Playing with Trees and Logic

Olivier Serre

UFR d'Informatique

Playing with Trees and Logic

Mémoire d'habilitation à diriger des recherches

par

Olivier Serre

Soutenu publiquement le 10 mars 2015 devant le jury constitué de

Pierre-Louis Curien	Rapporteur
Paul Gastin	Rapporteur
Damian Niwiński	Rapporteur
Christel Baier	Examinatrice
Anca Muscholl	Examinatrice
Jean-Marc Talbot	Examinateur

À Solveig et aux rires des enfants

Remerciements

Je voudrais tout d'abord remercier les rapporteurs de ce manuscrit, Pierre-Louis Curien, Paul Gastin et Damian Niwiński. Ce sont trois scientifiques que je respecte profondément et c'est pour moi un grand honneur qu'ils m'ont fait en acceptant cette tâche. Je les remercie également d'avoir digéré sans broncher ce manuscrit qui devaient faire «environ quatre-vingts pages» et qui, après quelques années de rédaction, a largement excédé la longueur annoncée initialement. Pierre-Louis, je te remercie pour ta lecture précise et tes commentaires précieux ; je te remercie aussi pour ton ouverture d'esprit scientifique rare qui permet de donner à ce jury une coloration sémantique et qui me laisse à penser qu'un jour nos communautés se rapprocheront encore un peu plus (on y travaille !). Paul, je te remercie pour avoir suivi mes travaux depuis longtemps puisque tu étais au LIAFA au début de ma thèse ; je te remercie également pour la confiance que tu me témoignes depuis toutes ces années. Damian, je te remercie pour tes travaux qui sont depuis le début de ma carrière scientifique une source constante d'inspiration ; je te remercie aussi pour ta gentillesse, que ce soit à un arrêt de bus tôt un matin à Turku (arrêt de bus où se trouvait également Stephen Cook !) ou tard dans la nuit à Varsovie, où tu m'as posé l'une des questions les plus pertinente de ces dernières années sur mes travaux.

Je voudrais également remercier les examinateurs de cette habilitation, Christel Baier, Anca Muscholl et Jean-Marc Talbot. Christel, Vielen Dank for having kindly accepted to be part of the jury: your work on probabilitstic automata has been a great source of inspiration for me and you textbook *Principle of Model Checking* is a precious companion. Jean-Marc, je suis ravi que tu ais accepté de venir pour cette soutenance : ta présence témoigne de ta curiosité scientifique (tu vas bientôt être un véritable expert des schémas récursifs !) et des liens qui unissent depuis quelques années nos deux équipes. Anca, je ne pouvais imaginer cette soutenance sans ta présence : tu avais co-encadré ma thèse il y a déjà pas mal d'années et tu as depuis toujours suivi mon parcours avec intérêt ; j'ai une profonde admiration pour tes travaux dont la diversité et la richesse sont rares dans notre communauté.

Je voudrais aussi remercier Jean-Éric Pin pour tous les conseils qu'il m'a donnés depuis ce premier jour où j'étais venu lui demander de faire un stage de DEA avec lui. Un peu plus tard, il m'a appris que tout théorème de moins de vingt-quatre heures était faux, mais qu'en s'accrochant on pouvait en donner un énoncé correct. Je le remercie enfin pour être cet ange gardien qui garde un œil sur moi et sait m'aider lorsque c'est nécessaire.

Ce manuscrit ne serait pas ce qu'il est sans toutes celles et ceux avec qui j'ai pu collaborer au cours de ces années. Merci donc (par ordre purement chronologique) à Alexis-Julien, Igor, Christof, Madhu, Vince, Carsten, Matthew, Andrzej, Luke, Arnaud, Antoine, Benjamin, Axel, Nello, Blaise, Anca, Marc, Vincent, Chris, Axel, Dietmar, Nathanaël, Sophie, Moshe et (par anticipation : ça portera chance) Didier, Michel, Nathanaël et Frédéric.

Il serait injuste ici d'oublier de remercier ceux qui, même si nous n'avons jamais écrit ensemble, ont discuté de sciences (ou autres !) de nombreuses heures avec moi : je pense à Thomas bien sûr, mais aussi à Łukasz, à Paul-André, à Jean, à Olivier, à Christian, à Sylvain, à Christiane, à Peter, à

Ines, à Irène, et plus généralement à celles et ceux qui composent le LIAFA-PPS. Merci aussi aux membres de l'équipe *Automates et Applications* pour leur confiance.

Je voudrais également remercier mes étudiants et stagiaires — David, Vincent, Axel, Charles, Laureline — ils m'ont tous beaucoup apporté et j'espère que j'ai pu leur transmettre ma passion.

I would also like to thanks Luke Ong. He is such a nice person and such a fantastic researcher! The talk *Infinite trees, higher-order recursion schemes and game semantics* he gave at PPS on the 23rd of October 2005 had a very strong impact on my research, and when he offered me to visit him some month later in Oxford it was the beginning of such an exciting collaboration with him and his co-workers. Without Luke I would never had the opportunity to work with Matthew Hague and Christopher Broadbent: it was so nice to have you guys in Paris during your post-doctoral stay!

Je voudrais aussi remercier Christof Löding pour nos collaborations au cours de ces années et aussi pour tous les bons moments passés à Aachen avec lui (un grand merci aussi à Wolfgang et Renate !). Chercher et écrire avec Christof est l'une des choses que je préfère dans ce métier.

Je voudrais aussi remercier toutes celles et ceux qui font que la vie au LIAFA est simple et agréable. En premier lieu Noëlle et Nathalie pour leur efficacité et leur gentillesse. Merci Noëlle pour ces discussions d'après 19h quand le LIAFA est désert. Merci aussi à Laifa et à Houy pour les discussions tôt le matin et pour prendre toujours cinq minutes pour régler mes problèmes techniques. Merci aussi à Pierre et Valérie qui dirigent depuis plusieurs années ce laboratoire d'une main de maître ; et merci aussi d'avoir su me mettre la pression pour enfin rédiger cette habilitation, même si elle est quatre fois plus longue que ce que vous m'aviez conseillé !

Merci aussi à mes collègues du DIX avec qui j'enseigne depuis de nombreuses années. Une mention spéciale à l'équipe du Modal Web : Dominique, Baptiste, Andrey, et les «historiques», Thomas, Sylvie, Julien, Christophe et Cécile.

Dans des remerciements, il y a toujours un moment plutôt net où l'on bascule entre les collègues et les amis/la famille. Il y a une personne cependant que je ne sais pas où situer dans cette nébuleuse : il est plus qu'un ami, et il est aussi mon collègue le plus fidèle et le plus inspirant. Merci donc à Arnaud Carayol. C'est sans doute avec toi que j'ai produit mes plus beaux résultats, et c'est aussi grâce à toi que j'ai réussi à continuer à avancer quand j'avais envie de baisser les bras. Merci pour tous ces moments de chercheurs, et merci surtout pour tous les autres, passés et à venir.

Un grand merci aussi à ceux dont l'amitié fidèle a permis d'avoir une autre vie que celle de chercheur en informatique : Vaness, Éric et Weijia, Jean-Pierre, Lydie, Céline et Xavier, Guillemette. Merci aussi à François M., qui un jour de décembre 2013 a su me faire comprendre qu'il me fallait finir ce manuscrit. Merci aussi à Didier, Ingrid, Marie-Pierre, Kris, Magdalena et Julie pour tout ce qu'ils font pour ma famille dans cette autre vie qui est la nôtre…

Merci à ma famille : mes parents, ma grand mère, ma sœur, mes beaux-parents, ma belle sœur,… Votre soutien indéfectible est l'un des repères les plus précieux qui soit.

Je voudrais aussi remercier mes enfants. Merci donc, par ordre d'apparition, à Timothée, Félix et Arsène : vous êtes ce que j'ai fait de plus beau. Gardez votre pureté le plus longtemps possible, et ne vous souciez pas de ce que les autres en penseront : ils auront tort ; la vérité n'est pas dans un moule bien lisse dans lequel chacun devrait entrer, mais elle se situe dans la richesse de ce qui est rugueux et différent.

Enfin, je voudrais terminer par celle qui est là depuis si longtemps et sans qui je ne serais jamais devenu celui que je suis aujourd'hui. Merci donc à toi Solveig. J'admire tellement de choses chez toi — la compagne, la mère, la chercheur… — que je ne pourrais trouver les mots pour le dire fidèlement en quelques lignes. Ce manuscrit est déjà bien assez long, et il s'agit là d'une autre histoire…

Structure of this Document

This document collects selected pieces of the research work I have been doing since the defense of my PhD thesis in 2004 (actually the focus here is on work done since 2007).

I voluntarily made a selection between my contributions in order to write a thematically consistent story (nevertheless personal references not cited in this document are given on page 163). In the end, it rather seems that I have written two stories, and for this reason this document mainly consists on two fully independent chapters (with their own introduction and perspectives) preceded by a short preliminary chapter introducing basic objects. This explains why there is no global introduction nor general conclusion.

