following (informal) parity game $\mathbb{G}_{\mathcal{A},t}$. The main vertices in game $\mathbb{G}_{\mathcal{A},t}$ are pairs (u, q) made of a node u in t and a state q in Q. In a node (u, q), Éloïse picks a transition $(q, t(u), q_1, \ldots, q_{ar(t(u))})$ and goes to an intermediate vertex where Abelard picks some i such that $1 \leq i \leq ar(t(u))$, and the play proceeds then from $(u \cdot i, q_i)$. In case $ar(t(u)) = 0$ the play loops forever in (u, q). The colouring function in $\mathbb{G}_{\mathcal{A},t}$ assigns colour $\rho(q)$ to vertex (u, q) if $ar(t(u)) \neq 0$ and otherwise it assigns 0 if $(q, t(u)) \in \mathrm{Acc}$ and 1 otherwise. Intermediate vertices controlled by Abelard gets the maximal colour used in $\mathcal{A}$ (hence, have no impact on who wins a play). The following result is due to Gurevich and Harrington [24].

Theorem 2.2. *Éloïse has a winning strategy in the parity game $\mathbb{G}_{\mathcal{A},t}$ from (ε, q_0) if and only if the tree t is accepted by $\mathcal{A}$.*

3 RECURSION SCHEMES

For each type τ, we assume an infinite set Var_τ of variables of type τ, such that Var_{τ_1} and Var_{τ_2} are disjoint whenever $\tau_1 \neq \tau_2$; and we write Var for the union of Var_τ as τ ranges over types. We use letters $x, y, \varphi, \psi, \chi, \xi$ etc. to range over variables.

A (deterministic) ***recursion scheme*** is a quadruple $\mathcal{S} = (\Sigma, \mathcal{N}, \mathcal{R}, I)$ where

- Σ is a ranked alphabet of ***terminals*** (including a distinguished symbol $\bot : o$ that we shall omit when descriibing Σ)
- $\mathcal{N}$ is a finite set of typed ***non-terminals***; we use upper-case letters F, H, etc. to range over non-terminals
- $I \in \mathcal{N}$ is a distinguished ***initial symbol*** of type o
- $\mathcal{R}$ is a finite set of ***rewrite rules***, *one for each* non-terminal $F : (\tau_1, \ldots, \tau_n, o)$, of the form

$$F\, \xi_1 \cdots \xi_n \;\rightarrow\; e$$

  where each ξ_i is a variable of type τ_i, and e is a term in $\mathcal{T}(\Sigma \cup \mathcal{N} \cup \{\, \xi_1, \cdots, \xi_n \,\})$. Note that the expressions on either side 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.

In this paper we use recursion schemes as generators of Σ-labelled trees. Informally the *value tree* $[\![\, \mathcal{S} \,]\!]$ of (or the tree *generated* by) a recursion scheme $\mathcal{S}$ is a possibly infinite term (of ground type), constructed from the terminals in Σ, that is obtained, starting from the initial symbol I, by unfolding the rewrite rules of $\mathcal{S}$ *ad infinitum*, replacing formal by actual parameters each time.

To define $[\![\, \mathcal{S} \,]\!]$, we first introduce a map $(\cdot)^{\perp} : \bigcup_{A} \mathcal{T}_A(\Sigma \cup \mathcal{N}) \longrightarrow \bigcup_{A:ord(A)\leq 1} \mathcal{T}_A(\Sigma)$ that takes a term and replaces each non-terminal, together with its arguments, by $\perp$. We define $(\cdot)^{\perp}$ by structural recursion as follows: we let f range over Σ-symbols, and F over non-terminals in $\mathcal{N}$

$$\begin{array}{rcl} f^{\perp} & = & f \\ F^{\perp} & = & \perp \\ (st)^{\perp} & = & \begin{cases} \perp & \text{if } s^{\perp} = \perp \\ (s^{\perp} t^{\perp}) & \text{otherwise.} \end{cases} \end{array}$$

Clearly if $s \in \mathcal{T}(\Sigma \cup \mathcal{N})$ is of ground type, so is $s^{\perp} \in \mathcal{T}(\Sigma)$.

Next we define a one-step reduction relation $\rightarrow_{\mathcal{S}}$ which is a binary relation over terms in $\mathcal{T}(\Sigma \cup \mathcal{N})$. Informally, $t \rightarrow_{\mathcal{S}} t'$ just if t' is obtained from t by replacing some occurrence of a non-terminal F by the right-hand side of its rewrite rule in which all formal parameters are in turn replaced by their respective actual parameters, subject to the proviso that the F must occur at the head of a subterm of ground type. Formally $\rightarrow_{\mathcal{S}}$ is defined by induction over the following rules:

- (*Substitution*). $Ft_1 \cdots t_n \rightarrow_{\mathcal{S}} e[t_1/\xi_1, \cdots, t_n/\xi_n]$ where $F\xi_1 \cdots \xi_n \rightarrow e$ is a rewrite rule of G.
- (*Context*). If $t \rightarrow_{\mathcal{S}} t'$ then $(st) \rightarrow_{\mathcal{S}} (st')$ and $(ts) \rightarrow_{\mathcal{S}} (t's)$.

Note that $\mathcal{T}^{\infty}(\Sigma)$ is a complete partial order with respect to the approximation ordering $\sqsubseteq$ defined by: $t \sqsubseteq t'$ just if $\mathsf{Dom}(t) \subseteq \mathsf{Dom}(t')$ and for all $w \in \mathsf{Dom}(t)$, we have $t(w) = \perp$ or $t(w) = t'(w)$. *I.e.* t' is obtained from t by replacing some $\perp$-labelled nodes by Σ-labelled trees. If one views $\mathcal{S}$ as a rewrite system, it is a consequence of the Church-Rosser property [16] that the set $\{\, t^{\perp} \in \mathcal{T}^{\infty}(\Sigma) : \text{there is a finite reduction sequence } S = t_0 \rightarrow_{\mathcal{S}} \cdots \rightarrow_{\mathcal{S}} t_n = t \,\}$ is directed. Hence, we can finally define the Σ-labelled ranked tree $[\![\, \mathcal{S} \,]\!]$, called the ***value tree*** of (or the tree ***generated*** by) $\mathcal{S}$:

$$[\![\, \mathcal{S} \,]\!] \;=\; \sup\{\, t^{\perp} \in \mathcal{T}^{\infty}(\Sigma) : \text{there is a finite reduction sequence } S = t_0 \rightarrow_{\mathcal{S}} \cdots \rightarrow_{\mathcal{S}} t_n = t \,\}.$$

Example 3.1. Consider the order-2 recursion scheme $\mathcal{S}$ with non-terminals $I : o$, $F : (o \rightarrow o) \rightarrow (o \rightarrow o) \rightarrow o$, $C_p : (o \rightarrow o) \rightarrow (o \rightarrow o) \rightarrow o \rightarrow o$, variables $x : o$, $\varphi, \psi : o \rightarrow o$, terminals a, b, c, d of arity 2, 1, 1 and 0 respectively, and the following rewrite rules:

$$\begin{cases} I & \rightarrow & F\, b\, c \\ F\, \varphi\, \psi & \rightarrow & a\, (F\, (C_p\, b\, \varphi)\, (C_p\, c\, \psi))\, (\varphi\, (\psi\, d)) \\ C_p\, \varphi\, \psi\, x & \rightarrow & \varphi\, (\psi\, x) \end{cases}$$

The non-terminal C_p is to be understood as a mechanism to compose its two first arguments and apply the result to the third argument.

The first steps of rewriting of $\mathcal{S}$ are given in Figure 2.

The value tree $t = [\![\, \mathcal{S} \,]\!]$ (depicted in Figure 3) has domain $\{0^k 1^h \mid k \geq 0 \text{ and } h \leq 2k+3\}$, and is defined, for every $k \geq 0$ by $t(0^k) = a$, $t(0^k 1^h) = b$ if $h \leq k+1$, $t(0^k 1^h) = c$ if $k+2 \leq h \leq 2k+2$ and $t(0^k 1^h) = d$ if $h = 2k+3$.

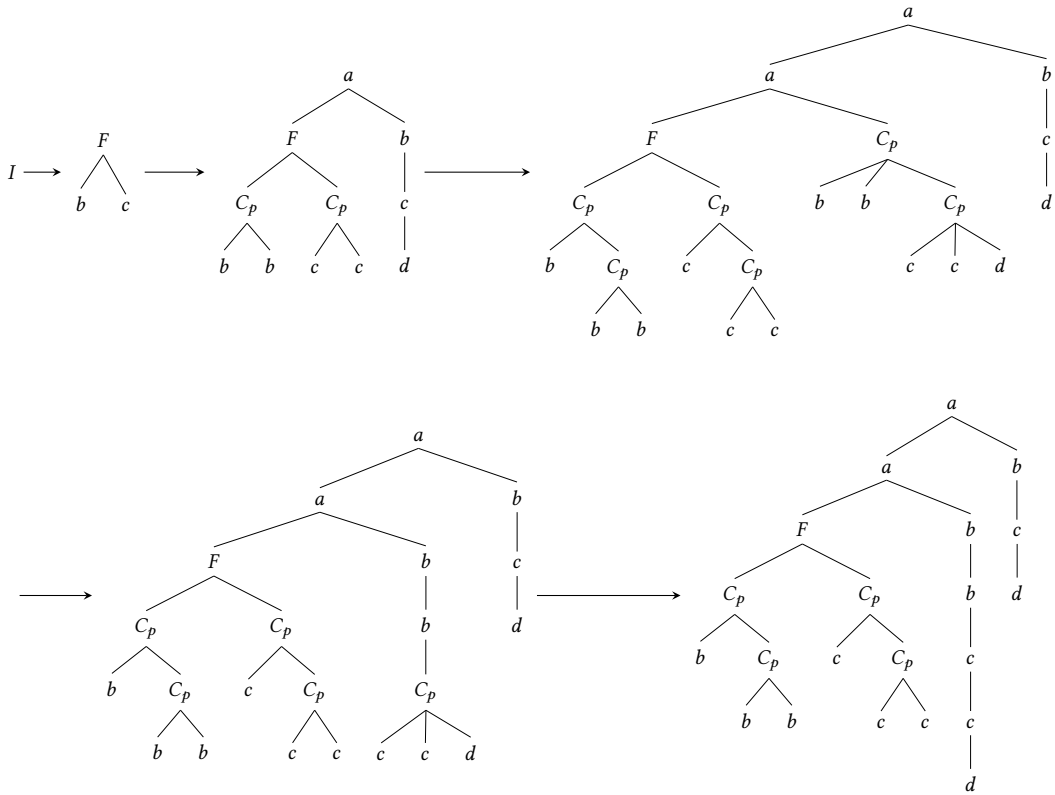


Fig. 2. First steps of rewriting of the recursion scheme from Example 3.1.

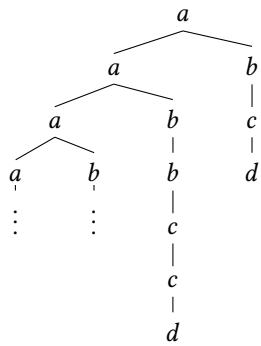


Fig. 3. The tree generated by the recursion scheme from Example 3.1 and by the CPDA from Example 4.5.

4 COLLAPSIBLE PUSHDOWN AUTOMATA

4.1 Stacks with Links and Their Operations

Fix an alphabet Γ of ***stack symbols*** and a distinguished ***bottom-of-stack symbol*** $\perp \in \Gamma$. An ***order-0 stack*** (or simply ***0-stack***) is just a stack symbol. An ***order-(n + 1) stack*** (or simply ***(n + 1)-stack***) s is a non-null sequence, written $[s_1 \cdots s_l]$, of n-stacks such that every non-$\perp$ Γ-symbol γ that occurs in s has a *link* to a stack of some order e (say, where $0 \leq e \leq n$) situated below it in s; we call the link an ***(e + 1)-link***. The ***order*** of a stack s is written $ord(s)$. The ***height*** of a stack $[s_1 \cdots s_l]$ is defined as l.

Remark 4.1. One way to give a formal semantics of the stack operations is to work with appropriate numeric representations of the links as explained in [30, Section 3.2]. We believe that the informal presentation should be sufficient for this work and hence refer the reader to [30] for a formal definition of stacks.

As usual, the bottom-of-stack symbol $\perp$ cannot be popped from or pushed onto a stack. Thus we require an ***order-1 stack*** to be a non-null sequence $[\gamma_1 \cdots \gamma_l]$ of elements of Γ such that for all $1 \leq i \leq l$, $\gamma_i = \perp$ iff $i = 1$. We inductively define $\perp_k$, the ***empty k-stack***, as follows: $\perp_0 = \perp$ and $\perp_{k+1} = [\perp_k]$.

We first define the operations pop_i and top_i with $i \geq 1$: $top_i(s)$ returns the top $(i-1)$-stack of s, and $pop_i(s)$ returns s with its top $(i-1)$-stack removed. Precisely let $s = [s_1 \cdots s_{l+1}]$ be a stack with $1 \leq i \leq ord(s)$:

$$top_i(\underbrace{[s_1 \cdots s_{l+1}]}_{s}) = \begin{cases} s_{l+1} & \text{if } i = ord(s) \\ top_i(s_{l+1}) & \text{if } i < ord(s) \end{cases}$$

$$pop_i(\underbrace{[s_1 \cdots s_{l+1}]}_{s}) = \begin{cases} [s_1 \cdots s_l] & \text{if } i = ord(s) \text{ and } l \geq 1 \\ [s_1 \cdots s_l \; pop_i(s_{l+1})] & \text{if } i < ord(s) \end{cases}$$

By abuse of notation, we set $top_{ord(s)+1}(s) = s$. Note that $pop_i(s)$ is undefined if $top_{i+1}(s)$ is a one-element i-stack. For example $pop_2([[\perp \alpha \beta]])$ and $pop_1([[\perp \alpha \beta][\perp]])$ are both undefined.