The first story, called *Higher-Order Recursion Schemes & Collapsible Pushdown Automata* is built on top of the papers [HMOS08; CMHOS08; BCOS10; CS12a; CS12b; BCHS12; BCHS13] while the second story, called *Tree Automata and Imperfect Information Games* is built on top of the papers [GS09; CHS11; CHS14b; CHS14a; FPS13; CLS12].

As the goal was to keep this as little technical as possible, I tried to give examples of the main objects rather than proofs of the statements they are used in. Of course most of the proofs have been published already and the missing ones can easily be obtained by sending me an email.

L'illogisme irrite. Trop de logique ennuie. La vie échappe à la logique, et tout ce que la seule logique construit reste artificiel et contraint. Donc est un mot que doit ignorer le poète, et qui n'existe que dans l'esprit.

André Gide, *Journal*.

Preliminaries 1

This chapter is devoted to classical notions that will be used in several places in this document. We start with notations for basic objects, and then for words and graphs. Then we briefly recall the notion of finite automata on finite and infinite words. Then we introduce a very basic class of games on graphs, *two-player perfect information turn-based games*, that will be used in Chapter 2 due to their tight connections with logic-related problem, and extended in many ways in Chapter 3 in order to solve the emptiness problem for several classes of finite automata on infinite trees. Finally, we terminate with basic notions on logic with a special focus on first-order and second-order logic.

1 Some Basic Objects

A ***probability distribution*** over a finite set X is a map $d : X \to [0,1]$ such that $\sum_{x \in X} d(x) = 1$. In the sequel we denote by $\mathcal{D}(X)$ the set of probability distributions over X. The ***support*** of a distribution d is the set $\mathrm{Supp}(d) = \{x \in X \mid d(x) > 0\}$.

Given some set X and some equivalence relation $\sim$ over X, $[x]_\sim$ stands for the equivalence class of x for $\sim$ and $X/_\sim = \{[x]_\sim \mid x \in X\}$ denotes the set of equivalence classes of $\sim$.

If f is a function, we denote by $\mathrm{Dom}(f)$ its domain.

2 Words and Graphs

An ***alphabet*** A is a (possibly infinite) set of letters. In the sequel A^* denotes the set of finite words over A, and A^ω the set of infinite words over A. The ***empty word*** is written ε; the length of a word u is denoted by $|u|$. For any $k \geqslant 0$, we let $A^k = \{u \mid |u| = k\}$, $A^{\leqslant k} = \{u \mid |u| \leqslant k\}$ and $A^{\geqslant k} = \{u \mid |u| \geqslant k\}$. 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, denoted $u \sqsubseteq v$, iff there exists a word w such that $v = u \cdot w$. We denote by $u \sqsubset v$ the fact that u is a strict prefix of v (*i.e.* $u \sqsubseteq v$ and $u \neq v$). For some word u and some integer $k \geqslant 0$, we denote by u^k the word obtained by concatenating k copies of u (with the convention that $u^0 = \varepsilon$).

A ***graph*** is a pair $G = (V, E)$ where V is a set of ***vertices*** and $E \subseteq V \times V$ is a set of ***edges***. For every vertex v we let $E(v) = \{w \mid (v,w) \in E\}$. A ***dead-end*** is a vertex v such that $E(v) = \varnothing$. The ***size*** of a graph is defined to be $|V| + |E|$.

3 Finite Automata, Regular Languages and ω-Regular Languages

We often use the notion of regular languages of finite and infinite words in this document. Hence, we briefly recall the definitions and refer to classical textbooks for more details on that topic

(see *e.g.* [HMU07] for finite words and [PP04] for infinite words).

A ***finite automaton*** is a tuple $\mathcal{A} = (Q, A, \Delta, I, F)$ where Q is a finite set of states, A is an input alphabet, $\Delta \subseteq Q \times A \times Q$ is a transition relation, I is a set of initial states and F is a set of final states.

Let $u = a_1 a_2 \cdots a_\ell \in A^*$. A run of $\mathcal{A}$ on u is a sequence $q_0, q_1, \cdots q_\ell$ such that $q_0 \in I$ and, for all $1 \leqslant i \leqslant \ell$, $(q_{i-1}, a_i, q_i) \in \Delta$; it is accepting if one has $q_\ell \in F$. A word u is accepted by $\mathcal{A}$ if there exists an accepting run of $\mathcal{A}$ on u. Finally, we let $L(\mathcal{A}) \subseteq A^*$ denote the set of all words accepted by $\mathcal{A}$. A language $L \subseteq A^*$ of finite words is ***regular*** if there exists a finite automaton $\mathcal{A}$ such that $L = L(\mathcal{A})$.

Let $u = a_1 a_2 a_3 \cdots \in A^\omega$. A run of $\mathcal{A}$ on u is an infinite sequence $q_0, q_1, q_2 \cdots$ such that $q_0 \in I$ and, for all $i \geqslant 1$, $(q_{i-1}, a_i, q_i) \in \Delta$; it is accepting if one has $q_i \in F$ for infinitely many i. An infinite word u is (Büchi-) accepted by $\mathcal{A}$ if there exists an accepting run of $\mathcal{A}$ on u. Finally, we let $L(\mathcal{A}) \subseteq A^\omega$ denote the set of all infinite words accepted by $\mathcal{A}$. A language $L \subseteq A^\omega$ of infinite words is ***ω-regular*** if there exists a finite automaton $\mathcal{A}$ such that $L = L(\mathcal{A})$.

4 Two-Player Perfect Information Turn-Based Games on Graphs

An ***arena*** is a triple $\mathcal{G} = (G, V_E, V_A)$ where $G = (V, E)$ is a graph and $V = V_E \uplus V_A$ is a partition of the vertices among two players, Éloïse and Abélard. For simplicity in the definitions, we assume that G has no dead-end[1].

Éloïse and Abélard 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 $v_0 \in V_E$, Abélard 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 $0 \leqslant i$ 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 $0 \leqslant i < \ell$ one has $v_{i+1} \in E(v_i)$.

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

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 Abélard. 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 Abélard are defined likewise.

We now define some classical winning conditions.

- A ***reachability*** winning condition is one of the form $V^* F V^\omega$, *i.e.* winning plays are those that eventually visit a vertex in F (we refer to vertices in F as final ones).

- A ***safety*** winning condition is one of the form $(V \backslash F)^\omega$, *i.e.* winning plays are those that eventually visit a vertex in F (we refer to vertices in F as forbidden ones).

[1]Note that for the use of games in the present document this is not restrictive as, up to coding, we can always reduce to this setting.

- A ***Büchi*** winning condition is one of the form $\bigcap_{k\geqslant 0} V^k V^* F V^\omega$, *i.e.* winning plays are those that infinitely often visit vertices in F (we refer to vertices in F as final/Büchi ones).
- A ***co-Büchi*** winning condition is one of the form $V^*(V\backslash F)V^\omega$, *i.e.* winning plays are those that finitely often visit vertices in F (we refer to vertices in F as forbidden/Büchi ones).
- A ***parity*** winning condition is defined by a *colouring* function Col that is a mapping $\text{Col} : V \to \text{Colours} \subset \mathbb{N}$ where Colours is a *finite* set of ***colours***. The parity winning condition associated with Col is the set $\Omega_{\text{Col}} = \{v_0 v_1 \cdots \in V^\omega \mid \liminf(\text{Col}(v_i))_{i\geqslant 0} \text{ is even}\}$, *i.e.* a play is winning if and only if the smallest colour infinitely often visited is even.

Note that Büchi winning conditions are those parity conditions using only colours 0 and 1 (final vertices are those coloured by 0) while co-Büchi winning conditions are those parity conditions using only colours 1 and 2 (forbidden vertices are those coloured by 1).

Finally a reachability (resp. safety/Büchi/co-Büchi/parity) game is one equipped with a reachability (resp. safety/Büchi/co-Büchi/parity) winning condition. For notation of such games we often replace the winning condition by the objects that is used to define it (*i.e.* a set F or a colouring function Col).

Of special interest are strategies that do not require memory. A ***positional strategy*** is a strategy φ such that for any two partial plays of the form $\lambda.v$ and $\lambda'.v$, one has $\varphi(\lambda \cdot v) = \varphi(\lambda' \cdot v)$, *i.e.* φ only depends on the current vertex. It is a well known result that positional strategies suffice to win in parity games (see *e.g.* [Zie98] for a proof of this result in the setting of games played on infinite graphs).

Theorem 1 (Positional determinacy)

Let $\mathbb{G}$ be a parity game. Then for any vertex, either Éloïse or Abélard has a positional winning strategy.

We represent positional strategies as functions from V_E (or V_A depending on the player) into V. Using Theorem 1 it is simple to obtain the following complexity result on solving parity games on finite graphs.