There are two kinds of *push* operations. We start with the *order-1 push*. Let γ be a non-$\perp$ stack symbol and $1 \leq e \leq ord(s)$, we define a new stack operation $push_1^{\gamma,e}$ that, when applied to s, first attaches a link from γ to the $(e-1)$-stack *immediately* below the top $(e-1)$-stack of s, then pushes γ (with its link) onto the top 1-stack of s. Formally for $1 \leq e \leq ord(s)$ and $\gamma \in (\Gamma \setminus \{\perp\})$, we define

$$push_1^{\gamma,e}(\underbrace{[s_1 \cdots s_{l+1}]}_{s}) = \begin{cases} [s_1 \cdots s_l \; push_1^{\gamma,e}(s_{l+1})] & \text{if } e < ord(s) \\ [s_1 \cdots s_l \; s_{l+1} \; \gamma^{\dagger}] & \text{if } e = ord(s) = 1 \\ [s_1 \cdots s_l \; push_1^{\widehat{\gamma}}(s_{l+1})] & \text{if } e = ord(s) \geq 2 \text{ and } l \geq 1 \end{cases}$$

where

- $\gamma^{\dagger}$ denotes the symbol γ with a link to the 0-stack s_{l+1}
- $\widehat{\gamma}$ denotes the symbol γ with a link to the $(e-1)$-stack s_l; and we define

$$push_1^{\widehat{\gamma}}(\underbrace{[t_1 \cdots t_{r+1}]}_{t}) = \begin{cases} [t_1 \cdots t_r \; push_1^{\widehat{\gamma}}(t_{r+1})] & \text{if } ord(t) > 1 \\ [t_1 \cdots t_{r+1} \; \widehat{\gamma}] & \text{otherwise } i.e. \; ord(t) = 1 \end{cases}$$

The higher-order $push_j$, where $j \geq 2$, simply duplicates the top $(j-1)$-stack of s. Precisely, let $s = [s_1 \cdots s_{l+1}]$ be a stack with $2 \leq j \leq ord(s)$:

$$push_j(\underbrace{[s_1 \cdots s_{l+1}]}_{s}) = \begin{cases} [s_1 \cdots s_{l+1} \; s_{l+1}] & \text{if } j = ord(s) \\ [s_1 \cdots s_l \; push_j(s_{l+1})] & \text{if } j < ord(s) \end{cases}$$

In case $j = ord(s)$ above, the link structure of s_{l+1} is preserved by the copy that is pushed on top by $push_j$.

We also define, for any stack symbol γ an operation on stacks that rewrites the topmost stack symbol *without modifying* its link. Formally:

$$rew_1^\gamma \underbrace{[s_1 \cdots s_{l+1}]}_{s} = \begin{cases} [s_1 \cdots s_l \, rew_1^\gamma s_{l+1}] & \text{if } ord(s) > 1 \\ [s_1 \cdots s_l \, \widehat{\gamma}] & \text{if } ord(s) = 1 \text{ and } l \geq 1 \end{cases}$$

where $\widehat{\gamma}$ denotes the symbol γ with a link to the same target as the link from s_{l+1}. Note that $rew_1^\gamma(s)$ is undefined if either $top_2(s)$ or s is the empty 1-stack.

Finally there is an important operation called *collapse*. We say that the n-stack s_0 is a ***prefix*** of an n-stack s, written $s_0 \leq s$, just in case s_0 can be obtained from s by a sequence of (possibly higher-order) *pop* operations. Take an n-stack s where $s_0 \leq s$, for some n-stack s_0, and $top_1(s)$ has a link to $top_e(s_0)$. Then *collapse* s is defined to be s_0.

Example 4.2. To avoid clutter, when displaying n-stacks in examples, we shall omit 1-links (indeed by construction they can only point to the symbol directly below), writing e.g. $[[\bot][\bot\alpha\,\beta]]$ instead of $[[\bot][\bot\,\overset{\curvearrowleft}{\alpha}\,\overset{\curvearrowleft}{\beta}]]$.

Take the 3-stack $s = [[[\bot\,\alpha]]\,[[\bot][\bot\,\alpha]]]$. We have

$$\begin{aligned} push_1^{\gamma,2}(s) &= [[[\bot\,\alpha]]\,[[\bot]\overset{\curvearrowleft}{[\bot\,\alpha\,\gamma]}]]] \\ collapse\,(push_1^{\gamma,2}(s)) &= [[[\bot\,\alpha]]\,[[\bot]]] \\ \underbrace{push_1^{\gamma,3}(rew_1^\beta(push_1^{\gamma,2}(s)))}_{\theta} &= [[[\bot\,\alpha]]\,\overset{\curvearrowleft}{[[\bot]\overset{\curvearrowleft}{[\bot\,\alpha\,\beta\,\gamma]}]}]]. \end{aligned}$$

Then $push_2(\theta)$ and $rew_1^\alpha(push_3(\theta))$ are respectively

$$\overset{\curvearrowleft}{[[[\bot\,\alpha]]\,[[\bot]\overset{\curvearrowleft}{[\bot\,\alpha\,\beta\,\gamma]}\overset{\curvearrowleft}{[\bot\,\alpha\,\beta\,\gamma]}]]]} \text{ and}$$

$$\overset{\curvearrowleft}{[[[\bot\,\alpha]]\,[[\bot]\overset{\curvearrowleft}{[\bot\,\alpha\,\beta\,\gamma]}]\,[[]\overset{\curvearrowleft}{[\bot\,\alpha\,\beta\,\alpha]}]]}.$$

We have $collapse\,(push_2(\theta)) = collapse\,(rew_1^\alpha(push_3(\theta))) = collapse(\theta) = [[[\bot\,\alpha]]]$.

The set Op_n^Γ of order-n CPDA ***stack operations*** over stack alphabet Γ (or simply Op_n if Γ is clear from the context) comprises six types of operations:

(1) pop_k for each $1 \leq k \leq n$,
(2) $push_j$ for each $2 \leq j \leq n$,
(3) $push_1^{\gamma,e}$ for each $1 \leq e \leq n$ and each $\gamma \in (\Gamma \setminus \{\bot\})$,
(4) rew_1^γ for each $\gamma \in (\Gamma \setminus \{\bot\})$,
(5) *collapse*, and
(6) *id* for the identity operation (*i.e.* $id(s) = s$ for all stack s).

4.2 Collapsible Pushdown Automata (CPDA)

An ***order-n (deterministic) collapsible pushdown automaton with input*** (n-CPDA) is a 6-tuple $(A, \Gamma, Q, \delta, q_0, F)$ where A is an input alphabet containing a distinguished symbol $\mathfrak{e}$ standing for silent transition, Γ is a stack alphabet, Q is a finite set of states, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is the set of final states and $\delta : Q \times \Gamma \times A \to Q \times Op_n$ is a transition (partial) function such that,

for all $q \in Q$ and $\gamma \in \Gamma$, if $\delta(q, \gamma, \mathfrak{e})$ is defined then for all $a \in A$, $\delta(q, \gamma, a)$ is undefined (i.e. if some silent transition can be taken, then no other transition is possible).

In the special case where $\delta(q, \gamma, \mathfrak{e})$ is undefined for all $q \in Q$ and $\gamma \in \Gamma$ we refer to $\mathcal{A}$ as an $\mathfrak{e}$-free n-CPDA. In the special case where δ never performs a *collapse* (*i.e.* links can safely be forgotten) we obtain the (weaker) model of ***higher-order pushdown automata***.

Configurations of an n-CPDA are pairs of the form (q, s) where $q \in Q$ and s is an n-stack over Γ; the ***initial configuration*** is $(q_0, \bot_n)$ and ***final configurations*** are those whose control state belongs to F. Note that in some context (*e.g.* when generating a tree or a game), final configurations are useless but we still assume a set F for homogeneity in definitions.

An n-CPDA $\mathcal{A} = (A, \Gamma, Q, \delta, q_0, F)$ naturally defines an A-labelled *deterministic* (transition) graph $\mathrm{Graph}(\mathcal{A}) = (V, E)$ whose vertices V are the configurations of $\mathcal{A}$ and whose edge relation $E \subseteq V \times A \times V$ is given by: $((q, s), a, (q', s')) \in E$ iff $\delta(q, top_1(s), a) = (q', op)$ and $s' = op(s)$. Such a graph is called an ***n-CPDA graph***.

Example 4.3. Consider the order-2 CPDA $\mathcal{A} = (\{1, 2, \mathfrak{e}\}, \{\bot, \alpha\}, \{q_a, q_b, q_c, q_d, \widetilde{q}_a, \widetilde{q}_b, \widetilde{q}_c\}, \delta, \widetilde{q}_a, q_d)$ with δ as follows:

- $\delta(\widetilde{q}_a, \bot, \mathfrak{e}) = \delta(q_a, \alpha, 1) = (q_a, push_1^{\alpha})$;
- $\delta(q_a, \alpha, 2) = (\widetilde{q}_b, push_2)$;
- $\delta(\widetilde{q}_b, \alpha, \mathfrak{e}) = \delta(q_b, \alpha, 2) = (q_b, pop_1)$;
- $\delta(q_b, \bot, 2) = (\widetilde{q}_c, pop_2)$;
- $\delta(\widetilde{q}_c, \alpha, \mathfrak{e}) = \delta(q_c, \alpha, 2) = (q_c, pop_1)$;
- $\delta(q_c, \bot, 2) = (q_d, id)$;

Its transition graph is depicted in Figure 4.

In this paper we will use n-CPDA for three different purposes: as words acceptors, as generators for infinite trees and as generators of parity game.

4.3 Using an n-CPDA as a Words Acceptor

A order-n CPDA $\mathcal{A} = (A, \Gamma, Q, \delta, q_0, F)$ accepts the set of words $w \in (A \setminus \{\mathfrak{e}\})^*$ labelling a run from the initial configuration to a final configuration (interpreting $\mathfrak{e}$ as a silent move). We write $L(\mathcal{A})$ for the accepted language.

Following the notations from Section 2.4, and letting, for a word $w = a_1 \cdots a_k \in A_{\mathfrak{e}}^*$, $L_w = \mathfrak{e}^* a_1 \mathfrak{e}^* \cdots \mathfrak{e}^* a_k \mathfrak{e}^*$ one has

$$L(\mathcal{A}) = \{w \in A_{\mathfrak{e}}^* \mid (q_0, \bot_n) \xrightarrow{L_w} (q_f, s) \text{ with } q_f \in F\}$$

Example 4.4. Consider again the order-2 CPDA from Example 4.3. Then its accepted language is $\{1^k 2^{2k+3} \mid k \geq 0\}$.

4.4 Using an n-CPDA as an Infinite Tree Generator

We now explain how to generate a Σ-labelled ranked and ordered trees using an n-CPDA. For this, let Σ be a ranked alphabet, let $m = \max\{ar(a) \mid a \in \Sigma\}$ and consider an n-CPDA $\mathcal{A} = (A, \Gamma, Q, \delta, q_0, F)$ where $A = \{1, \ldots, m\} \cup \{\mathfrak{e}\}$ together with a function $\rho : Q \to \Sigma$ that we extend as a function from configurations of $\mathcal{A}$ by letting $\rho((q, s)) = \rho(q)$. Moreover, assume that, for all $q \in Q$ and $\gamma \in \Gamma$, $\{a \mid a \neq \mathfrak{e} \text{ and } (q, \gamma, a) \in \mathrm{Dom}(\delta)\} = \{1, \ldots, ar(\rho(q))\}$.

Then, let $G = \mathrm{Graph}(\mathcal{A})$ be the A-labelled deterministic transition graph associated with $\mathcal{A}$ and following Section 2.4, consider the tree $t_{\mathcal{A}} : \mathrm{Tree}(G) \to \Sigma$ obtained by taking as domain the tree obtained by unfolding G from the initial configuration $(q_0, \bot_n)$, contracting all $\mathfrak{e}$-transitions and labelling its nodes thanks to function ρ.

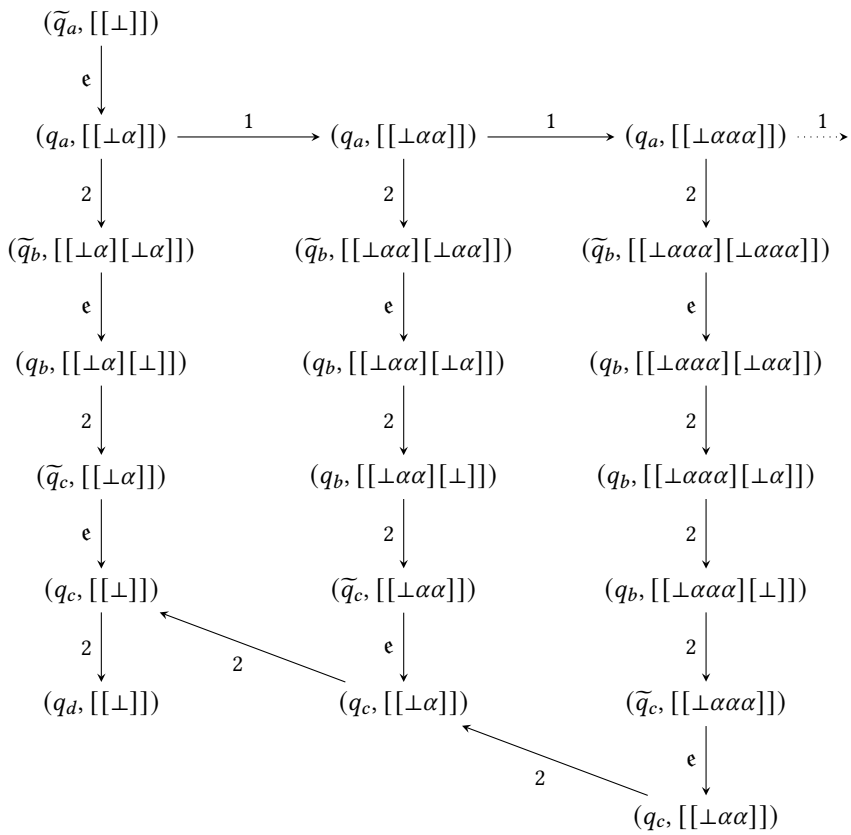


Fig. 4. The transition graph of the order-2 CPDA from Example 4.3

Example 4.5. Consider again the order-2 CPDA from Example 4.3 and the ranked alphabet $\Sigma = \{a, b, c, d\}$ where $ar(a) = 2$, $ar(b) = ar(c) = 1$, and $ar(d) = 0$. Define $\rho : Q \to \Sigma$ by letting $\rho(q_a) = \rho(\widetilde{q}_a) = a$, $\rho(q_b) = \rho(\widetilde{q}_b) = b$, $\rho(q_c) = \rho(\widetilde{q}_c) = c$ and $\rho(q_d) = d$. Then, the tree $t_{\mathcal{A}}$ generated by $\mathcal{A}$ with the labelling function ρ is the one depicted in Figure 3.

4.5 Using an n-CPDA to Define a Parity Game

Let $\mathcal{A} = (A, \Gamma, Q, \delta, q_0, F)$ be an order-n CPDA and let (V, E) be the graph obtained from $\mathrm{Graph}(\mathcal{A})$ by removing edge-labels. To stick to the definition in Section 2.8 we assume that $\mathrm{Graph}(\mathcal{A})$ has no dead-end. Let $Q_{\mathrm{E}} \uplus Q_{\mathrm{A}}$ be a partition of Q and let $\rho : Q \to C \subset \mathbb{N}$ be a colouring function (over states). Altogether they define a partition $V_{\mathrm{E}} \uplus V_{\mathrm{A}}$ of V whereby a vertex belongs to V_{E} iff its control state belongs to Q_{E}, and a colouring function $\rho : V \to C$ where a vertex is assigned the colour of its control state. The structure $\mathcal{G} = (\mathrm{Graph}(\mathcal{A}), V_{\mathrm{E}}, V_{\mathrm{A}})$ defines an arena and the pair $\mathbb{G} = (\mathcal{G}, \rho)$ defines a parity game (that we call an ***n-CPDA parity game***).

In this context, one can also use a CPDA to define a strategy. Indeed, fix an order-n CPDA $\mathcal{A} = (A, \Gamma, Q, \delta, q_0, F)$ defining a n-CPDA parity game $\mathbb{G}$.

Consider a partial play $v_0 v_1 \cdots v_\ell$ in $\mathbb{G}$ where $v_0 = (q_0, \bot_n)$, together with the sequence of labels $\lambda \in A^*$ of the corresponding path. As $\mathcal{A}$ is deterministic, one can represent a strategy as a (partial) function $\varphi : A^* \to A$.

Now let $\mathcal{A}' = (A, \Gamma', Q', \delta', q'_0, F')$ be an order-n CPDA together with a function $\tau : Q' \times \Gamma' \to A$. Then $\mathcal{A}'$ defines a strategy $\varphi_{\mathcal{A}'}$ for Éloïse in $\mathbb{G}$ by letting $\varphi_{\mathcal{A}'}(\lambda) = \tau((q', top_1(s')))$ where (q', s') is the (unique) configuration reached by $\mathcal{A}'$ from its initial configuration by reading λ (seen as an element in A^* as explained above).

We say that $\mathcal{A}$ and $\mathcal{A}'$ are ***synchronised*** if, for all $u \in A^*$ and $a \in A$, if (q, s) and (q', s') denote the respective configurations reached by $\mathcal{A}$ and $\mathcal{A}'$ when reading u from their initial configuration, then $\delta(q, top_1(s), a)$ and $\delta'(q', top_1(s'), a)$ yields the same stack operation up to a renaming of γ in $push_1^{\gamma,e}$ and rew_1^{γ}. In particular it implies that the stacks s and s' have the same shape if one defines the *shape* of a stack as the stack obtained by replacing all symbols appearing in s by a fresh symbol $\sharp$ (but keeping the links).

5 KNOWN RESULTS

We give now a few known results that we will build on in the following. Complete proofs of these results can be found in the companion papers [30] and [5].

5.1 The Equi-Expressivity Theorem

In [29, 30] the following equi-expressivity result was proved (see also [12] for an alternative proof).

Theorem 5.1 (Equi-expressivity). *Order-n recursion schemes and order-n collapsible pushdown automata are equi-expressive for generating trees. That is, we have the following.*

(1) *Let $\mathcal{S}$ be an order-n recursion scheme over Σ and let t be its value tree. There is an order-n CPDA $\mathcal{A} = (A, \Gamma, Q, \delta, q_0, F)$, and a function $\rho : Q \to \Sigma$ such that t is the tree generated by $\mathcal{A}$ and ρ.*
(2) *Let $\mathcal{A} = (A, \Gamma, Q, \delta, q_0, F)$ be an order-n CPDA, and let t be the Σ-labelled tree generated by $\mathcal{A}$ and a function $\rho : Q \to \Sigma$. There is an order-n recursion scheme over Σ whose value tree is t.*

Moreover the inter-translations between schemes and CPDA are polytime computable.

5.2 Collapsible Pushdown Parity Games And μ-Calculus Definable Sets

We refer the reader to [5] for a unified and self content presentation of the following results.

Collapsible pushdown parity games were first studied in [29] where it was established that one could decide the winner from a given initial vertex.

Theorem 5.2. *Let $\mathcal{A} = (A, \Gamma, Q, \delta, q_0, F)$ be the n-CPDA and let $\mathbb{G}$ be an n-CPDA parity game defined from $\mathcal{A}$. Then deciding whether $(q_0, \bot_n)$ is winning for Éloïse is an n-ExpTime complete problem.*

This was further extended in [6] where the computation of a (finite presentation) of the winning region was considered. In particular, from a game $\mathbb{G}$ one can build a new game that behaves the same but where the winning region is explicitly marked.

Theorem 5.3. *Let $\mathcal{A} = (A, \Gamma, Q, \delta, q_0, F)$ be an n-CPDA and let $\mathbb{G}$ be the n-CPDA parity game defined from $\mathcal{A}$. Then, one can build an order-n CPDA $\mathcal{A}' = (A, \Gamma', Q', \delta', q'_0, F')$ such that the following holds.*

(1) *Restricted to the reachable configurations from their respective initial configuration, the transition graph of $\mathcal{A}$ and $\mathcal{A}'$ are isomorphic.*
(2) *For every configuration (q, s) of $\mathcal{A}$ that is reachable from the initial configuration, the corresponding configuration (q', s') of $\mathcal{A}'$ is such that (q, s) is winning for Éloïse in $\mathbb{G}$ if and only if $q' \in F$.*

Regarding μ-calculus global model-checking against graphs generated by CPDA the following logical counterpart of Theorem 5.3 was first proved in [6].

Theorem 5.4. *Let $\mathcal{A} = (\Gamma, Q, \delta, q_0, F)$ be an n-CPDA and let φ be a μ-calculus formula defining a subset of vertices in the configuration graph of $\mathcal{A}$. Then, one can build an order-n CPDA $\mathcal{A}' = (A, \Gamma', Q', \delta', q_0', F')$ and a mapping $\chi : Q' \to Q$ such that the following holds.*

(1) *Restricted to the reachable configurations from their respective initial configuration, the transition graph of $\mathcal{A}$ and $\mathcal{A}'$ are isomorphic.*
(2) *For every configuration (q, s) of $\mathcal{A}$ that is reachable from the initial configuration, the corresponding configuration (q', s') of $\mathcal{A}'$ is such that $q = \chi(q')$, and φ holds in (q, s) if and only if $q' \in F$.*

Finally, in [12], the computation of a finite description of a winning strategy was studied.

Theorem 5.5. *Let $\mathcal{A}$ be an n-CPDA defining an n-CPDA parity game $\mathbb{G}$. If the initial configuration is winning for Éloïse then one can effectively construct an n-CPDA $\mathcal{A}'$ that is synchronised with $\mathcal{A}$ and realises a well-defined winning strategy for Éloïse in $\mathbb{G}$ from the initial configuration.*

6 LOCAL MODEL-CHECKING

Recall that, for a formula φ (without free variables if one considers MSO logic; on the pointed tree if one considers μ-calculus) and a tree t the *(local) model-checking* problem asks to decide whether φ holds in t.

In the case of trees, MSO formulas without free variable and μ-calculus formulas have the same expressive power (see *e.g.* [32]). It is also a very standard result (obtained by combining the previous equivalence with Theorem 2.1 and Theorem 2.2; see also [4, 47] for a direct construction) that μ-calculus model-checking and deciding the winner in a parity game played on an arena obtained by considering a synchronised product of the tree to model-check together with a finite graph (describing the formula and its dynamics) are two equivalent problems. Hence, the equi-expressivity Theorem together with the fact that CPDA parity games are decidable (Theorem 5.2), directly imply the decidability of the MSO/μ-calculus model-checking problem against trees generated by recursion schemes/CPDA. Note that historically this result was first established for trees generated by recursion schemes by Ong in [38] using tools from innocent game semantics (in the sense of Hyland and Ong [31])

Theorem 6.1. *MSO (equivalently μ-calculus) local model-checking problem is decidable for any tree generated by a recursion scheme (equivalently by a collapsible pushdown automaton).*

7 GLOBAL MODEL-CHECKING

Recall that, for a formula φ and a tree t the *global model-checking* asks (if there is one) for a description of the set $[[\varphi]]_t$ of nodes of t where φ holds.

7.1 Exogeneous & Endogeneous Approaches to Global Model-Checking

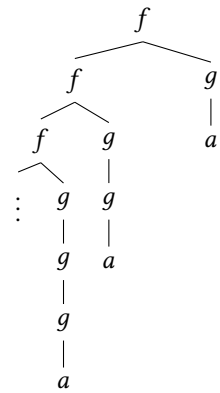
We present here two approaches to the global model-checking problem: the exogenous one where the set is described by an external device (here a CPDA), and the endogenous one which is new and where the description is internalised by a recursion scheme with "polarized" labels.

- *Exogeneous* approach: Given a Σ-labelled tree $t : \mathsf{Dom}(t) \to \Sigma$ and a formula φ, output a description by means of a word acceptor device recognising $[[\varphi]]_t \subseteq \mathsf{Dom}(t)$.
- *Endogeneous* approach: Given a Σ-labelled tree $t : \mathsf{Dom}(t) \to \Sigma$ and a formula φ, output a finite description of the $(\Sigma \cup \underline{\Sigma})$-labelled tree $t_\varphi : \mathsf{Dom}(t) \to \Sigma \cup \underline{\Sigma}$, where $\underline{\Sigma} = \{\underline{\sigma} \mid \sigma \in \Sigma\}$ is a marked copy of Σ, such that t_φ and t have the same domain, and $t_\varphi(u) = \underline{t(u)}$ if $u \in [[\varphi]]_t$ and $t_\varphi(u) = t(u)$ otherwise.

Example 7.1. Let $\mathcal{S}$ be the order-2 recursion scheme with non-terminals $I : o$, $F : ((o,o),o,o)$ variables $x : o$, $\zeta : (o,o)$, terminals f, g, a of arity $2, 1, 0$ respectively, and the following production rules:

$$\begin{cases} I & \to & F\,g\,a \\ F\,\zeta\,x & \to & f\,(F\,\zeta\,(\zeta\,x))\,(\zeta\,x) \end{cases}$$

The value tree $t = [\![\, \mathcal{S} \,]\!]$ is the following Σ-labelled tree:



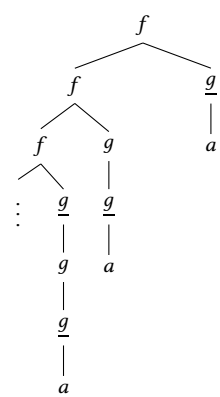
Let φ be the μ-calculus formula defining the nodes which are labelled by g such that the length of the (unique) path to an a-labelled node is odd. Formally (we refer to [4, 47] for syntax and semantics of μ-calculus), $\varphi = p_g \wedge \mu X.(\diamond_1 p_a \vee \diamond_1 \diamond_1 X)$, where p_g (resp. p_a) is a propositional variable asserting that the current node is labelled by g (resp. a).

An *exogenous* approach to the global model-checking problem is to output a description (*e.g.* by means of a finite-state automaton) of the set $[[\varphi]]_t = \{1^n 2 1^k \mid n + k \text{ is odd}\}$, which in this special case is a regular language.

An *endogenous* approach to this problem is to output the following recursion scheme:

$$\begin{cases} I & \to & H\,\underline{g}\,a \\ H\,z & \to & f\,(\underline{H}\,g\,z)\,z \\ \underline{H}\,z & \to & f\,(H\,\underline{g}\,z)\,z \end{cases}$$

with non-terminals $I : o$, $H : (o,o)$ and a variable $z : o$. The value tree of this new scheme is as desired:



We now define a general concept, called *reflection*, and which expresses the ability to perform the endogenous approach within a class of trees for a given logic.

Definition 7.2. (Reflection) Let C be a class of trees, and let $\mathcal{L}$ be some logical formalism. Let t be a tree in C and let φ be an $\mathcal{L}$-formula. We say that a tree $t' \in C$ is a ***φ-reflection*** of t just if $t' = t_\varphi$. We say that the class C is ***$\mathcal{L}$-reflective*** in case for all $t \in C$ and all $\varphi \in \mathcal{L}$ one has $t_\varphi \in C$.

In case the class C is finitely presented (*i.e.* each element of C comes with a finite presentation, *e.g.* given by a recursion scheme or by a collapsible pushdown automaton), we say that C is ***$\mathcal{L}$-effectively-reflective*** if C is $\mathcal{L}$-reflective and moreover one can effectively construct, for any (presentation of) $t \in C$ and any $\varphi \in \mathcal{L}$, (a presentation of) t_φ: *i.e.* there is an algorithm that, given a formula $\varphi \in \mathcal{L}$, transforms a presentation of an element in C into a presentation of its φ-reflection.

In the sequel, we prove that the class of trees generated by recursion schemes/CPDAs is μ-calculus-effectively-reflective as well as MSO-effectively-reflective.

7.2 μ-Calculus Reflection

Regarding μ-calculus, the following result providing an exogeneous and an endogeneous description of μ-calculus definable sets is a simple consequence of the equi-expressivity theorem together with Theorem 5.4.

Theorem 7.3. *Let t be a Σ-labelled tree generated by an order-n recursion scheme $\mathcal{S}$ and φ be a μ-calculus formula.*

(1) *There is an algorithm that takes $(\mathcal{S}, \varphi)$ as its input and outputs an order-n CPDA $\mathcal{A}$ such that $L(\mathcal{A}) = [[\varphi]]_t$.*
(2) *There is an algorithm that takes $(\mathcal{S}, \varphi)$ as its input and outputs an order-n recursion scheme that generates t_φ.*

Proof. First remark that (2) implies (1). To see why this is so, assume that we can construct an order-n recursion scheme generating t_φ. Thanks to Theorem 5.1, we can construct an order-n CPDA $\mathcal{A}$ which, together with a mapping $\rho : Q \mapsto \Sigma \cup \underline{\Sigma}$, generates t_φ. Taking $\{q \in Q \mid \rho(q) \in \underline{\Sigma}\}$ as a set of final states, and using $\mathcal{A}$ as a finite words acceptor it immediately follows that $L(\mathcal{A}) = [[\varphi]]_t$.

We now concentrate on (2). Fix an order-n recursion scheme $\mathcal{S} = (\Sigma, \mathcal{N}, \mathcal{R}, I)$ and let t be its value tree; let $m = \max\{ar(a) \mid a \in \Sigma\}$. Let φ be a μ-calculus formula. Using Theorem 5.1, we can construct an n-CPDA $\mathcal{A} = (A, \Gamma, Q, \delta, q_0, F)$ with $A_{\mathfrak{e}} = \{1, \ldots, m\}$ and a mapping $\rho : Q \to \Sigma$ such that t is the tree generated by $\mathcal{A}$ and ρ.