Theorem 2

Let $\mathbb{G}$ be a parity game played on a *finite* arena and let v be some vertex. Then deciding whether v is winning for Éloïse is an NP∩co-NP problem.

***Remark* 1.** *Finding a polynomial time for the previous problem is one of the most challenging question on the community working on games on graphs.*

5 Relational Structures and Logics

We now give some brief background on logic (mainly for trees and labelled transition systems). More thorough introductions to the topic can be found in many textbooks and survey, *e.g.* in [EFT96; Tho97].

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, think of a complete binary node-labelled tree, *i.e.* a map $t : \{0,1\}^* \to A$ that associates with any node $u \in \{0,1\}^*$ a label in a finite alphabet A. Then t can be represented by the relational structure $\underline{t} = (\{0,1\}^*, S_0, S_1, \sqsubset, (Q_a)_{a \in A})$ where S_0 (resp. S_1) is the left successor relation defined by $S_0 = \{(u, u0) \mid u \in \{0,1\}^*\}$ (resp. $S_1 = \{(u, u1) \mid u \in \{0,1\}^*\}$), $\sqsubset$ is the strict prefix ordering on $\{0,1\}^*$, and $Q_a = t^{-1}(a)$ for each label $a \in A$.

In a similar manner labelled transition systems (*i.e.* vertex- and edge-labelled directed graphs) as defined in Section 2 of Chapter 2 can be represented as relational structures: 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 trees or labelled transition systems can be formalised in logical languages. 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.* [EFT96]). 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. 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 x_i as p_i for each $i = 1, \ldots, k$.

Example 1 (FO formulas on trees)

As an example of an FO formula on $\{a, b\}$-labelled complete binary trees, consider the formula

$$\varphi \;:=\; \exists x\, [Q_a(x) \wedge \forall y\; x \sqsubset y \Rightarrow Q_b(y)]$$

Formula φ holds in a tree if and only if it contains a subtree whose root is labelled by a while all other nodes are labelled by b.

Consider the following formula

$$root(x) \;:=\; \neg\exists y\, [S_0(y,x) \vee S_1(y,x)]$$

It holds in a tree if and only if one interprets x as the root of the tree.

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 labelled transition systems). 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.

Example 2 (MSO formulas on trees)

Consider the following formula

$$\varphi \;:=\; \exists X\, [\forall x\; x \in X \Rightarrow (Q_a(x) \wedge \exists y\; (y \in X \wedge x \sqsubset y))]$$

This formula holds in a tree if and only if it contains an infinite branch with infinitely many nodes labelled by a.

Second-order quantification permits to express the prefix relation, *i.e.*

$$x \sqsubset y \;:=\; \neg[x = y] \wedge \forall X\, [x \in X \wedge \forall z \forall z' (z \in X \wedge (S_0(z, z') \vee S_1(z, z'))) \Rightarrow z' \in X] \Rightarrow y \in X$$

Indeed, the second part of the formula states that any set of nodes X that is closed under the child-relation and that contains x should also contains y.

Concerning expressiveness, FO is much weaker than MSO. For that reason some intermediate logics have been introduced by extending FO with various operators.

The ***Transitive closure logic FO(TC)*** (see *e.g.* [WT07] from which the forthcoming definitions and examples are taken) is defined by extending FO with formulas of the type

$$\psi \;:=\; [TC_{\overline{x},\overline{y}}\, \varphi(\overline{x}, \overline{y}, \overline{z})]\, \overline{s}, \overline{t}$$

where $\varphi(\overline{x}, \overline{y}, \overline{z})$ is an FO(TC) formula, $\overline{x}, \overline{y}$ are disjoint tuples of free variables of the same length $k > 0$, $\overline{s}, \overline{t}$ are tuples of variables of length k and the free variables in ψ are those free variables in φ together with $\{\overline{s}, \overline{t}\}$ and minus $\{\overline{x}, \overline{y}\}$ (that are bound by the TC quantifier). Let $\mathfrak{A}$ be a relational structure, let $\overline{c}, \overline{d}$ and $\overline{e}$ be the interpretations of the variables $\overline{z}$, $\overline{s}$ and $\overline{t}$ in φ. Let E be the relation of k-tuples defined by

$$E(\overline{c}) \;:=\; \{(\overline{a}, \overline{b}) \mid (\mathfrak{A}, \overline{a}, \overline{b}, \overline{c}) \models \varphi(\overline{x}, \overline{y}, \overline{z})\}$$

and $E^+(\overline{c})$ be its transitive closure, *i.e.* $(\overline{a}, \overline{b}) \in E^+(\overline{c})$ if and only if there exists a sequence $\overline{f}_0, \overline{f}_1, \ldots \overline{f}_\ell$ (with $\ell > 0$) such that $\overline{f}_0 = \overline{a}$, $(\overline{f}_i, \overline{f}_{i+1}) \in E(\overline{c})$ for each $i = 1, \ldots, \ell - 1$ and $\overline{f}_\ell = \overline{b}$. Then the semantics of the FO(TC) formula ψ is defined by letting $(\mathfrak{A}, \overline{c}, \overline{d}, \overline{e}) \models \psi$ if and only if $(\overline{d}, \overline{e}) \in E^+(\overline{c})$.

Example 3 (FO(TC) formula on labelled transition systems)

Assume one considers labelled transition systems and let Σ be a subset of the labels allowed on edges and let E_a be the binary relation for a-labelled edges. Then the following formula express that from a vertex x a vertex y is reachable via a path with labels from Σ

$$Reach_\Sigma(x, y) \;:=\; [TC_{x,y}\, (x = y \vee \bigvee_{a \in \Sigma} E_a(x, y))]x, y$$

The restriction of FO(TC) where the only transitive closure formulas allowed are of the form $Reach_\Sigma(x, y)$ as in Example 3 is called ***reachability logic*** and is denoted $\boldsymbol{FO(Reach)}$.

In this document, we will also consider a fixpoint modal logic called ***μ-calculus***. As we will never explicitly write a formula from μ-calculus and mostly rely on the fact that there exists a strong connexion between μ-calculus and parity games, we do not give any definition regarding to this logic and refer the reader to [AN01; Wil01].

I was ridin', I was ridin', oh
The sun, the sun, the sun was rising from the field
I got a feeling I just can't shake

I got a feeling that just won't go away
You've got it, just keep on pushing and, keep on pushing and
Push the sky away
And if your friends think that you should do it different

And if they think that you should do it the same
You've got it, just keep on pushing and, keep on pushing and
Push the sky away
And if you feel you got everything you came for

If you got everything and you don't want no more
You've got it, just keep on pushing and, keep on pushing and
Push the sky away
And some people say it's just rock 'n' roll

Aw, but it gets you right down to your soul
You've got it, just keep on pushing and, keep on pushing and
Push the sky away
You've got it, just keep on pushing and, keep on pushing and

Push the sky away
You've got it, just keep on pushing and, keep on pushing and

Push the sky away

Nick Cave, *Push The Sky Away*.

Higher-Order Recursion Schemes & Collapsible Pushdown Automata

2

An old model of computation, recursion schemes 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. This chapter, based on a series of papers [HMOS08; CMHOS08; BCOS10; CS12b; CS12a; BCHS12; BCHS13], gives an overview of the contributions I was involved in on that topic.

The presentation tries to avoid excessive formalism and hence no proofs are given. We also mention results obtained by other authors to give a complete landscape and also to permit to appreciate the impact of our results in the community.

The chapter is structured as follows. We start in Section 1 by an informal introduction to the process of modelling higher-order programs for verification[1], and one can think of it as a "practical motivation" for the content of this chapter. Section 2 is devoted to historical considerations on the topic. The general definitions are given in Section 3. Section 4 introduces the reader with recursion schemes and Section 5 provide examples to get the reader familiar with schemes. Section 6 introduces the notion of collapsible pushdown automata while Section 7 revisits the example of Section 5 in this new setting. Section 8 introduces safety, a syntactical constraint on schemes, and present one of the central results of the chapter, namely the Equi-expressivity theorem, stating that recursion schemes and collapsible pushdown automata have the same expressive power for generating infinite node-labelled ranked trees. Section 9 presents standard logical questions (model-checking) as well as less classical ones (reflection, selection) and gives decidability status of these various problems as well as some consequences. Section 10 considers parity games played on configuration graph of collapsible pushdown automata, which is the central tool for tackling logical questions as explained in Section 11; it presents the second central result of the chapter, namely Theorem 24 that establishes the decidability of the previous games. Section 12 presents recent advances based on saturation techniques that permit efficient computations and are at the core of the C-SHORe tool. Finally, Section 13 present several natural problems as well as perspectives related to the topics discussed in the present chapter.

1 Modelling Higher-Order Programs

Via languages such as C++, Haskell, OCaML, Javascript, Python, or Scala, modern day programming increasingly embraces higher-order procedure calls. This is a challenge for software verification, which usually does not model recursion accurately, or models only first-order calls (*e.g.*

[1] This section is actually taken from [BCHS13].

SLAM [BR02] and Moped [Sch02]). The `map` function, found in many functional programming languages, is one example of a higher-order function. It is a generic iterator: it takes as arguments a function f and a list of elements, and as the result, returns a new list with f applied to each element from the list.

In this section we give an informal introduction to the process of modelling higher-order programs for verification. In particular, we show how a simple example program can be modelled using a higher-order recursion scheme, and then we show how this scheme is evaluated using a collapsible pushdown system. For a more systematic approach to modelling higher-order programs with recursion schemes, we refer the reader to work by Kobayashi *et al.* [KSU11].

For this section, consider the toy example below.

```
Main = MakeReport Nil
MakeReport x = if * (Commit x)
               else (AddData x MakeReport)
AddData y φ = if * (φ (Error End)) else (φ Cons(_, y))
```

In this example, `*` represents a non-deterministic choice (that may, for example, be a result of some input by the user). Execution begins at the `Main` function whose aim is to make a report which is a list. We begin with an empty report and send it to `MakeReport`. Either `MakeReport` indicates the report is finished and commits the report somehow, or it adds an item to the head of the list, using the `AddData` function, which takes the report so far, and a continuation. `AddData` either detects a problem with the new data (maybe it is inconsistent with the rest of the report) and flags an error by passing `Error` to the continuation, or extends the report with some item. In this case, the programmer has not provided error handling as part of the `MakeReport` function, and so an `Error` may be committed.

1.1 Higher-Order Recursion Schemes

As a first step in modelling this program, we introduce, informally, higher-order recursion schemes. These are rewrite systems that generate the computation tree of a functional program. A rewrite rule takes the form

$$N\ \varphi\ x\ \rightarrow\ t$$

where N is a typed non-terminal with (possibly higher-order) arguments φ and x. A term $N\ t_\varphi\ t_x$ rewrites to t with t_φ substituted for φ and t_x substituted for x. Note that recursion schemes require t to be of ground type. We will illustrate the behaviour of a recursion scheme and its use in analysis using the toy example from above.

We can directly model our example with the scheme

$$\begin{cases} main & \rightarrow & M\ nil \\ M\ x & \rightarrow & or\ (commit\ x)\ (A\ x\ M) \\ A\ y\ \varphi & \rightarrow & or\ (\varphi\ (error\ end))\ (\varphi\ (cons\ y)) \end{cases}$$

where M is the non-terminal associated with the `MakeReport` function, and A is the non-terminal associated with the `AddData` function; nil, or, $commit$, $error$, $cons$ and end are terminal symbols of arity 0, 2, 1, 1, 1 and 0 respectively (e.g. in the second rule, or takes the two arguments $(commit\ x)$ and $(A\ x\ M)$). The scheme above begins with the non-terminal $main$ and, through a sequence of rewrite steps, generates a tree representation of the evolution of the program. Figure 1 shows such a sequence.

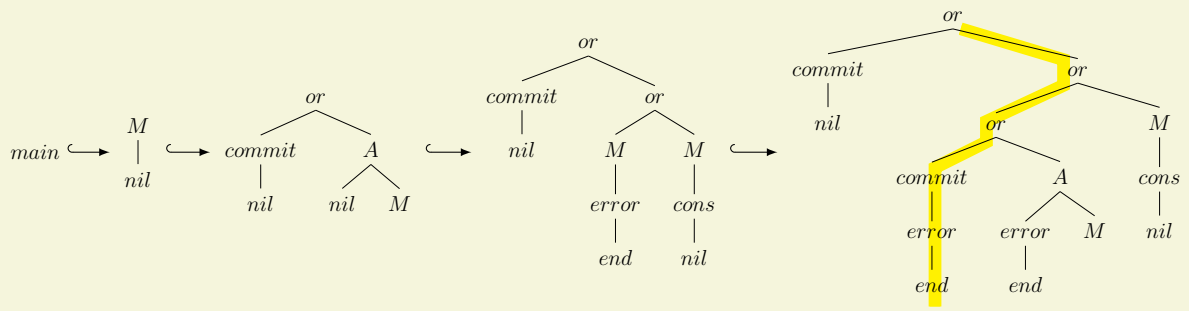


Figure 1 – The behaviour of a toy recursion scheme

Beginning with the non-terminal $main$, we apply the first rewrite rule to obtain the tree representing the term $(A\ nil)$. We then apply the second rewrite rule, instantiating x with nil to obtain the next tree in the sequence. This continues *ad infinitum* to produce a possibly infinite tree labelled only by terminals.

We are interested in ensuring the correctness of the program. In our case, this means ensuring that the program never attempts to $commit$ an $error$. By inspecting the rightmost tree in Figure 1, we can identify a branch labelled $or, or, or, commit, error, end$. This is an error situation because $commit$ is being called with an $error$ report. In general we can define the regular language $L_{err} = or^*\ commit\ or^*\ error\ end$. If the tree generated by the recursion scheme contains a branch labelled by a word appearing in L_{err}, then we have identified an error in the program.

1.2 Collapsible Pushdown Automata

Our approach for the verification of recursion schemes is to exploit the connection between higher-order recursion schemes and an automata model called *collapsible pushdown automata* (CPDA). These two formalisms are, in fact, equivalent[2].

Theorem 8 (Equi-Expressivity Theorem) [HMOS08]

For each order-n recursion scheme, there is an order-n collapsible pushdown automaton generating the same tree, and vice-versa. Furthermore, the translations in both directions are linear.

Note that other successful approaches were based on *intersection types* (e.g. [Kob11b; NRO12]) but we will not discuss them in this document.

We describe at a high level the structure of a CPDA and how they can be used to evaluate recursion schemes. In our case, this means outputting a sequence of non-terminals representing each path in the tree. More formal definitions are given in Section 6. At any moment, a CPDA is in a *configuration* (q, s), where q is a control state taken from a finite set Q, and s is a higher-order collapsible stack. In the following we will focus on the stack. Control states are only needed to ensure that sequences of stack operations occur in the correct order and are thus elided for clarity.

In the case of our toy example, we have an order-2 recursion scheme and hence an order-2 stack. An order-1 stack is a stack of characters γ from a finite alphabet Γ. An order-2 stack is a stack of order-1 stacks. Thus we can write $[\,[\,main\,]_1\,]_2$ to denote the order-2 stack containing only

[2] A more precise statement is given on Page 50

the order-1 stack $[main]_1$; $[main]_1$ is an order-1 stack containing only the character $main$. In general Γ will contain all subterms appearing in the original statement of our toy example recursion scheme. The evolution of the CPDA stack is given in Figure 2 and explained below.