Let $G = (V, E)$ be the transition graph of $\mathcal{A}$. From Graph($\mathcal{A}$) we define a new (deterministic) graph $G' = (V, E')$ obtained by contracting the $\mathfrak{e}$-labelled edges in G, *i.e.* $E' = \{(v_1, a, v_2) \mid a \in A_{\mathfrak{e}} \text{ and } v_1 \xrightarrow{\mathfrak{e}^* a} v_2\}$. From the definitions it directly follows that G' equipped with the labelling function induced by ρ defines the same tree as G, namely t.

Assume that we have, for every state q of $\mathcal{A}$, a predicate p_q that holds at a node $(q', s) \in V$ iff $q = q'$. Then the formula φ can be translated to a formula φ' on G' as follows: for each $a \in \Sigma$, replace every occurrence of the predicate p_a in φ by the disjunction $\bigvee_{q \in Q, \rho(q) = a} p_q$. Then, by definition of φ' and because G' generates t, one has

$$[[\varphi]]_t = \{u \in A_{\mathfrak{e}}^* \mid (G', v'_u) \models \varphi'\}$$

where we denote by v'_u the (unique, if exists) vertex in G' that is reachable from the initial configuration by a path labeled by u.

In turn φ' can be translated to a formula $\varphi_{\mathfrak{e}}$ on G by replacing in φ' every sub-formula of the form $\diamond_a \psi$ by $\mu X.(\diamond_a \psi \vee \diamond_{\mathfrak{e}} X)$, *i.e.* we replace the assertion "take an a-edge to a vertex where ψ holds"

by the assertion "take a (possibly empty) finite sequence of $\mathfrak{e}$-edges to a vertex from which there is an a-edge to a vertex where ψ holds". Then, by definition of $\varphi_{\mathfrak{e}}$ and from how G' was defined from G, one has, for every $u \in A$, that

$$(G, v_u) \models \varphi_{\mathfrak{e}} \quad \text{iff} \quad (G', v'_{u_{\mathfrak{e}}}) \models \varphi'$$

where we denote by v_u the (unique, if exists) vertex in G that is reachable from the initial configuration by a path labeled by u and where we denote by $v'_{u_{\mathfrak{e}}}$ the (unique, if exists) vertex in G' that is reachable from the initial configuration by a path labeled by the word $u_{\mathfrak{e}} \in A_{\mathfrak{e}}^*$ obtained from u by removing all occurrences of $\mathfrak{e}$.

Now use Theorem 5.4 for $\mathcal{A}$ and $\varphi_{\mathfrak{e}}$, leading to a new order-n CPDA $\mathcal{A}' = (A, \Gamma', Q', \delta', q'_0, F')$ and a mapping $\chi : Q' \to Q$. As the transitions graphs (when restricted to reachable configurations from the initial configuration) of $\mathcal{A}$ and $\mathcal{A}'$ are isomorphic, it follows that $\mathcal{A}'$ and $\rho \circ \chi : Q' \to \Sigma$ generates t.

It follows at once that the tree t_φ is generated by $\mathcal{A}'$ and the mapping $\rho' : Q' \to \Sigma \cup \underline{\Sigma}$ defined by

$$\rho'(q') = \begin{cases} \rho(\chi(q')) & \text{if } q' \notin F \\ \underline{\rho(\chi(q'))} & \text{otherwise.} \end{cases}$$

According to Theorem 5.1, one can construct from $\mathcal{A}'$ an order-n recursion scheme generating t_φ. □

Remark 7.4. There are natural questions concerning complexity. The first one concerns the algorithm in Theorem 7.3: it is n-time exponential in both the size of the scheme and the size of the formula. This is because we need to solve an order-n CPDA parity game when invoking Theorem 5.4 (see [5]) built by taking a product of an order-n CPDA equi-expressive with $\mathcal{S}$ (thanks to Theorem 5.1 its size is polynomial in the one of $\mathcal{S}$) with a finite transition system of polynomial size in that of φ. The second issue concerning complexity is how the size of the new scheme (obtained in the second point of Theorem 7.3) relates to that of $\mathcal{S}$ and φ. For similar reasons, it is n-time exponential in the size of $\mathcal{S}$ and φ. The last one concerns the size of the order-n CPDA in the first point of Theorem 7.3: because it is constructed from the new scheme given by the second point, it is also n-time exponential in the size of $\mathcal{S}$ and φ.

7.3 MSO Reflection

It is natural to ask if trees generated by recursion schemes are reflective with respect to MSO. Indeed, μ-calculus and MSO are equivalent for expressing properties of a deterministic tree at the *root*, but not other nodes; see *e.g.* [32]. In fact one would need backwards modalities to express all of MSO in μ-calculus.

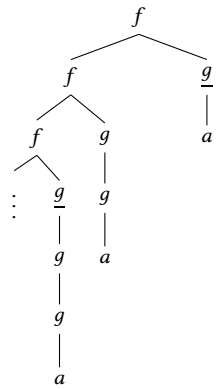
Example 7.5. Consider the following property $\varphi(x)$ (definable in MSO but not in μ-calculus) on nodes x of a tree: "x is the right son of an f-labelled node, and there is a path from x to an a-labelled node which contains an odd number of occurrences of g-labelled nodes".

Returning to the scheme of Example 7.1, an *exogenous* approach to the global model-checking problem is to output a description (*e.g.* by means of a finite-state automaton) of the language $[\![\varphi]\!]_t = \{1^n 2 \mid n \text{ is even}\}$, which in this special case is a regular language.

An *endogenous* approach to this problem is to output the following recursion scheme:

$$\begin{cases} I & \to & \underline{F}\, g\, a \\ \underline{F}\, \varphi\, x & \to & f\,(F\, g\,(\varphi\, x))\,(\underline{g}\, x) \\ F\, \varphi\, x & \to & f\,(\underline{F}\, g\,(\varphi\, x))\,(\underline{g}\, x) \end{cases}$$

with non-terminals $I : o$, $F : ((o,o),o,o)$, $\underline{F} : ((o,o),o,o)$ and variables $x : o$, $\varphi : (o,o)$. The value tree of this new scheme is as desired:



Relying on the μ-calculus reflection, one can prove that the class of trees generated by recursion schemes is MSO-reflective.

Theorem 7.6. *Let t be a Σ-labelled tree generated by an order-n recursion scheme $\mathcal{S}$ and $\varphi(x)$ be an MSO formula.*

(1) *There is an algorithm that takes $(\mathcal{S},\varphi)$ as its input and outputs an order-n CPDA $\mathcal{A}$ such that $L(\mathcal{A}) = [[\varphi]]_t$.*
(2) *There is an algorithm that takes $(\mathcal{S},\varphi)$ as its input and outputs an order-n recursion scheme that generates t_φ.*

Proof. We only concentrate on (2) as it implies (1) using the same argument as in the proof of Theorem 7.3.

For any node u, we let t_u denote the tree obtained from t by marking the node u (and no other node). Recall (see definition on page 9) that formally t_u is a $\Sigma \times \{0,1\}$-labelled tree. The second component of the alphabet is used to indicate the position of marked node. In addition, for a Σ-labelled tree t, we let $\tilde{t}$ denote the $\Sigma \times \{0,1\}$-labelled tree such that $\mathsf{Dom}(t) = \mathsf{Dom}(\tilde{t})$ and for all $u \in \mathsf{Dom}(\tilde{t})$, $\tilde{t}(u) = (t(u),0)$ which corresponds to the tree t in which no node is marked.

Using Theorem 2.1, one can construct a tree automaton $\mathcal{B}_{\varphi(x)}$ that accepts t_u iff $(t,u) \models \varphi(x)$. We let S denote the set of control states of $\mathcal{B}_{\varphi(x)}$.

In order to construct t_φ, we first annotate t with informations on the behaviour of $\mathcal{B}_{\varphi(x)}$ on the subtrees of $\tilde{t}$.

We mark t by μ-calculus definable sets to obtain an enriched tree denoted $\bar{t}$. With each pair $(q,d) \in S \times \{1,\dots,m\}$, where $m = \max\{ar(a) \mid a \in \Sigma\}$, we associate a μ-calculus formula $\psi_{q,d}$ such that, for every node u, $u \in [[\psi_{q,d}]]_t$ iff the d-child of u exists and $\mathcal{B}_{\varphi(x)}$ has an accepting run on $\tilde{t}[ud]$ starting from q, where $\tilde{t}[ud]$ denotes the subtree of $\tilde{t}$ rooted at ud. The existence of $\psi_{q,d}$ is due to fact that acceptance by parity tree automata is expressible in μ-calculus [44]. The tree $\bar{t}$ is the $\Sigma' = \Sigma \times 2^{S\times\{1,\dots,m\}}$-labelled ranked tree with domain $\mathsf{Dom}(t)$ where for every $u \in \mathsf{Dom}(t)$,

$$\bar{t}(u) = (t(u), \{(q,d) \mid \mathcal{B}_{\varphi(x)}[q] \text{ accepts } \tilde{t}[ud]\})$$

where $\mathcal{B}_{\varphi(x)}[q]$ denotes the automaton obtained from $\mathcal{B}_{\varphi(x)}$ by changing the initial state to q. Then, by (successive applications of) Theorem 7.3, $\bar{t}$ can be generated by an order-n collapsible automaton.

Using the annotations on $\bar{t}$, for any node u, one can decide, only considering the path from the root to u, whether $\mathcal{B}_{\varphi(x)}$ accepts t_u. More precisely, there exists a regular words language L over

$\Sigma' \cup \{1, \ldots, m\}$ such that a node u of t satisfies φ if and only if the word obtained by reading in $\bar{t}$ the labels and directions from the root to the node u belongs to L.

To prove the existence of the regular language L, we introduce the notion of partial accepting run of $\mathcal{B}_{\varphi(x)}$ on $\tilde{t}$ stopping at u in state p which is defined similarly to an accepting run except that it is required to assign the state p to the node u and that it is undefined for all nodes below u. Note that all infinite branches for which it is defined are required to satisfy the acceptance condition.

For a node u, we let $P(u)$ denote the set of states $p \in S$ for which $\mathcal{B}_{\varphi(x)}$ has a partial accepting run on $\tilde{t}$ stopping at u in state p. Remark that, by definition, $P(\varepsilon)$ is the set of initial states of $\mathcal{B}_{\varphi(x)}$.

For two nodes u and ud in $\bar{t}$, the set $P(ud)$ can be computed from $P(u)$, the label of u in $\tilde{t}$ and the direction d. Indeed, for a node u labelled by $(\sigma, Q) \in \Sigma \times 2^{S \times \{1,\ldots,m\}}$ in $\bar{t}$ and for all states $q \in S$, q belongs to $P(ud)$ if and only if there exists a state $p \in P(u)$ and a transition $(p, (\sigma, 0), p_1, \ldots, p_k)$ of $\mathcal{B}_{\varphi(x)}$ such that for all $i \in [1, k] \setminus \{d\}$, (p_i, i) belongs to Q and p_d is equal to q.

Using this property, one easily constructs a finite automaton over $\Sigma' \cup \{1, \ldots, m\}$ which after reading the path from the root to a node u computes the set $P(u)$.

The final remark is that for a node u, we can decide whether $\mathcal{B}_{\varphi(x)}$ accepts t_u by only considering the set $P(u)$ and the label of u in $\bar{t}$. Indeed, for a node u labeled by $(\sigma, Q) \in \Sigma \times 2^{S \times \{1,\ldots,m\}}$ in $\bar{t}$, t_u is accepted by $\mathcal{B}_{\varphi(x)}$ if and only if there exists a node $p \in P(u)$ and a transition $(p, (\sigma, 1), p_1, \ldots, p_k)$ such that for all $i \in [1, k]$, (p_i, i) belongs to Q.

Hence, one easily constructs a finite automaton accepting the language L.

Finally, an order-n CPDA generating t_φ is obtained by taking a synchronised product between an order-n CPDA generating $\bar{t}$ and a finite *deterministic* automaton recognising L: a node in the generated tree is marked iff the associated control state in the automaton recognising L is final. □

8 SOME CONSEQUENCES OF MSO REFLECTION

We derive two consequences of the MSO reflection. The first one (Corollary 8.1) is to show how MSO reflection can be used to construct, starting from a scheme that may have non-productive rules, an equivalent one that does not have such divergent computations. The second application consists in proving (Theorem 8.4) that the class of trees generated by recursion schemes is closed under the operation of MSO interpretation followed by tree unfolding, hence providing a result in the same flavour as the one obtained by Caucal for safe schemes in [15].

8.1 Avoiding Divergent Computations

Let $\mathcal{S}$ be the order-2 recursion scheme with non-terminals $I : o$, $H : (o, o)$, $F : ((o, o, o), o)$, variables $z : o$, $\varphi : (o, o, o)$, terminals $\Sigma = \{f, a\}$ of arity 2 and 0 respectively, and the following production rules:

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

The second rule $H\,z \;\to\; H\,(H\,z)$ is *divergent* (or *non-productive*), meaning that it generates a node labelled $\perp$ in the value tree (see Figure 5).

Consider now a scheme with production rules

$$\begin{array}{rcl} I & \to & f\,(\boxtimes)\,(F\,f) \\ F\,\varphi & \to & \varphi\,a\,(F\,\varphi) \end{array}$$

Then it produces the same tree as the previous scheme up to relabelling nodes labelled by $\boxtimes$ by $\perp$. Note that this latter scheme does not lead any divergent computation.

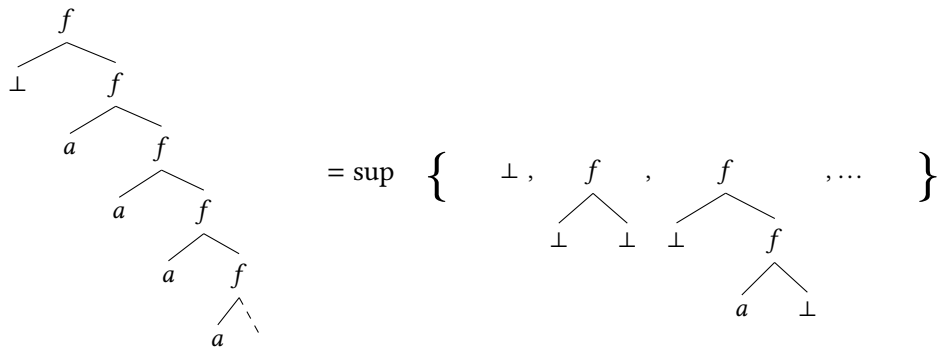


Fig. 5. The value tree of the recursion scheme $\mathcal{S}$

Actually, MSO effective reflection leads to the following general result, showing that one can always design an "equivalent" non-divergent scheme. See also [26, 28, 43] for alternative proofs of this statement.