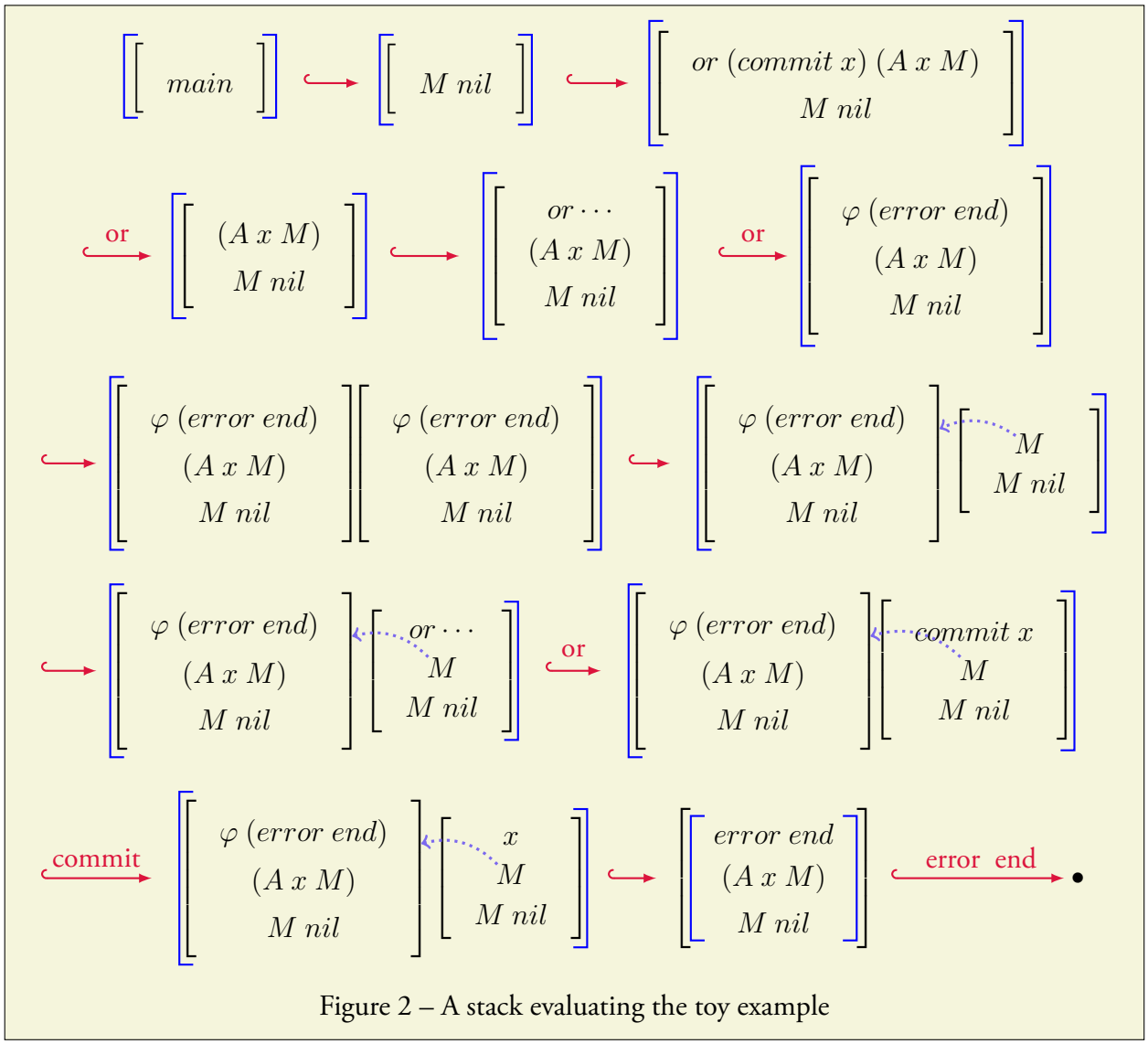


Figure 2 – A stack evaluating the toy example

The first step is to rewrite $main$ using $main \to M\ nil$. Since $(M\ nil)$ is a subterm of our recursion scheme, we have $(M\ nil) \in \Gamma$ and we simply rewrite the stack $[[[main]_1]_2$ to $[[[M\ nil]_1]_2$.

The next step is to call the function M. As is typical in the execution of programs, a function call necessitates a new stack frame. In particular, this means pushing the body of M (that is $(or\ (commit\ x)\ (A\ x\ M))$) onto the stack, resulting in the third stack in Figure 2. Note that we do not instantiate the variable x, hence we use only the subterms appearing in the recursion scheme.

Recall that we want to obtain a CPDA that outputs a sequence of terminals representing each path in the tree. To evaluate the term $or\ (\cdots)\ (\cdots)$ we have to output the terminal or and then (non-deterministically) choose a branch of the tree to follow. Let us choose $(A\ x\ M)$. Hence, the CPDA outputs the terminal or and rewrites the top term to $(A\ x\ M)$. Next we make a call to the A function, pushing its body on to the stack, and then pick out the $(\varphi\ (error\ end))$ branch of the or terminal. This takes us to the end of the second row of Figure 2.

To proceed, we have to evaluate $\varphi\ (error\ end)$. To be able to do this, we have to know the value of φ. We can obtain this information by inspecting the stack and seeing that the second argument

of the call of A is M. However, since we can only see the top of a stack, we would have to remove the character $\varphi\ (error\ end)$ to be able to determine that $\varphi = M$, thus losing our place in the computation.

This is where we use the power of order-2 stacks. An order-2 stack is able — via a $push_2$ operation — to create a copy of its topmost order-1 stack. Hence, we perform this copy (note that the top of the stack is written on the right) and delve into the copy of the stack to ascertain the value of φ. While doing this we also create a *collapse link*, pictured as an arrow from M to the term $\varphi\ (error\ end)$. This collapse link is a pointer from M to the context in which M will be evaluated. In particular, if we need to know the value of x in the body of M, we will need to know that M was called with the $error$ argument, within the term $\varphi\ (error\ end)$; the collapse link provides a pointer to this information (in other words we have encoded a closure in the stack). We can access this information via a *collapse* operation. These are the two main features of a higher-order collapsible stack, described formally in Section 6.

To continue the execution, we push the body of M on to the stack, output the or symbol and choose the $(commit\ x)$ branch. Since $commit$ is a terminal, we output it and pick out x for evaluation. To know the value of x, we have to look into the stack and follow the collapse link from M to $\varphi\ (error\ end)$. Note that we do not need to create a copy of the stack here because x is an order-0 variable and thus represents a self-contained execution. Since $error\ end$ is the value of the argument we are considering, we pick it out and then output $error$ followed by end before terminating. This completes the execution corresponding to the error branch identified in Figure 1.

1.3 A Collapsible Pushdown Approach to Verification

The CPDA output $or, or, or, commit, error, end$ in the execution above. This is an error sequence in L_{err} and should be flagged. In general, to check a safety property of a program, we take the finite automaton $\mathcal{A}_{err}$ representing the regular language L_{err} and form a synchronised product with the CPDA described above. This results in a CPDA that keeps in its control state the progression of $\mathcal{A}_{err}$. Thus we are interested in whether the CPDA is able to reach an accepting state of $\mathcal{A}_{err}$.

Of course, in general one can be interested in checking more involved properties of system than safety, typically any property expressible in a rich logical formalism, *e.g.* the monadic second order logic. Again, one can use the correspondence between recursion schemes and collapsible pushdown automata to restate the verification problem on the recursion scheme as a model-checking problem on the infinite transition system generated by the equi-expressive CPDA. Designing such reductions and explaining how to solve the model-checking problem (and its extensions) on CPDA is the core topic of this chapter.

2 Historical Background

2.1 Recursive Applicative Program Schemes

Historically (we refer the reader to [Mir06] that, among others, contains a very detailed and rich history of the topic), recursion schemes go back to Nivat's *recursive applicative program schemes* [Niv72] that correspond to order-1 recursion schemes in our sense (see also related work by Garland and Luckham on so called *monadic recursion schemes* [GL73]). To Nivat, a recursive applicative program scheme is a finite system of equations, each of the form $F_i(x_1, \ldots, x_{n_i}) = p_i$ where the x_j are order-0 variables and p_i is some order-0 term over the nonterminals (the F_i's), terminals and the variables $x_1, \ldots, x_{n_i}$. In Nivat's work, a *program* is a pair : a program scheme together with

an *interpretation* over some domain. An interpretation gives any terminal a function (of the correct rank) over the domain. Taking the least fixed point of the rewriting rules of a program scheme gives a (possibly infinite) term over the terminals alphabet (known as the value of the program in the *free/Hebrand interpretation*); applying the interpretation to this infinite term gives the *value* of the program. Hence, the program scheme gives the uninterpreted syntax tree of some functional program that is then fully specified owing to the interpretation.

Nivat also defined a notion of equivalence for program schemes: two schemes are equivalent if and only if they compute the same function under *any* interpretations. Later, Courcelle and Nivat [CN78] showed that two schemes are equivalent if and only if they generate the same infinite term tree. This latter result clearly stresses the importance of studying the trees generated by a scheme. Following the work by Courcelle [Cou78a; Cou78b] the equivalence problem for schemes is known to be interreducible to the problem of decidability of language equivalence between deterministic pushdown automata (DPDA). Research on the equivalence for program schemes was halted until Sénizergues [Sén97; Sén02] established decidability of DPDA equivalence (see also simplifications by Stirling [Sti01; Sti00a]) which therefore also solved the scheme equivalence problem. For more insight about this topic we refer the reader to [Sti00b].

2.2 Extension of Schemes to Higher-Orders

In Nivat's program scheme, both the nonterminals and the variables have order-0, hence they cannot be used to provide a model of higher-order recursive programs. In the late 1970s, there was a substantial effort in extending them in order to capture higher-order [Ind76; Dam77a; Dam77b; ES77; ES78]. Note that evaluation, *i.e.* computing the value of a scheme in some interpretation, has been a very active topic, in particular because different evaluation policies, *e.g.* *call by name* (OI) or *call by value* (IO), lead to different semantics [ES77; ES78; Dam82]. In a very influential paper [Dam82], Damm introduced *order-n λ-schemes* and extended the previously mentioned result of Courcelle and Nivat. Damm's schemes mostly coincide with the *safe* fragment of the recursion schemes as we will define them later in this chapter. It is important to note that so far there was no known model of automata equi-expressive with Damm's scheme; in particular, there was no known reduction of the equivalence problem for schemes to a language equivalence problem for (some model of) automata.

Later, Damm and Goerdt [Dam82; DG86] considered the word languages generated by level-n λ-schemes and they showed that they coincide with a hierarchy previously defined by Maslov [Mas74; Mas76]. To define his hierarchy Maslov introduced *higher-order pushdown automata (PDA)* and he also gave an equivalent definition of the hierarchy in terms of *higher-order indexed grammars*. In particular, Maslov's hierarchy offers an attractive classification of the semi-decidable languages: orders 0, 1 and 2 are respectively the regular, context-free and indexed languages though little is known about languages at higher orders (see [IM08] for recent results on this topic). Later, Engelfriet [Eng83; Eng91] considered the characterisation of complexity classes by higher-order pushdown automata and he showed that alternating pushdown automata characterise deterministic iterated exponential time complexity classes.

2.3 Higher-Order Recursion Schemes as Generators of Infinite Structures

Since the late 1990s there has been a strong interest for infinite structures admitting finite descriptions (either internal, algebraic, logical or transformational), mainly motivated by applications to program verification. See [BGR11] for an overview about this topic. The central question is model-checking: given some presentation of a structure and some formula, decide whether the for-

mula holds. Of course here decidability is a trade-off between the richness of the structure and the expressivity of the logic.

Of special interest are tree-like structures. Higher-order PDA as a generating device for (possibly infinite) labelled ranked trees were first studied by Knapik, Niwiński and Urzyczyn [KNU02]. As in the case of word languages, an infinite hierarchy of trees is defined according to the order of the generating PDA; lower orders of the hierarchy are well-known classes of trees: orders 0, 1 and 2 are respectively the regular [Rab69], algebraic [Cou95] and hyperalgebraic trees [KNU01]. Knapik *et al.* considered another method of generating such trees, namely by higher-order (deterministic) recursion schemes that satisfy the constraint of *safety*. A major result in their work is the equi-expressivity of both methods as tree generators. In particular it implies that the equivalence problem for higher-order *safe* recursion schemes is interreducible to the problem of decidability of language equivalence between deterministic higher-order PDA.

An alternative approach was developed by Caucal who introduced in [Cau02] 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 [CW03]), which goes from trees to graphs. He showed that the tree hierarchy coincides with the trees generated by safe schemes.

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

A partial answer was later obtained by Knapik, Niwiński, Urzyczyn and Walukiewicz who proved that order-2 homogeneously-typed (but not necessarily safe) recursion schemes are equi-expressive with a variant class of order-2 pushdown automata called *panic automata* [KNUW05].

Finally, Hague, Murawski, Ong and Serre gave a complete answer to the question in [HMOS08]. They introduced a new kind of higher-order pushdown automata, which generalises *pushdown automata with links* [AMO05], or equivalently panic automata, to all finite orders, called *collapsible pushdown automata* (CPDA), in which every symbol in the stack has a link to a (necessarily lower-ordered) stack situated somewhere below it. A major result of their paper is that for every $n \geqslant 0$, order-n recursion schemes and order-n CPDA are equi-expressive as generators of trees.

2.4 Decidability of Monadic Second Order Logic

This quest of finding an alternative description of those trees generated by recursion schemes was led in parallel with the study of the decidability of the model-checking problem for the monadic second order logic (MSO) and the modal μ-calculus (see [Tho97; AN01; GTW02; FGW07] for background about these logics and connections with finite automata and games). Decidability of the MSO theories of trees generated by safe schemes was established by Knapik, Niwiński and Urzyczyn [KNU02] and independently by Caucal [Cau02] (who actually proved a stronger decidability result that holds on graphs as well); the decidability for order-2 (possibly unsafe) schemes followed from [KNUW05] and was obtained thanks to the equi-expressivity with panic automata (see also [AMO05] for an equivalent result).

In 2006, Ong showed decidability of μ-calculus (hence, MSO) for arbitrary recursion schemes [Ong06], and established that this problem is n-EXPTIME complete. This result was obtained using tools from innocent game semantics (in the sense of Hyland and Ong [HO00]) and does not rely on an equivalent automata model for generating trees.

Thanks to their equi-expressivity result, Hague *et al.* provided an alternative proof of the MSO

decidability for schemes. Indeed, thanks to the equi-expressivity between schemes and CPDA together with the well-known connections between MSO model-checking (for trees) and parity games, the model-checking problem for schemes is interreducible to the problem of deciding the winner in a two-player perfect information turn-based parity game played over the labelled transition system (*i.e.* transition graph) associated with a CPDA. They extended the techniques and results of Walukiewicz (for pushdown games) [Wal01], Cachat (for higher-order pushdown) [Cac03b] (see also [CMHOS08] for a more precise study on higher-order pushdown games) and the one from Knapik *et al.* [KNUW05]. These techniques were extended by Broadbent, Carayol, Ong and Serre to establish stronger results on schemes, in particular closure under MSO marking [BCOS10], and later by Carayol and Serre to prove that recursion schemes enjoy the effective MSO selection property [CS12a].

Some years later, following initial ideas by Aehlig [Aeh06] and Kobayahsi [Kob09b], Kobayashi and Ong [KO09] gave another proof of the decidability of MSO. The proof consists in showing that one can associate, with any scheme and formula, a typing system (based on intersection types) such that the scheme is typable in this system if and only if the formula holds. Typability is then reduced to solving a parity game.

In [SW11], Salvati and Walukiewicz used Krivine machines [Kri07] as an abstract model to represent the sequence of rewriting of a scheme (actually, the authors work with the equivalent formalism of the λY-calculus). A Krivine machine computes the weak head normal form of a λY-term, using explicit substitutions (called here environments). Then, MSO decidability for scheme was obtained by solving parity games played on configurations of a Krivine machine. They also provide in [SW12] a translation from recursion schemes which is very close to the one that independently obtained by Carayol and Serre in [CS12a].

3 Definitions

3.1 Trees and Terms

Let A be a finite alphabet. A ***tree*** t with directions in A (or simply a tree if A is clear from the context) is a non-empty prefix-closed subset of A^*. Elements of t are called ***nodes*** and ε is called the ***root*** of t. For any node $u \in t$ and any direction $a \in A$, we refer to $u \cdot a$, when it belongs to t, as the ***a-child*** of u. A node with no child is a ***leaf***. For any node $u \in t$, the subtree of t rooted at u, denoted t_u is the tree $\{v \in A^* \mid u \cdot v \in t\}$. We denote by $\mathrm{Trees}^\infty(A)$ the set of trees with directions in A.

A ***ranked alphabet*** A is an alphabet that comes together with an arity function, $\varrho : A \to \mathbb{N}$. The ***terms*** built over a ranked alphabet A are those trees with directions $\overrightarrow{A} = \bigcup_{f \in A} \overrightarrow{f}$ where $\overrightarrow{f} = \{f_1, \ldots, f_{\varrho(f)}\}$ if $\varrho(f) > 0$ and $\overrightarrow{f} = \{f\}$ if $\varrho(f) = 0$. For a tree $t \in \mathrm{Trees}^\infty(\overrightarrow{A})$ to be a term, we require, for all nodes u, that the set $A_u = \{d \in \overrightarrow{A} \mid u \cdot d \in t\}$ is empty if and only if u ends with some $f \in A$ (hence $\varrho(f) = 0$) and if A_u is non-empty then it is equal to some $\overrightarrow{f}$ for some $f \in A$. We denote by $\mathrm{Terms}(A)$ the set of terms over A.

For $c \in A$ of arity 0, we denote by c the term $\{\varepsilon, c\}$. For $f \in A$ of arity $n > 0$ and for terms $t_1, \ldots, t_n$, we denote by $f(t_1, \ldots, t_n)$ the term $\{\varepsilon\} \cup \bigcup_{i \in [1,n]} \{f_i\} \cdot t_i$. These notions together with natural graphical representations of terms are illustrated in Figure 3.

Note that our (very general) definition of a tree as a non-empty prefix-closed subset of A^* captures the following one (illustrated in the right part of Figure 3). A ***node-labelled ranked tree*** if a map $t : Dom \to \Sigma$ where Dom is a tree (in the previous sense), Σ is a ranked alphabet of node labels and for all node $u \in Dom$ one has $\varrho(t(u)) = |\{a \mid u \cdot a \in Dom\}|$, *i.e.* the arity of the label

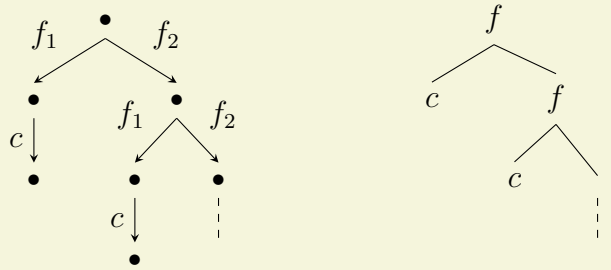


Figure 3 – Two representations of the infinite term $f_2^*\{f_1 c, f_1, \varepsilon\} = f(c, f(c, f(\cdots)))$ over the ranked alphabet $\{f, c\}$ assuming that $\varrho(f) = 2$ and $\varrho(c) = 0$.

of a given node coincides with the number of its children. Indeed, a node-labelled ranked tree is nothing else but a term. In the following we may use any of the representations depending on the context.