Corollary 8.1. *Let $\mathcal{S}$ be an order-n recursion scheme with terminals Σ and generating a tree t. Then one can construct another order-n scheme $\mathcal{S}_\perp$ with terminals $\Sigma \cup \{\boxtimes\}$ where $\boxtimes : o$ is a fresh symbol of arity 0 and such that*

(1) *The trees t and $t_\perp$ have the same domain;*
(2) *the tree $t_\perp$ generated by $\mathcal{S}_\perp$ does not contain any node labelled $\perp$;*
(3) *for every node u, $t_\perp(u) = t(u)$ if $t(u) \neq \perp$ and $t_\perp(u) = \boxtimes$ otherwise.*

Proof. Let $\mathcal{S} = (\Sigma, \mathcal{N}, \mathcal{R}, I)$ and let $@ : o \to o$ be a fresh terminal symbol of arity 1. Define $\mathcal{S}_@ = (\Sigma \cup \{@\}, \mathcal{N}, \mathcal{R}_@, I)$ where $\mathcal{R}_@$ is the set of production rules $\{Fx_1 \cdots x_n \to @\, m \mid Fx_1 \cdots x_n \to m$ belongs to $\mathcal{R}\}$, *i.e.* we append a symbol @ whenever performing a rewriting step. Denote by $t^@$ the tree generated by $\mathcal{S}_@$.

It is fairly simple to notice the following:

- $t^@$ does not contain any node labelled $\perp$;
- t is obtained from $t^@$ by
  (1) replacing any infinite piece of branch (possibly starting from another node than the root) made only of nodes labelled by @ by a single node labelled by $\perp$;
  (2) contracting any finite path made of nodes labelled by @.

Now consider an MSO formula φ stating that a node is labelled by @, is the source of an infinite sequence of nodes labelled by @ and is the child of a node not labelled by @ (*i.e.* the node is the first one in an infinite piece of branch labelled by @). Thanks to Theorem 7.6, we can build a new recursion scheme $\mathcal{S}_{@,\varphi}$ that generates $t^@_\varphi$.

Now one obtains $\mathcal{S}_\perp$ by doing the following modification from $\mathcal{S}_{@,\varphi}$:

(1) in any production rule, replace any occurrence of a ground subterm of the form $\underline{@}\, s$ by the ground term $\boxtimes$;
(2) in any production rule, replace any occurrence of a ground subterm of the form $@\, s$ by the ground term s.

Hence, the tree $t_\perp$ produced by $\mathcal{S}_\perp$ is obtained from $t^@$ by (1) replacing any infinite branch made of nodes labelled by @ by a single node labelled by $\boxtimes$ and (2) by contracting any finite path made of nodes labelled by @. Therefore $\mathcal{S}_\perp$ is as expected. □

8.2 An à la Caucal Result for General Schemes

A natural extension of the MSO reflection is to use MSO to define new edges in the structure and not simply to mark certain nodes. This corresponds to the well-known mechanism of MSO interpretations [17]. Furthermore to obtain back a tree from this new graph, we choose a vertex as a root and we unfold the graph from it. As both MSO interpretation and unfolding are graph transformations that preserve the decidability of MSO, combining these two transformations provides a very powerful mechanism for constructing infinite trees with a decidable MSO model-checking problem.

Remark 8.2. If we only use MSO interpretations followed by unfolding to produce trees (and graphs) starting from the class of finite trees, we obtain the Caucal hierarchy [15]. The trees in this hierarchy are exactly the trees generated by *safe* recursion schemes.

8.2.1 Main Result. We first present a definition of MSO interpretations which is tailored to our setting. An ***MSO interpretation*** $\mathcal{I}$ over Σ-labelled ranked trees is given by a domain formula $\varphi_\delta(x)$, a formula $\varphi_a(x)$ for each $a \in \Sigma$ and a formula $\varphi_d(x, y)$ for each direction $d \in \{1, \dots, m\}$ where $m = \max\{ar(a) \mid a \in \Sigma\}$.

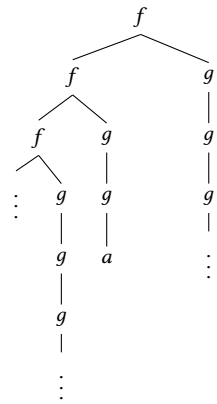
When applied to a Σ-labelled ranked tree t, an MSO interpretation $\mathcal{I}$ produces a graph, denoted $\mathcal{I}(t)$, whose vertices are the vertices of t satisfying $\varphi_\delta(x)$. A vertex u of $\mathcal{I}(t)$ is labelled by a iff u satisfies $\varphi_a(x)$ in t. Similarly there exists an edge labelled by $d \in \{1, \dots, m\}$ from a vertex u to a vertex v iff $(t, u, v) \models \varphi_d(x, y)$.

We say that the interpretation $\mathcal{I}$ is *well-formed* if for all Σ-labelled trees t, every vertex u of $\mathcal{I}(t)$ is labelled by exactly one $a \in \Sigma$ and has exactly one out-going edge for each direction in $\{1, \dots ar(a)\}$. Here, we restrict our attention to well-formed interpretations, which ensures that, when selecting a root in that graph, the generated tree (by unfolding from the root as explained in Section 8.2) is a Σ-labelled ranked tree. Given an MSO interpretation $\mathcal{I}$, one can decide if it is well-formed. Indeed, the fact that the graph obtained after applying $\mathcal{I}$ is well-formed can be expressed by a first-order formula ψ hence, it follows that there exists an MSO formula ψ^* such that for all tree t, $I(t) \models \psi$ if and only if $t \models \psi^*$ (see for instance [17]). Using Theorem 2.1, we can construct a parity tree automaton $\mathcal{A}_{\neg\psi^*}$ accepting the trees that do not satisfy ψ^* and, as emptiness is decidable for parity tree automata, the decidability follows.

Example 8.3. We revisit examples 7.1 and 7.5. Consider the scheme of Example 7.1 and the formula φ of Example 7.5 (recall that φ holds in a node u iff u is the right son of an f-labelled node and there is a path from u to an a-labelled node which contains an odd number of occurrences of g-labelled nodes). MSO reflection consisted in "marking" the nodes where φ holds.

Consider the MSO interpretation $\mathcal{I}$ which removes all nodes below a node where φ holds. All node labels are preserved. Finally all edges are preserved and a loop labelled by g is added to every node where φ holds. It is easily seen that $\mathcal{I}$ is a well-formed interpretation. By applying $\mathcal{I}$ to the tree t of example 7.1 and then unfolding it from its root, we obtain the tree on the right which is generated by the scheme on the left.

$$\begin{cases} Z & \to & \underline{F}\, g\, (g\, a) \\ G & \to & g\, G \\ \underline{F}\, \zeta\, x & \to & f\, (F\, g\, (\zeta\, x))\, G \\ F\, \zeta\, x & \to & f\, (\underline{F}\, g\, (\zeta\, x))\, x \end{cases}$$



More generally, we have the following result (whose proof is given in the next section).

Theorem 8.4. *Let t be a Σ-labelled ranked tree generated by an order-n recursion scheme and let $\mathcal{I}$ be a well-formed MSO interpretation. For any vertex r in $\mathcal{I}(t)$, the tree generated by $\mathcal{I}(t)$ from the root r can be generated by an order-$(n+1)$ recursion scheme.*

Remark 8.5. A natural question is whether every tree generated by order-$(n+1)$ recursion scheme can be obtained by unfolding a well-formed MSO interpretation of a tree generated by an order-n recursion scheme. This is for instance true when considering the subfamily of safe recursion schemes [15, 34]. The answer is negative. Indeed, a positive answer for possibly unsafe schemes would imply that safe schemes of any given order are as expressive (for generating trees) as unsafe ones of the same level, contradicting a result of Parys stating that unsafe schemes are strictly more expressive than safe schemes for generating trees [39].

8.2.2 Proof of Theorem 8.4. Let t be a Σ-labelled tree given by some order-n recursion scheme $\mathcal{S}$ and let $\mathcal{I}$ be a well-formed MSO interpretation given by formulas $\varphi_\delta(x)$, $\varphi_a(x)$ for each $a \in \Sigma$ and $\varphi_\ell(x,y)$ for each direction $\ell \in \{1, \dots m\}$ where $m = \max\{ar(a) \mid a \in \Sigma\}$. Let r be a vertex in $\mathcal{I}(t)$ and let t' be the tree generated by $\mathcal{I}(t)$ from the root r. We want to show that t' is generated by some order-$(n+1)$ scheme. By Theorem 5.1, it is enough to show that $\mathcal{I}(t)$ restricted to the vertices reachable from r is isomorphic to the $\mathfrak{e}$-closure[2] of the transition graph of some order-$(n+1)$ CPDA restricted to its reachable configurations.

The proof starts by annotating t with MSO definable sets, leading to a tree $\bar{t}$ that, thanks to Theorem 7.6, can be generated by a recursion scheme. We then show that tree-walking automata (a class of finite memory device that walk around an input tree) can be used to accept pairs of nodes that are connected by a new edge after applying $\mathcal{I}$ to t. Finally, we build the announced $(n+1)$-CPDA $\mathcal{A}$ by mimicking an n-CPDA generating $\bar{t}$ and simulating the previous tree-walking automata.

Notations. For all Σ-labelled trees t, all nodes u and v of t, we let $t_{u,v}$ be the tree obtained from t by marking the pair (u,v). Formally, $t_{u,v}$ is the $(\Sigma \times 2^{\{1,2\}})$-labelled tree such that $\mathsf{Dom}(t_{u,v}) = \mathsf{Dom}(t)$ and for $w \in \mathsf{Dom}(t_{u,v})$, $t_{u,v}(w) = (t(w), X)$ where $1 \in X$ iff $w = u$ and $2 \in X$ iff $w = v$.

Similarly we define $t_{u,\bullet}$ (*resp.* $t_{\bullet,v}$, *resp.* $t_{\bullet,\bullet}$), for a fresh symbol $\bullet$, as the tree obtained by marking u by 1 (*resp.* v by 2, *resp.* marking no node). *I.e.* $\bullet$ means here that no node is marked with the corresponding index (1 and/or 2).

[2]Formally, this means that any sequence of transitions $v_0 \xrightarrow{a} v_1 \xrightarrow{\mathfrak{e}} v_2 \xrightarrow{\mathfrak{e}} \cdots \xrightarrow{\mathfrak{e}} v_{k-1} \xrightarrow{\mathfrak{e}} v_k$ such that v_k is not the source of an $\mathfrak{e}$-transition is replaced by the transition $v_0 \xrightarrow{a} v_k$.

Using Theorem 2.1, one can construct for any formula $\varphi(x_1, x_2)$ a parity tree automaton $\mathcal{B}_\varphi$ that accepts $t_{u,v}$ iff $(t, u, v) \models \varphi(x_1, x_2)$. For all $\ell \in \{1, \ldots, m\}$, we write $\mathcal{B}_\ell$ for the parity tree automaton corresponding to the formula $\varphi_\ell(x, y)$ of $\mathcal{I}$ and we let Q_ℓ be its finite set of control states and Δ_ℓ be its transition relation.

Annotation of t by MSO definable sets. The first step of the construction is to annotate t with information concerning essentially the behaviour of the automata $\mathcal{B}_\ell$ on the subtrees of t. The resulting annotated version of t is denoted $\bar{t}$. More precisely, the annotated tree $\bar{t}$ has, for each node u such that $(t, u) \models \varphi_\delta(x)$, the following finite information:

- the unique $a \in \Sigma$ such that $(t, u) \models \varphi_a(x)$. Unicity is by definition of a well-formed interpretation.
- $d_\uparrow \in \{1, \ldots, m\} \cup \{\mathfrak{r}\}$ which is the direction from the father of the curent node to the current node, *i.e.* u is of the form $u'd_\uparrow$ for some $u' \in \mathsf{Dom}(t)$, and $\mathfrak{r}$ if the current node is the root.
- for each $\ell \in \{1, \ldots, m\}$, we have:
  - $i_\ell \in \{\uparrow, \downarrow, \circlearrowleft, \bot\}$ such that
    - $i_\ell = \bot$ iff there is no node v such that $(t, u, v) \models \varphi_\ell(x, y)$,
    - $i_\ell = \downarrow$ iff there is a unique node v such that $(t, u, v) \models \varphi_\ell(x, y)$ and v is below u,
    - $i_\ell = \uparrow$ iff there is a unique node v such that $(t, u, v) \models \varphi_\ell(x, y)$ and v is not below u,
    - $i_\ell = \circlearrowleft$ iff $(t, u, u) \models \varphi_\ell(x, y)$.
  - the set R_ℓ of states $q \in Q_\ell$ such that there exists an accepting run of $\mathcal{B}_\ell$ starting from q on the subtree of t rooted at u.
  - the set S_ℓ of pairs $(d, q) \in \{1, \ldots, m\} \times Q_\ell$ such that there exists an accepting run of $\mathcal{B}_\ell$ starting from q on the subtree of $t_{\bullet,\bullet}$ rooted at ud,
  - the set T_ℓ of pairs $(d, q) \in \{1, \ldots, m\} \times Q_\ell$ such that there exists a node v below ud such that $\mathcal{B}_\ell$ has an accepting run starting from ud on the subtree of $t_{\bullet,v}$ rooted at ud.

Let Σ' be the resulting labelling alphabet of $\bar{t}$. As the information annotated on $\bar{t}$ is MSO definable in t, we know from Theorem 7.6 that $\bar{t}$ is generated by some order-n recursion scheme.

Replacing MSO formulas on t by tree-walking automata working on $\bar{t}$. Fix a direction $\ell \in \{1, \ldots, m\}$. Thanks to the extra information available on $\bar{t}$, it is possible to decide if a pair of nodes (u, v) satisfies the formula $\varphi_\ell(x, y)$ on t using a deterministic tree-walking automaton running on $\bar{t}$. Intuitively a tree-walking automaton is a finite memory device that walks around an input tree, choosing what move to make according to its current state and the node label.

Formally, a ***deterministic tree-walking automaton*** (introduced in the early seventies by Aho and Ullman [3]; see also [21]) working on Σ-labelled trees is a tuple $\mathcal{W} = (Q, q_0, F, \delta)$ where Q is a finite set of states, $q_0 \in Q$ is the initial state, F is a set of final states and δ is a transition function. The transition function associates to a pair $(p, a) \in Q \times \Sigma$, corresponding respectively to the current state and node label, a pair $(q, x) \in Q \times (\{\uparrow, -\} \cup \{1, \ldots, m\})$ where q is the new state and x is a movement to perform. Intuitively $-$ corresponds to "stay in the current node", $\uparrow$ to "go to the parent node" and $d \in \{1, \ldots, m\}$ corresponds to "go to the d-child". A pair of nodes (u, v) is *accepted* by $\mathcal{W}$ if it can reach v in a final state starting from u in the initial state.

We claim that there exists a deterministic tree-walking automaton $\mathcal{W}_\ell$ such that, for any pair (u, v) of nodes of t, we have:

$$\mathcal{W}_\ell \text{ accepts } (u, v) \text{ in } \bar{t} \text{ iff } (t, u, v) \models \varphi_\ell(x, y).$$

The automaton $\mathcal{W}_\ell$ works in two phases: during the first phase the automaton only goes up in the tree and during the second phase it only goes down in the tree. Both phases can potentially be empty. In fact, to accept a pair (u, v) the automaton will first go up to the greatest common ancestor of u and v and down to v.

Assume that $\mathcal{W}_\ell$ started at a node u and denote by v the unique node (if it exists) such that $(t, u, v) \models \varphi_\ell(x, y)$.

Initialisation. The automaton is in its initial state q_0 at node u that is labelled by the tuple $(a, d_\uparrow, (i_k, R_k, S_k, T_k)_{k\in\{1,\dots,m\}})$. The automaton checks in which of the following four cases it is:

- The node v does not exists (*i.e.* $i_\ell = \perp$). No transition is defined.
- The node v is equal to u (*i.e.* $i_\ell = \circlearrowleft$). The automaton goes to the accepting state.
- The node v is not below u (*i.e.* $i_\ell = \uparrow$). The automaton begins the first phase while memorising the set X of states $q \in Q_\ell$ such that $\mathcal{B}_\ell$ admits an accepting run starting from q on the subtree of $t_{u,\bullet}$ rooted at u. This set can easily be computed from Δ_ℓ and S_ℓ.
- The node v is below u (*i.e.* $i_\ell = \downarrow$). The automaton begins the second phase. It computes the unique direction d and the set Y of states $q \in Q_\ell$ such that:
  - $\mathcal{B}_\ell$ admits an accepting run on the tree $t_{u,\bullet}$ deprived of the nodes below ud and assigning state q to ud,
  - $(d, q) \in T_\ell$.

First phase. The automaton is at some node w and stores the set X of states $q \in Q_\ell$ such that $\mathcal{B}_\ell$ admits an accepting run starting from q on the subtree of $t_{u,\bullet}$ rooted at w. The label of the node w is $(a, d_\uparrow, (i_k, R_k, S_k, T_k)_{k\in\{1,\dots,m\}})$. The automaton goes up in the tree (while remembering $d_\uparrow$ and X) to a node w', the father of w, whose label is $(a', d'_\uparrow, (i'_k, R'_k, S'_k, T'_k)_{k\in\{1,\dots,m\}})$.

There are now three possible cases:

- The node v is the current node. This is the case iff there exists a state $q \in R'_\ell$ and a transition in Δ_ℓ starting in state q with $(a, \{2\})$ as label and associating state q_i to the i-child, such that:
  - $q_{d_\uparrow}$ belongs to X ,
  - for all $i \neq d_\uparrow$, $(i, q_i) \in S_\ell$.

  In this case, the automaton goes to the accepting state.
- The node v is below the j-child of w' for some $j \in \{1, \dots, ar(a')\}$. This is the case iff there exists a state $q \in R'_\ell$, a transition in Δ_ℓ starting in state q with $(a', \emptyset)$ as label and associating state q_i to the i-child, and such that there exists $j \neq d_\uparrow \in \{1, \dots, ar(a')\}$ with
  - $q_{d_\uparrow}$ belongs to X,
  - for all $i \neq j, d_\uparrow$ one has $(i, q_i) \in S'_\ell$,
  - $(j, q_j) \in T'_\ell$.

  In this case, the automaton begins the second phase while memorising the set Y of all states q_j matching this definition together with the direction [3] j.
- The node v is not below w'. This is the case when the two previous cases do not apply. The automaton updates X using $d_\uparrow$, Δ_ℓ and the former value of X and goes on in the first phase.

The second phase The automaton is at some node w and stores a direction d and the set Y of states $q \in Q_\ell$ such that:

- $\mathcal{B}_\ell$ admits an accepting run on the tree $t_{u,\bullet}$ deprived of the nodes below wd and assigning state q to wd
- there exists a node v below wd such that $\mathcal{B}_\ell$ admits an accepting run starting in state q on the subtree of $t_{\bullet,v}$ rooted at wd.

The automaton goes down in direction d (while remembering Y) to a node $w' = wd$ whose label is $(a', d, (i'_k, R'_k, S'_k, T'_k)_{k\in\{1,\dots,m\}})$.

There are now two cases:

[3]Due to the restriction imposed on $\varphi_\ell(x, y)$ by the fact that $\mathcal{I}$ is a well-formed MSO interpretation, there cannot be two different directions. Otherwise, we would have $v_1 \neq v_2$ such that $(t, u, v_1) \models \varphi(x, y)$ and $(t, u, v_2) \models \varphi(x, y)$.

- The node v is below the d'-child of w' for some $d' \in \{1, \dots, ar(a')\}$. This is the case iff there exists a state $q \in Y$ and a transition in Δ_ℓ starting in state q with $(a, \emptyset)$ as label and associating states q_i to the i-child such that
  - for all $i \neq d'$, $(i, q_i) \in S'_\ell$,
  - $(d', q_{d'}) \in T'_\ell$.

  In this case, the automaton updates the set Y with all states $q_{d'}$ matching this condition, stores the direction d' and goes on in the second phase.
- The node v is the current node. This is precisely when the previous case does not hold. The automaton moves to an accepting state.

Construction of the order-$(n+1)$ CPDA $\mathcal{A}$ generating t'. By Theorem 5.1, there exists an order-n CPDA $C = (\{1, \dots, m\} \cup \{\mathfrak{e}\}, \Gamma, Q, \delta, q_\iota, F)$, and a mapping $\rho : Q \to \Sigma'$ such that $\bar{t}$ is the tree generated by C and ρ.

Hence, for every node $u = d_1 \cdots d_k \in \mathrm{Dom}(t)$, there exists a unique sequence of configurations $(q_0, s_0), \dots, (q_k, s_k)$ of C such that:

- there exists a path in $\mathrm{Graph}(C)$ labelled by $\mathfrak{e}^*$ from the initial configuration to (q_0, s_0),
- for all $i \in \{0, \dots, k\}$, (q_i, s_i) is not the source of an $\mathfrak{e}$-labelled arc in $\mathrm{Graph}(C)$,
- for all $i \in \{0, \dots, m-1\}$, there exists a path labelled by a word in $d_{i+1}\mathfrak{e}^*$ from (q_i, s_i) to (q_{i+1}, s_{i+1}) in $\mathrm{Graph}(C)$.

Such a sequence can be encoded as an order-$(n+1)$ stack s_u using symbols from $\Gamma \cup A$ (recall that the s_i are order-n stacks) in the following way:

$$s_u = [\tilde{s_0}, \tilde{s_1}, \dots, \tilde{s_{k-1}}, push_1^{q_k,1}(s_k)]$$

where for all $i \in \{0, \dots, k-1\}$, $\tilde{s_i} = push_1^{d_{i+1},1}(push_1^{q_i,1}(s_i))$.

The automaton $\mathcal{A}$ works on stacks corresponding to some s_u for some $u \in \mathrm{Dom}(t)$. Its set of control states contains a distinguished state $q_\star$ and all states of the tree-walking automata $(\mathcal{W}_\ell)_{\ell \in \{1,\dots,m\}}$ which we assumed to be disjoint. The configurations of $\mathcal{A}$ that are source of non-$\mathfrak{e}$-labelled arcs will be of the form $(q_\star, s_u)$ for some $u \in \mathrm{Dom}(t)$. The intended behaviour of $\mathcal{A}$ is that, for some $\ell \in \{1, \dots, m\}$, if $(t, u, v) \models \varphi_\ell(x, y)$ then $\mathcal{A}$ can go from the configuration $(q_\star, s_u)$ to the configuration $(q_\star, s_v)$ by a path labelled by $\ell\mathfrak{e}^*$.

First $\mathcal{A}$ moves by an ℓ-labelled transition to the configuration (q_0^ℓ, s_u) where q_0^ℓ is the initial state of the tree-walking automaton $\mathcal{W}_\ell$. Recall that the information about the location of the vertex v is given by the annotations in $\rho(top_1(s_u))$.

In a configuration of the form (p, s_u) with p a state of $\mathcal{W}_\ell$, $\mathcal{A}$ simulates the behaviour of $\mathcal{W}_\ell$ on $\bar{t}$ at node u in state p by a sequence of $\mathfrak{e}$-transitions. As $\bar{t}(u) = \rho(top_1(s_u))$, $\mathcal{A}$ can compute the transition taken by the automaton $\mathcal{W}_\ell$ on $\bar{t}$ at node u in state p. The behaviour of $\mathcal{A}$ will be such that if $\mathcal{W}_\ell$ goes from (p, u) to (q, u') in one step then $\mathcal{A}$ will go through a sequence of $\mathfrak{e}$-transitions from (p, s_u) to $(q, s_{u'})$.

We distinguish several cases depending on the movement performed by $\mathcal{W}_\ell$.

- $\mathcal{W}_\ell$ stays in the current node in state q. Then $\mathcal{A}$ changes its state to q by an $\mathfrak{e}$-transition.
- $\mathcal{W}_\ell$ goes to its parent node in state q (*i.e.* $u = u'd$ and $\mathcal{W}_\ell$ ends up in u' in state q). Then $\mathcal{A}$ performs pop_{n+1} followed by a pop_1 and moves to state q. The stack content of $\mathcal{A}$ is now $pop_1(pop_{n+1}(s_u)) = s_{u'}$ hence $\mathcal{A}$ is in configuration $(q, s_{u'})$.
- $\mathcal{W}_\ell$ goes to its d-child in state q (*i.e.* $u' = ud$ and $\mathcal{W}_\ell$ ends up in ud in state q). Assume that s_u is equal to:

$$[\tilde{s_0}, \tilde{s_1}, \dots, \tilde{s_{k-1}}, push_1^{q_k,1}(s_k)]$$

and that s_{ud} is of the from:

$$[\tilde{s_0}, \tilde{s_1}, \ldots, push_1^{d,1}(push_1^{q_k,1}(s_k)), push_1^{q_{k+1},1}(s_{k+1})]$$

As C generates $\bar{t}$, there exists a path π in Graph(C) from (q_k, s_k) to (q_{k+1}, s_{k+1}) labelled by $d\mathfrak{e}^*$.

Then $\mathcal{A}$ starts by performing a $push_1^{d,1}$ followed by $push_{n+1}$, pop_1 and pop_1. At this point the stack is:

$$[\tilde{s_0}, \tilde{s_1}, \ldots, push_1^{d,1}(push_1^{q_k,1}(s_k)), s_k].$$

Then $\mathcal{A}$ simulates the order-n operations of C along the path π using $\mathfrak{e}$-transitions. When no $\mathfrak{e}$-transition of C can be applied, $\mathcal{A}$ performs a last $\mathfrak{e}$-transition where it performs a $push_1^{q_{k+1},1}$ and goes to state q_{k+1} hence, reaching configuration (q, s_{ud}).

Eventually $\mathcal{A}$ will reach a configuration of the form (q_f^ℓ, s_v) where q_f^ℓ is the accepting state of $\mathcal{W}_\ell$. It then goes to the state $q_\star$.

From its initial configuration, $\mathcal{A}$ deterministically builds the stack s_{u_0} (which correspond to the vertex u_0 from which $\mathcal{I}(t)$ is unfolded) by using sequence of $\mathfrak{e}$-transitions and goes to the state $q_\star$.

By construction, we have that the $\mathfrak{e}$-closure of $\mathcal{A}$ restricted to the vertices reachable from its initial configuration is isomorphic to $\mathcal{I}(t)$ restricted to the vertices reachable from u_0. The isomorphism simply maps a configuration $(q_\star, s_u)$ of $\mathcal{A}$ to $u \in \mathsf{Dom}(t)$.

To generate t' it remains to retrieve the node label, *i.e.* define a function from the set of control states of $\mathcal{A}$ into Σ. As defined so far $\mathcal{A}$ does not have the necessary information for that: indeed, relevant configurations all have $q_\star$ as their control state. But it is not hard to replace $q_\star$ by a variant in $\{q_\star^a \mid a \in \Sigma\}$: instead of going to $(q_\star, s_u)$, $\mathcal{A}$ goes to $(q_\star^a, s_u)$ where a is the first component labelling u in $\bar{t}$ which can easily be recover by $\mathcal{A}$ (it suffices to do a pop_1 operation and the information is then in the topmost symbol). Then, if ρ is defined by $\rho(q_\star^a) = a$, the tree t' is generated by $\mathcal{A}$ and ρ, which concludes the proof.

9 SELECTION

We now focus on a more general problem than global model-checking, often known as the *synthesis* problem. For simplicity we start by a case study motivated by the famous question of the definability of choice functions on the infinite binary tree [25].

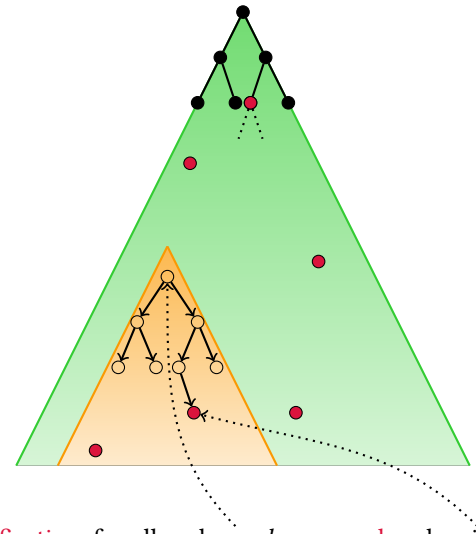
9.1 A Case Study: Choice Functions

Consider an infinite binary tree t in which some nodes are coloured (*i.e.* labelled) in red; we also assume that the domain of t is included in $\{1, 2\}^*$ and we identify direction 1 with the left and direction 2 with the right. Assume that t satisfies the following extra property (we say that t is *well-formed*): every subtree contains at least one red node. Our goal is for all nodes x to *choose/select* a red node y in the subtree $t[x]$ rooted at x (see Figure 6 for an illustration).