3.2 Labelled Transition Systems

A rooted labelled transition system is an edge-labelled directed graph with a distinguished vertex, called the root. Formally, a ***rooted labelled transition system*** $\mathcal{L}$ (Lts for short) is a tuple $\langle D, r, \Sigma, (\xrightarrow{a})_{a\in\Sigma} \rangle$ where D is a finite or countable set called the ***domain***, $r \in D$ is a distinguished element called the ***root***, Σ is a finite set of ***labels*** and for all $a \in \Sigma$, $\xrightarrow{a} \subseteq D \times D$ is a binary relation on D. Elements of D are called ***vertices***.

For any $a \in \Sigma$ and any pair $(s, t) \in D^2$ we write $s \xrightarrow{a} t$ to indicate that $(s, t) \in \xrightarrow{a}$, and we refer to it as an ***a-transition*** with *source* s and *target* t. For a word $w = a_1 \cdots a_n \in \Sigma^*$, we define a binary relation $\xrightarrow{w}$ on D by letting $s \xrightarrow{w} t$ (meaning that $(s, t) \in \xrightarrow{w}$) if there exists a sequence $s_0, \ldots, s_n$ of elements in D such that $s_0 = s$, $s_n = t$, and for all $i \in [1, n]$, $s_{i-1} \xrightarrow{a_i} s_i$. These definitions are extended to languages over Σ by taking, for all $L \subseteq \Sigma^*$, the relation $\xrightarrow{L}$ to be the union of all $\xrightarrow{w}$ for $w \in L$.

When considering Lts associated with computational models, it is usual to allow silent (or internal) transitions. The symbol for silent transitions is usually ε but here, to avoid confusion with the empty word, we will instead use λ. Following [Sti00a], we forbid a vertex to be the source of both a silent transition and of a non-silent transition. Formally, an **Lts *with silent transitions*** is an Lts $\langle D, r, \Sigma, (\xrightarrow{a})_{a\in\Sigma} \rangle$ whose set of labels contains a distinguished symbol denoted $\lambda \in \Sigma$ and such that for all $s \in D$, if s is the source of a λ-transition, then s is not the source of any a-transition with $a \neq \lambda$. We denote by Σ_λ the set $\Sigma \backslash \{\lambda\}$ of non-silent transition labels. For all words $w = a_1 \cdots a_n \in \Sigma_\lambda^*$, we denote by $\xRightarrow{w}$ the relation $\xrightarrow{L_w}$ where $L_w \stackrel{def}{=} \lambda^* a_1 \lambda^* \cdots \lambda^* a_n \lambda^*$ is the set of words over Σ obtained by inserting arbitrarily many occurrences of λ in w.

An Lts (with silent transitions) is said to be ***deterministic*** if for all vertices s, t_1 and t_2 in D and all label a in Σ, if $s \xrightarrow{a} t_1$ and $s \xrightarrow{a} t_2$ then $t_1 = t_2$.

From now on, we *always* assume that the Lts we consider are *deterministic*.

We associate a tree to every Lts with silent transitions $\mathcal{L}$, denoted $\mathrm{Tree}(\mathcal{L})$, with directions in Σ_λ, reflecting the possible behaviours of $\mathcal{L}$ starting from the root. For this we let $\mathrm{Tree}(\mathcal{L}) \stackrel{def}{=} \{w \in$

$\Sigma_\lambda^* \mid \exists s \in D, r \overset{w}{\Longrightarrow} s\}$. As $\mathcal{L}$ is deterministic, $\mathrm{Tree}(\mathcal{L})$ is obtained by unfolding the underlying graph of $\mathcal{L}$ from its root and contracting all λ-transitions. Figure 4 presents an Lts with silent transitions together with its associated tree $\mathrm{Tree}(\mathcal{L})$.

As examplified in Figure 4, the tree $\mathrm{Tree}(\mathcal{L})$ does not reflect the diverging behaviours of $\mathcal{L}$ (*i.e.* the ability to perform an infinite sequence of silent transitions). For instance in the Lts of Figure 4, the vertex s diverges whereas the vertex t does not. A more informative tree can be defined in which diverging behaviours are indicated by a $\bot$-child for some fresh symbol $\bot$. This tree, denoted $\mathrm{Tree}^\bot(\mathcal{L})$, is defined by letting $\mathrm{Tree}^\bot(\mathcal{L}) \overset{def}{=} \mathrm{Tree}(\mathcal{L}) \cup \{w\bot \in \Sigma_\lambda^*\bot \mid \forall n \geqslant 0, \exists s_n \text{ s.t. } r \overset{w\lambda^n}{\Longrightarrow} s_n\}$.

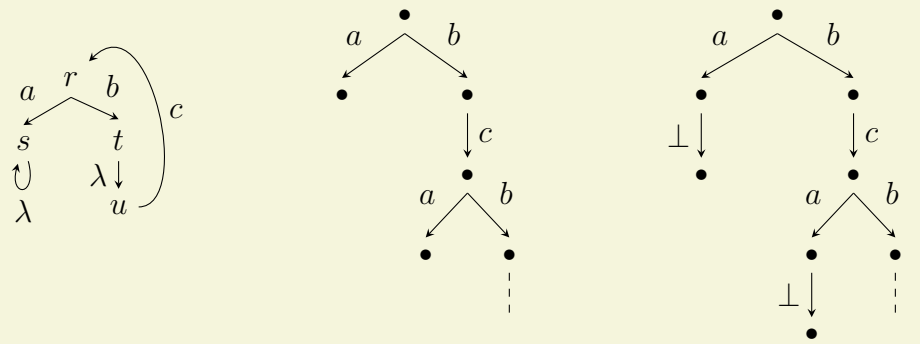


Figure 4 – An Lts $\mathcal{L}$ with silent transitions of root r (on the left), the tree $\mathrm{Tree}(\mathcal{L})$ (in the center) and the tree $\mathrm{Tree}^\bot(\mathcal{L})$ (on the right).

3.3 Simply Typed Terms

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

Intuitively, the *base* (or *ground*) type o corresponds to base elements (such as `int` in ML). An arrow type $\tau_1 \to \tau_2$ corresponds to a function taking an argument of type τ_1 and returning an element of type τ_2 (think *e.g.* to the function $x \mapsto x^2$ of type `int` $\to$ `int` in ML). Even if there are no specific types for functions taking more than one argument, those functions are represented in their curried form. Indeed, a function taking two arguments of type o and returning a value of type o, in its curried form, has the type $o \to o \to o = o \to (o \to o)$; intuitively the function only takes its first argument and returns a function expecting the second argument and returning the desired result.

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

Example 4

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

Let X be a set of typed symbols. For every symbol $f \in X$, and every type τ, we write $f : \tau$ to mean that f has type τ. The set of ***applicative terms*[3] *of type τ generated from X***, denoted $\mathrm{Terms}_\tau(X)$, is defined by induction over the following rules. If $f : \tau$ is an element of X then $f \in \mathrm{Terms}_\tau(X)$; if $s \in \mathrm{Terms}_{\tau_1 \to \tau_2}(X)$ and $t \in \mathrm{Terms}_{\tau_1}(X)$ then the applicative term obtained by applying t to s, denoted $s\,t$, belongs to $\mathrm{Terms}_{\tau_2}(X)$. For every applicative term t, and every type τ, we write $t : \tau$ to mean that t is an applicative term of type τ. ***Ground terms*** are those of type o. By convention, the application is considered to be left-associative, thus we write $t_1 t_2 t_3$ instead of $(t_1 t_2) t_3$. The size $|t|$ of a term t is defined inductively by letting $|f| = 1$ if $f \in X$ and $|t_1 t_2| = |t_1| + |t_2|$.

Example 5

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

The set of subterms of t, denoted $\mathrm{Sub}(t)$, is inductively defined by $\mathrm{Sub}(f) = \{f\}$ for $f \in X$ and $\mathrm{Sub}(t_1\,t_2) = \mathrm{Sub}(t_1) \cup \mathrm{Sub}(t_2) \cup \{t_1\,t_2\}$. The subterms of the term $f\,(f\,g)\,c : \mathrm{o}$ in Example 5 are $f\,(f\,g)\,c$, f, $f\,g$, $f\,(f\,g)$, c and g. A less permissive notion is that of ***argument subterms*** of t, denoted $\mathrm{ASub}(t)$, which only keep those subterms that appear as an argument. The set $\mathrm{ASub}(t)$ is inductively defined by letting $\mathrm{ASub}(t_1 t_2) = \mathrm{ASub}(t_1) \cup \mathrm{ASub}(t_2) \cup \{t_2\}$ and $\mathrm{ASub}(f) = \varnothing$ for $f \in X$. In particular if $t = F t_1 \cdots t_n$, $\mathrm{ASub}(t) = \cup_{i=1}^{n} (\mathrm{ASub}(t_i) \cup \{t_i\})$. The argument subterms of $f\,(f\,g)\,c : \mathrm{o}$ are $f\,g$, c and g. In particular, for all terms t, one has $|\mathrm{ASub}(t)| < |t|$.