A ***choice function*** is a function that with any node x associates such a y. An ***MSO choice function*** for t is a formula describing a choice function, *i.e.* a formula $\varphi(x, y)$ with two first-order free variables such that

$$\forall x \in t \;\; \exists!\, y \text{ s.t. } y \text{ is red}, x < y \text{ and } \varphi(x, y)$$

Guverich and Shelah proved that there does not exist an MSO formula $\varphi(x, X)$ such that for all non-empty set of nodes U of the infinite complete binary tree t_2, $(t_2, u, U) \models \varphi(x, X)$ for exactly one node u in U. In other terms, there is no MSO formula selecting (on the infinite complete binary tree) exactly one element for each non-empty subset. This result was later re-proven in a more elementary way by Carayol and Löding. Moreover they also exhibited in [10, Theorem 6] a tree



Specification: for all nodes x *choose* a red node y in $t[x]$

Fig. 6. Choice function

generated by a (safe) recursion scheme on which finding an MSO choice function (in the sense of the present article) fails.

Theorem 9.1. *[10, 25][9, Proposition 4.3] There is a well-formed tree generated by an order-3 (safe) recursion scheme for which no MSO choice function exists.*

Instead of using an MSO formula to describe a choice function, one can do the following (see Figure 7 for an illustration). Consider a partition $X_1 \uplus X_2$ of the set of nodes of t, and think of the nodes in X_1 (*resp.* X_2) as those where one should first go down to the left (*resp.* right) in order to find a red node. A partition (X_1, X_2) describes a choice function iff the following holds. For any node $u \in t$, define the sequence $u_0, u_1, u_2, \ldots$ by letting $u_0 = u$ and $u_{i+1} = u_i d_i$ where $d_i = 1$ if $u_i \in X_1$ and $d_i = 2$ if $u_i \in X_2$: *i.e.* $u_0, u_1, u_2, \ldots$ is the sequence of nodes visited starting from u and following the directions indicated by $X_1 \uplus X_2$. Then for some k, u_k is red.

Remark 9.2. Note that a partition $X_1 \uplus X_2$ always exists on a well-formed tree. Indeed, for every node u, consider the minimal depth of a red node in the left subtree and the minimal depth of a red node in the right subtree: if the smallest depth is in the left subtree one lets $u \in X_1$ otherwise one lets $u \in X_2$.

It directly follows from Theorem 9.1 that a partition defining a choice function cannot be captured by an MSO formula.

Corollary 9.3. *There is a well-formed tree generated by an order-3 (safe) recursion scheme such that, for all MSO formula $\varphi(x)$, the sets $X_1 = \{u \mid \varphi(x) \text{ holds in } u\}$ and $X_2 = \{u \mid \varphi(x) \text{ does not hold in } u\}$ do not define a choice function on t.*

In the same spirit as for global model-checking we propose an exogeneous and an endogeneous approach to the previous problem.

- *Exogenous* approach: Given a Σ-labelled well-formed tree $t : \mathrm{Dom}(t) \to \Sigma$, output a description by means of a word acceptor device of a subset $X_1 \subseteq \mathrm{Dom}(t)$ of nodes such that $(X_1, \mathrm{Dom}(t) \setminus X_1)$ defines a choice function for t.

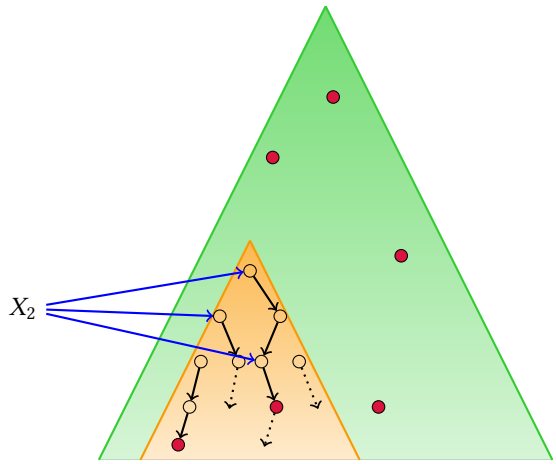


Fig. 7. Partition defining a choice function

- *Endogenous* approach: Given a Σ-labelled well-formed tree $t : \mathsf{Dom}(t) \longrightarrow \Sigma$, output a finite description of a $(\Sigma \times \{1,2\})$-labelled tree $t_{ch} : \mathsf{Dom}(t) \longrightarrow \Sigma \times \{1,2\}$ such that t_{ch} and t have the same domain, and such that $X_1 = \{u \mid u \in \mathsf{Dom}(t) \text{ and } t_{\mathrm{ch}}(u) = (a,1) \text{ for some } a \in \Sigma\}$ and $X_2 = \{u \mid u \in \mathsf{Dom}(t) \text{ and } t_{ch}(u) = (a,2) \text{ for some } a \in \Sigma\}$ define a choice function for t.

Contrasting with the impossibility result stated in Corollary 9.3 we have the following result that follows from a more general result (see Theorem 9.12 below).

Corollary 9.4. *Let t be a well-formed tree generated by an order-n recursion scheme $\mathcal{S}$.*

(1) *There is an algorithm that builds from $\mathcal{S}$ an order-n CPDA $\mathcal{A}$ such that $(L(\mathcal{A}), \mathsf{Dom}(t) \setminus L(\mathcal{A}))$ defines a choice function for t.*
(2) *There is an algorithm that builds from $\mathcal{S}$ an order-n recursion scheme $\mathcal{S}_{\mathrm{ch}}$ that generates a tree t_{ch} defining a choice function for t.*

Remark 9.5. As for the reflection property we note that, in the previous statement, item (2) implies item (1).

The corollaries 9.3 and 9.4 might seem, at first sight, contradictory. However they concern two orthogonal representations of choice functions: via MSO definability and via CPDA respectively. Applying Corollary 9.4 to the tree generated by a recursion scheme of Corollary 9.3 leads to a new recursion scheme of the same order but enriched with additional information.

9.2 Effective Selection Property

We now introduce the effective selection property and first present an exogenous approach.

Definition 9.6. (MSO selection problem: exogeneous approach) Let $\varphi(X_1, \cdots, X_\ell)$ be an MSO formula with ℓ second-order free variables, and let $t : \mathsf{Dom}(t) \longrightarrow \Sigma$ be a Σ-labelled ranked tree. The ***MSO selection problem*** is to decide whether the formula $\exists X_1 \ldots \exists X_\ell\, \varphi(X_1, \cdots, X_\ell)$ holds in t, and in this case to output a description, by means of word acceptor devices, of ℓ sets $U_1, \cdots, U_l \subseteq \mathsf{Dom}(t)$ such that $(t, U_1, \ldots, U_\ell) \models \varphi(X_1, \ldots, X_\ell)$.

We now give the endogenous approach. The idea is to describe subsets of nodes $U_1, \cdots U_\ell$, such that $(t, U_1, \ldots, U_\ell) \models \varphi(X_1, \ldots, X_\ell)$, by marking every node with a tuple of ℓ Booleans (a node u

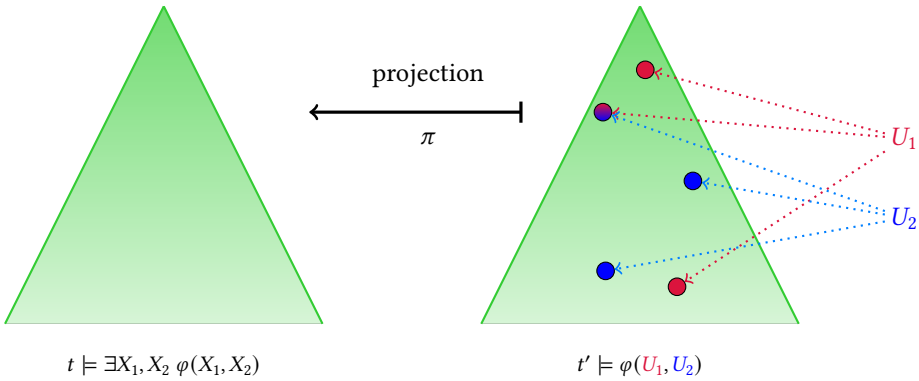


Fig. 8. The MSO selection problem

belongs to U_i iff the i-th Boolean is 1 in the tuple labelling u). See Figure 8 for an illustration of the following definition when $\ell = 2$.

Definition 9.7. (MSO selection problem: endogeneous approach) Let $\varphi(X_1, \cdots, X_\ell)$ be an MSO formula with ℓ second-order free variables, and let $t : \mathsf{Dom}(t) \to \Sigma$ be a Σ-labelled ranked tree. Call Ξ the ranked alphabet $\Sigma \times \{0,1\}^\ell$ defined by letting $ar((a, b_1, \ldots, b_\ell)) = ar(a)$. The ***MSO selection problem*** is to decide whether the formula $\exists X_1 \ldots \exists X_\ell\ \varphi(X_1, \cdots, X_\ell)$ holds in t, and if so to output a Ξ-labelled ranked tree $t_\varphi : \mathsf{Dom}(t_\varphi) \to \Xi$ such that the following holds:

(1) The trees t and t_φ have the same domain and for every node u, one has $t(u) = \pi(t_\varphi(u))$ where π is the projection from Ξ to Σ defined by $\pi((a, b_1, \ldots, b_\ell)) = a$.
(2) Defining, for every $1 \leq i \leq \ell$,

$$U_i = \{u \in \mathsf{Dom}(t) \mid t_\varphi(u) \text{ is of the form } (a, b_1, \ldots, b_{i-1}, 1, b_{i+1}, \ldots, b_\ell)\},$$

one has $(t, U_1, \ldots, U_\ell) \models \varphi(X_1, \ldots, X_\ell)$.

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

Example 9.8. (Local model-checking) Obviously the MSO selection problem captures the MSO model-checking problem. Indeed, it suffices to take $\ell = 0$ (*i.e.* there is no second-order free variable).

Example 9.9. (Global model-checking) The MSO selection problem captures the MSO global model-checking problem. Indeed, consider a tree t and an MSO formula $\varphi(x)$ with a first-order free variable and recall that the global model-checking asks for a description (endogenous or exogenous) of the set $[[\varphi]]_t = \{u \in \mathsf{Dom}(t) \mid (t, u) \models \varphi(x)\}$. Now let $\psi(X) = x \in X \Leftrightarrow \varphi(x)$ Then $\exists X\ \psi(X)$ is always true and there is a unique U such that $(t, U) \models \psi(X)$, and this U is equal to $[[\varphi]]_t$. Hence, an answer to the MSO selection problem for $\psi(X)$ on t leads to a solution to the global model-checking problem for φ on t.

Example 9.10. (Choice function) We now explain how to use the MSO selection problem to obtain a partition defining a choice function on a well-formed tree t. Consider an MSO formula $\varphi(X_1, X_2)$ that expresses the following

- X_1 and X_2 form a partition of the nodes of t.

- For all node x, there exists a red node z below x and a (finite) subset U of nodes that forms a path from x to z, and moreover for all $y \in U$ that is different from z the left-successor (*resp.* right-successor) of y belongs to U iff $y \in X_1$ (*resp.* $y \in X_2$).

One easily verifies that a solution (U_1, U_2) of the selection problem for $\varphi(X_1, X_2)$ on t provides a partition defining a choice function on t.

We now define the effective MSO selection property that characterises the classes of generators of trees for which an endogenous approach of the selection problem can be performed.

Definition 9.11. (Effective MSO selection property) Let $\mathcal{R}$ be a class of generators of trees. We say that $\mathcal{R}$ has the ***effective MSO selection property*** if there is an algorithm that transforms any pair $(R, \varphi(X_1, \ldots, X_\ell))$ with $R \in \mathcal{R}$ into some $R_\varphi \in \mathcal{R}$ (if exists) such that the tree generated by R_φ is a selector for φ in the tree generated by R.

Quite surprisingly (think of the negative result in Corollary 9.3), recursion schemes (equivalently CPDA) have the effective MSO selection property. The first proof [12] of this result (which is the one we give here) relies on the connection with the computation of a winning strategy in a CPDA parity game together with the fact that such a strategy can be defined by a CPDA that is *synchronised* with the one defining the arena. Alternative proofs were later given by Haddad in [27, 28] and by Grellois and Melliès in [23]. Both proofs are very different from the one we give here. Indeed, our proof uses the equi-expressivity theorem to restate the problem as a question on CPDA, and a drawback of this approach is that once the answer is given on the CPDA side one needs to go back to the scheme side, which is not complicated but yields a scheme (that in a sense has been normalised) that is very different from the original one. The advantage of the approaches in [28] (built on top of the intersection types approach by Kobayashi and Ong [37]) and in [23] (based on purely denotational arguments and connections with linear logic) is to work directly on the recursion scheme and to succeed to provide as a selector a scheme obtained from the original one by adding duplicated and annotated versions of the terminals.

Theorem 9.12. *Higher-order recursion schemes as well as collapsible pushdown automata have the effective MSO selection property.*

Proof. Thanks to Theorem 5.1, it suffices to prove the property for collapsible pushdown automata.

Let $\varphi(X_1, \cdots, X_\ell)$ be an MSO formula with ℓ second-order free variables, and let $\mathcal{A}$ be a collapsible pushdown automaton generating a tree t.

Thanks to the well-known equivalence between MSO logic and tree automata (see *e.g.* [45]), there is a tree automaton $\mathcal{B}_\varphi$ working on $\Sigma \times \{0, 1\}^\ell$ trees such that a tree t_φ is accepted by $\mathcal{B}_\varphi$ iff t_φ is a selector for φ in t.

Recall (see Section 2.9) that acceptance of a tree by a tree automaton can be seen as existence of a winning strategy in a parity game that is (informally) played as follows. The two players, Éloïse and Abelard move down the tree a pebble to which is attached a state of the automaton; the play starts at the root (with initial state attached to the pebble); at each round Éloïse provides a valid transition (w.r.t the current state and the current node label) of the automaton and Abelard moves the pebble to some child and updates the state attached to the pebble according to the transition chosen by Éloïse. In case the pebble reaches a leaf, the play loops forever and Éloïse wins iff the state and the node label are consistent with the acceptance condition on leaves; otherwise the play is infinite and Éloïse wins iff the smallest infinitely often visited priority is even.

For the tree t_φ, the underlying arena of the previous game is essentially a synchronised product of t_φ with the finite graph corresponding to $\mathcal{B}_\varphi$. Now consider a variant of this game where instead

of checking whether a given tree t is accepted by $\mathcal{B}_\varphi$ the players want to check, for a given tree t, whether there exists some t_φ such that t_φ is accepted by $\mathcal{B}_\varphi$ and t_φ is a marking of t. The game is essentially the same, except that now Éloïse is also giving the marking of the current vertex (*i.e.* π^{-1}). In this case, the game is a collapsible pushdown game (the arena being obtained as the synchronised product of t defined by a CPDA and the $\mathcal{B}_\varphi$ component together with one component where Éloïse is guessing the marking) and one directly checks that Éloïse wins from the root if and only if there is an annotation t_φ of t that is accepted by $\mathcal{B}_\varphi$, *i.e.* t_φ is a selector for φ in t. Call $\mathbb{G}$ this game and call $\mathcal{A}'$ the underlying CPDA.

Now, apply Theorem 5.5 to $\mathbb{G}$. Then either Éloïse has no winning strategy from the initial configuration and we are done (there is no selector). Otherwise one can effectively construct an n-CPDA $\mathcal{B}$ that is *synchronised* with $\mathcal{A}'$ and realises a well-defined *winning* strategy for Éloïse in $\mathbb{G}$ from the initial configuration. As $\mathcal{A}'$ and $\mathcal{B}$ are synchronised, we can consider their synchronised product; call it $\mathcal{A}''$. Hence, in $\mathcal{A}''$ the configurations contain extra informations (coming from $\mathcal{B}$); in particular, for any configuration, if the control state from the $\mathcal{A}'$-component is controlled by Éloïse, then the control state from the $\mathcal{B}$-component indicates the next move Éloïse should play: in particular, it provides a transition of the tree automaton, together with information regarding the marking. Transform $\mathcal{A}''$ by removing every transition that is not consistent with the strategy described by $\mathcal{B}$: then the tree generated by this new CPDA is isomorphic to some t_φ (that is a marking of t) together with an accepting run of $\mathcal{B}_\varphi$ on it. Now, if we forget the component from $\mathcal{B}_\varphi$ we obtain an n-CPDA $\mathcal{A}_\varphi$ that generates a selector t_φ for φ in t. □

Remark 9.13. A similar statement for *safe* schemes as well as for higher-order pushdown automata (*i.e.* collapsible pushdown automata that never perform a collapse) can be deduced from [8, 13, 22]. However, the machinery for general schemes is much more involved.

Remark 9.14 (Selection vs *Reflection).* In Example 9.9 we explained how one can reduce the MSO reflection to the MSO selection. In particular Theorem 9.12 directly implies MSO reflection for recursion schemes (Theorem 7.6).

One may wonder whether a converse reduction exists, *i.e.* whether one can transform any instance of the selection problem into (possibly several) instance(s) of the reflection problem.

Think of the simplest case where one deals with a single second-order free variable. If a reduction from selection to reflexion would exist it would mean that, given any formula $\varphi(X)$ and any recursion scheme $\mathcal{S}$ such that $\exists X\, \varphi(X)$ holds in the tree t generated by $\mathcal{S}$, then there exists another (possibly more complicated) formula $\psi(x)$ such that if one lets $U = [[\psi]]_t$ one has $(t, U) \models \varphi(X)$. Note here that we do not even ask for effectivity in constructing ψ from φ and that ψ may depend on both φ and $\mathcal{S}$.

Now remember that selection gives a framework to express the existence of a choice function (see Example 9.10; note by the way that one only needs a single free variable here as X_2 can safely be defined as the complement of X_1). Therefore, if one could reduce selection to reflection, it would mean that there would exist an MSO formula $\psi(x)$ that defines a choice function, which contradicts Corollary 9.3.

Hence, one can consider that the selection property is strictly more general than the reflection property.

REFERENCES

[1] Klaus Aehlig. 2006. A Finite Semantics of Simply-Typed Lambda Terms for Infinite Runs of Automata. In *"Proceedings of Computer Science Logic, 15th Annual Conference of the EACSL (CSL 2006)" (Lecture Notes in Computer Science)*, Vol. 4207. Springer-Verlag, 104–118.

[2] Klaus Aehlig, Jolie de Miranda, and C.-H. Luke Ong. 2005. Safety is not a Restriction at Level 2 for String Languages. In *Proceedings of the 8th International Conference on Foundations of Software Science and Computational Structures (FoSSaCS 2005) (Lecture Notes in Computer Science)*, Vol. 3411. Springer-Verlag, 490–501.

[3] Alfred V. Aho and Jeffrey D. Ullman. 1971. Translations on a Context-Free Grammar. *Information and Computation* 19, 5 (1971), 439–475.

[4] André Arnold and Damian Niwiński. 2001. *Rudiments of mu-Calculus*. Studies in Logic and the Foundations of Mathematics, Vol. 146. Elsevier.

[5] Christopher H. Broadbent, Arnaud Carayol, Matthew Hague, Andrzej S. Murawski C.-H. Luke Ong, and Olivier Serre. 2020. Collapsible Pushdown Games. (2020). https://www.irif.fr/~serre//PublisMisc/BCHMOS20.pdf

[6] Christopher H. Broadbent, Arnaud Carayol, C.-H. Luke Ong, and Olivier Serre. 2010. Recursion Schemes and Logical Reflexion. In *Proceedings of the 25th Annual IEEE Symposium on Logic in Computer Science (LiCS 2010)*. IEEE Computer Society, 120–129.

[7] Christopher H. Broadbent and C.-H. Luke Ong. 2009. On Global Model Checking Trees Generated by Higher-Order Recursion Schemes. In *Proceedings of the 12th International Conference on Foundations of Software Science and Computational Structures (FoSSaCS 2009) (Lecture Notes in Computer Science)*, Vol. 5504. Springer-Verlag, 107–121.

[8] Arnaud Carayol. 2006. *Automates infinis, logiques et langages*. Ph.D. Dissertation. Université de Rennes 1.

[9] Arnaud Carayol. 2019. *Automata, Logics and Games for Infinite Trees*. habilitation. Université Paris Est.

[10] Arnaud Carayol and Christof Löding. 2007. MSO on the Infinite Binary Tree: Choice and Order. In *"Proceedings of Computer Science Logic, 21st Annual Conference of the EACSL (CSL 2007)" (Lecture Notes in Computer Science)*, Vol. 4646. Springer-Verlag, 161–176.

[11] Arnaud Carayol, Antoine Meyer, Matthew Hague, C.-H. Luke Ong, and Olivier Serre. 2008. Winning Regions of Higher-Order Pushdown Games. In *Proceedings of the 23rd Annual IEEE Symposium on Logic in Computer Science (LiCS 2008)*. IEEE Computer Society, 193–204.

[12] Arnaud Carayol and Olivier Serre. 2012. Collapsible Pushdown Automata and Labeled Recursion Schemes: Equivalence, Safety and Effective Selection. In *Proceedings of the 27th Annual IEEE Symposium on Logic in Computer Science (LiCS 2012)*. IEEE Computer Society, 165–174.

[13] Arnaud Carayol and Michaela Slaats. 2008. Positional Strategies for Higher-Order Pushdown Parity Games. In *Proceedings of the 33rd Symposium, Mathematical Foundations of Computer Science (MFCS 2008) (Lecture Notes in Computer Science)*, Vol. 5162. Springer-Verlag, 217–228.

[14] Arnaud Carayol and Stefan Wöhrle. 2003. The Caucal Hierarchy of Infinite Graphs in Terms of Logic and Higher-Order Pushdown Automata. In *Proceedings of the 23rd International Conference on Foundations of Software Technology and Theoretical Computer Science (FST&TCS 2003) (Lecture Notes in Computer Science)*, Vol. 2914. Springer-Verlag, 112–123.

[15] Didier Caucal. 2002. On Infinite Terms Having a Decidable Monadic Theory. In *Proceedings of the 27th Symposium, Mathematical Foundations of Computer Science (MFCS 2002) (Lecture Notes in Computer Science)*, Vol. 2420. Springer-Verlag, 165–176.

[16] Alonzo Church and J. Barkley Rosser. 1936. Some Properties of Conversion. *Trans. Amer. Math. Soc.* 39, 3 (mar 1936), 472–472.

[17] Bruno Courcelle. 1994. Monadic Second-Order Definable Graph Transductions: A Survey. *Theoretical Computer Science* 126, 1 (1994), 53–75.

[18] Bruno Courcelle. 1995. The Monadic Second-Order Logic of Graphs IX: Machines and their Behaviours. *Theoretical Computer Science* 151 (1995), 125–162.

[19] Werner Damm and Andreas Goerdt. 1986. An Automata-Theoretical Characterization of the OI-Hierarchy. *Information and Computation* 71 (1986), 1–32.

[20] Heinz-Dieter Ebbinghaus, Jörg Flum, and Wolfgang Thomas. 1996. *Mathematical Logic* (second edition ed.). Springer-Verlag.

[21] Joost Engelfriet, Hendrik Jan Hoogeboom, and Jan-Pascal van Best. 1999. Trips on Trees. *Acta Cybernetica* 14, 1 (1999), 51–64.

[22] Séverine Fratani. 2006. *Automates à piles de piles ... de piles*. Ph.D. Dissertation. Université de Bordeaux.

[23] Charles Grellois and Paul-André Melliès. 2015. Finitary Semantics of Linear Logic and Higher-Order Model-Checking *(Lecture Notes in Computer Science)*, Vol. 9234. Springer-Verlag, 256–268.

[24] Yuri Gurevich and Leo Harrington. 1982. Trees, Automata, and Games. In *Proceedings of the Fourteenth Annual ACM Symposium on the Theory of Computing (STOC'82)*. ACM, 60–65.

[25] Yuri Gurevich and Saharon Shelah. 1983. Rabin's Uniformization Problem. *Journal of Symbolic Logic* 48, 4 (1983), 1105–1119.

[26] Axel Haddad. 2012. IO vs OI in Higher-Order Recursion Schemes. In *Proceedings 8th Workshop on Fixed Points in Computer Science (Electronic Proceedings in Theoretical Computer Science)*, Vol. 77. 23–30.

[27] Axel Haddad. 2013. Model Checking and Functional Program Transformations. In *Proceedings of the 33rd International Conference on Foundations of Software Technology and Theoretical Computer Science (FST&TCS 2013) (LIPIcs)*, Vol. 24. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 115–126.

[28] Axel Haddad. 2013. *Shape-Preserving Transformations of Higher-Order Recursion Schemes.* Ph.D. Dissertation. Université Paris Diderot - Paris 7.

[29] Matthew Hague, Andrzej S. Murawski, C.-H. Luke Ong, and Olivier Serre. 2008. Collapsible Pushdown Automata and Recursion Schemes. In *Proceedings of the 23rd Annual IEEE Symposium on Logic in Computer Science (LiCS 2008)*. IEEE Computer Society, 452–461.

[30] Matthew Hague, Andrzej S. Murawski, C.-H. Luke Ong, and Olivier Serre. 2017. Collapsible Pushdown Automata and Recursion Schemes. *ACM Transactions on Computational Logic* 18, 3 (2017), 25:1–25:42.

[31] J. Martin E. Hyland and C.-H. Luke Ong. 2000. 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. *Information and Computation* 163 (2000), 285–408.

[32] David Janin and Igor Walukiewicz. 1996. On the Expressive Completeness of the Propositional mu-Calculus with Respect to Monadic Second Order Logic. In *Proceedings of the 7th International Conference on Concurrency Theory (CONCUR 1996) (Lecture Notes in Computer Science)*, Vol. 1119. Springer-Verlag, 263–277.

[33] Teodor Knapik, Damian Niwiński, and Paweł Urzyczyn. 2001. Deciding Monadic Theories of Hyperalgebraic Trees. In *Proceedings of the 5th conference on Typed Lambda Calculi and Applications (TLCA 2001) (Lecture Notes in Computer Science)*, Vol. 2044. Springer-Verlag, 253–267.

[34] Teodor Knapik, Damian Niwiński, and Paweł Urzyczyn. 2002. Higher-Order Pushdown Trees Are Easy. In *Proceedings of the 5th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS 2002) (Lecture Notes in Computer Science)*, Vol. 2303. Springer-Verlag, 205–222.

[35] Teodor Knapik, Damian Niwiński, Pawel Urzyczyn, and Igor Walukiewicz. 2005. Unsafe Grammars and Panic Automata. In *Proceedings of the 32nd International Colloquium on Automata, Languages, and Programming (ICALP 2005) (Lecture Notes in Computer Science)*, Vol. 3580. Springer-Verlag, 1450–1461.

[36] Naoki Kobayashi. 2009. Types and Higher-Order Recursion Schemes for Verification of Higher-Order Programs. In *Proceedings of the 36th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 2009)*. ACM, 416–428.

[37] Naoki Kobayashi and C.-H. Luke Ong. 2009. A Type System Equivalent to the Modal Mu-Calculus Model Checking of Higher-Order Recursion Schemes. In *Proceedings of the 24th Annual IEEE Symposium on Logic in Computer Science (LiCS 2009)*. IEEE Computer Society, 179–188.

[38] C.-H. Luke Ong. 2006. On Model-Checking Trees Generated by Higher-Order Recursion Schemes. In *Proceedings of the 21st Annual IEEE Symposium on Logic in Computer Science (LiCS 2006)*. IEEE Computer Society, 81–90.

[39] Paweł Parys. 2012. On the Significance of the Collapse Operation. In *Proceedings of the 27th Annual IEEE Symposium on Logic in Computer Science (LiCS 2012)*. IEEE Computer Society, 521–530.

[40] Pawel Parys. 2018. Recursion Schemes and the WMSO+U Logic. In *Proceedings of the 35th Symposium on Theoretical Aspects of Computer Science (STACS 2018) (LIPIcs)*, Vol. 96. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 53:1–53:16.

[41] Nir Piterman and Moshe Y. Vardi. 2004. Global Model-Checking of Infinite-State Systems. In *Proceedings of the 16th International Conference on Computer Aided Verification (CAV 2004) (Lecture Notes in Computer Science)*, Vol. 3114. Springer-Verlag, 387–400.

[42] Michael O. Rabin. 1969. Decidability of Second-Order Theories and Automata on Infinite Trees. *Trans. Amer. Math. Soc.* 141 (1969), 1–35.

[43] Sylvain Salvati and Igor Walukiewicz. 2013. Using Models to Model-Check Recursive Schemes. In *Proceedings of the 11th International Conference on Typed Lambda Calculi and Applications (TLCA'13) (Lecture Notes in Computer Science)*, Vol. 7941. Springer-Verlag, 189–204.

[44] Robert S. Streett and E. Allen Emerson. 1989. An Automata Theoretic Decision Procedure for the Propositional Mu-Calculus. *Inf. Comput.* 81, 3 (1989), 249–264. https://doi.org/10.1016/0890-5401(89)90031-X

[45] Wolfgang Thomas. 1997. Languages, Automata, and Logic. In *Handbook of Formal Language Theory*, G. Rozenberg and A. Salomaa (Eds.). Vol. III. Springer-Verlag, 389–455.

[46] Igor Walukiewicz. 2004. A Landscape with Games in the Background. In *Proceedings of the 19th Annual IEEE Symposium on Logic in Computer Science (LiCS 2004)*. Computer Society Press, 356–366.

[47] Thomas Wilke. 2001. Alternating Tree Automata, Parity Games and Modal μ-Calculus. *Bulletin of the Belgian Mathematical Society* 8, 2 (2001), 359–391.