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

***Remark* 2.** *In the following, we will simply write term instead of applicative term and denote by* $\mathrm{Terms}(X)$ *the set of applicative terms of ground type over X. It should be clear from the context if we are referring to applicative terms over a typed alphabet or terms over a ranked alphabet. Of course, a ranked alphabet A can be seen as a typed alphabet by assigning to every symbol f of A the type $\underbrace{o \to \cdots \to o}_{\varrho(f)} \to o$. In particular, every symbol in A has order 0 or 1. The finite terms over A (seen as a ranked alphabet) are in bijection with the applicative ground terms over A (seen as a typed alphabet).*

4 Higher-Order Recursion Schemes

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

[3] which should not be confused with terms over a ranked alphabet (cf. Remark 2 below).

Definition 1 (Higher-Order Recursion Schemes)

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

- A is a ranked alphabet of ***terminals*** and $\bot$ is a distinguished terminal symbol of arity 0 (and hence of ground type) that does not appear in any production rule,
- N is a finite set of typed ***non-terminals***; we use upper-case letters $F, G, H, \ldots$ to range over non-terminals,
- $Z \in N$ is a distinguished ***initial symbol*** of type o which does not appear in any right-hand side of a production rule,
- $\mathcal{R}$ is a finite set of ***production rules***, one for each non-terminal $F : (\tau_1, \cdots, \tau_n, o)$, of the form

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

where the x_i are distinct variables with $x_i : \tau_i$ for $i \in [1, n]$ and e is a ground term in $\mathrm{Terms}((A \backslash \{\bot\}) \cup (N \backslash \{Z\}) \cup \{x_1, \ldots, x_n\})$. 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.

4.1 Rewriting System Associated with a Recursion Scheme

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

The term $M[t/x]$ obtained by replacing a variable $x : \tau$ by a term $t : \tau$ over $A \cup N$ in a term M over $A \cup N \cup V$ is defined[4] by induction on M by taking $\varphi[t/x] = \varphi$ for $\varphi \neq x \in A \cup N \cup V$, $x[t/x] = t$ and $(t_1\, t_2)[t/x] = t_1[t/x]\, t_2[t/x]$.

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

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

Example 6

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

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

Figure 5 depicts the first rewriting steps of $\to_{\mathcal{S}}$ starting from the initial symbol Z.

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

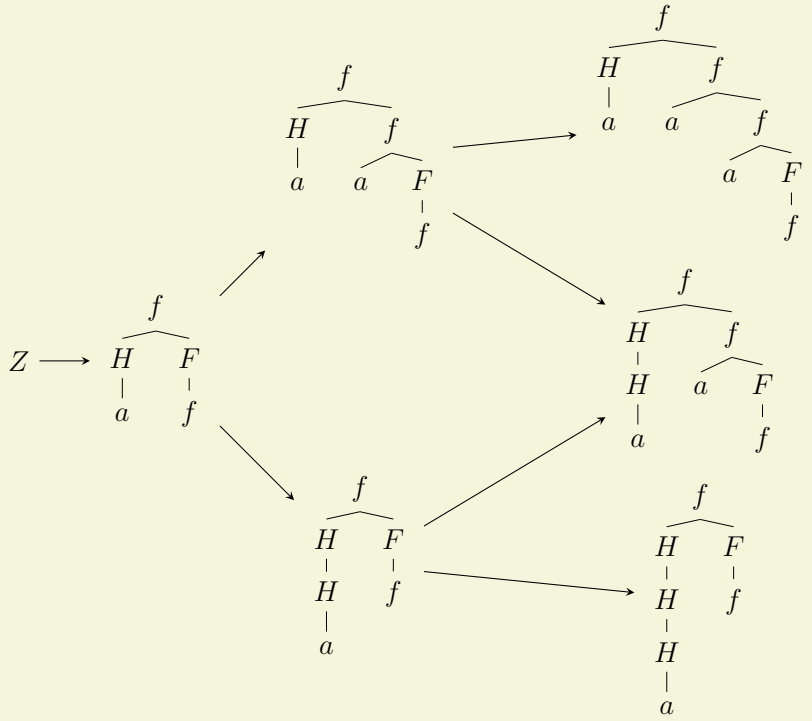


Figure 5 – First rewriting steps of $\rightarrow_{\mathcal{S}}$ for $\mathcal{S}$ from Example 6

As illustrated above, the relation $\rightarrow_{\mathcal{S}}$ is confluent, *i.e.* for all ground terms t,t_1 and t_2, if $t \rightarrow_{\mathcal{S}}^* t_1$ and $t \rightarrow_{\mathcal{S}}^* t_2$ (here $\rightarrow_{\mathcal{S}}^*$ denotes the transitive closure of $\rightarrow_{\mathcal{S}}$), then there exists t' such that $t_1 \rightarrow_{\mathcal{S}}^* t'$ and $t_2 \rightarrow_{\mathcal{S}}^* t'$. The proof of this statement is similar to proof of the confluence of the λ-calculus (See *e.g.* [Bar84]).

4.2 Value Tree of a Recursion Scheme

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

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

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

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

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

The set of terms over A together with $\preccurlyeq$ form a *directed complete partial order*. Meaning that any directed subset D of Terms(A) (*i.e.* D is not empty and for all $x, y \in D$, there exists $z \in D$ such that $x \preccurlyeq z$ and $y \preccurlyeq z$) admits a *supremum*, denoted sup D.

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

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

Example 7

The value tree of the recursion scheme $\mathcal{S}$ of Example 6 is depicted in Figure 6.

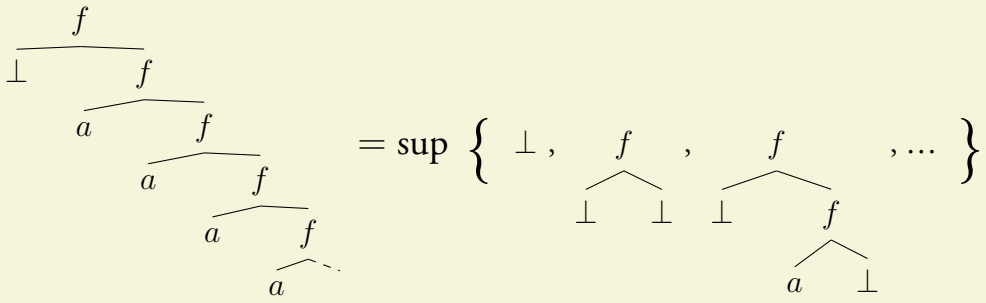


Figure 6 – The value tree of the recursion scheme $\mathcal{S}$ of Example 6

***Remark* 3.** *The relation* $\to_{\mathcal{S}}$ *is* **unrestricted** *in the sense that any ground subterm starting with a non-terminal can be rewritten. A more constrained rewriting policy referred to as* **outermost-innermost (OI)** *only allows to rewrite a ground non-terminal subterm if it is not below any non-terminal symbols (*i.e. *it is outermost) [Dam82]. In term of programming languages it corresponds to the* call-by-name *evaluation policy. The corresponding rewriting relation is denoted* $\to_{\mathcal{S},OI}$*. Note that using* $\to_{\mathcal{S},OI}$ *instead of* $\to_{\mathcal{S}}$ *does not change the value tree of the scheme,* i.e. $\sup\{\, t^{\perp} \mid Z \to_{\mathcal{S}}^{*} t \,\} = \sup\{\, t^{\perp} \mid Z \to_{\mathcal{S},OI}^{*} t \,\}$.

Another rewriting policy referred to as **innermost-outermost (IO)** *only allows to rewrite a ground non-terminal subterm if this subterm does not contain a ground non-terminal as subterm (*i.e. *it is innermost) [Dam82]. In term of programming languages it corresponds to the* call-by-value *evaluation policy. The corresponding rewriting relation is denoted* $\to_{\mathcal{S},IO}$*. Note that using* $\to_{\mathcal{S},IO}$ *instead of* $\to_{\mathcal{S}}$ *may change the value tree of the scheme. Indeed, consider as an example the recursion scheme* $\mathcal{S}'$ *obtained from the scheme* $\mathcal{S}$ *in Example 6 by replacing its first production rule by the following two rules:*

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

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

See [Had12; Had13] for comparisons and transformations between the OI and IO policy.

4.3 Labelled Recursion Schemes

A labelled recursion scheme is a recursion scheme without terminal symbols but whose productions are labelled by letters from a finite alphabet. This slight variation in the definition allows to associate an LTS to every labelled recursion scheme.

Definition 2 (Labelled recursion scheme) [CS12a]

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

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

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

where $a \in \Sigma \backslash \{\bot\}$, $F : (\tau_1, \cdots, \tau_n, o) \in N$, the x_i are distinct variables, each x_i is of type τ_i, and e is a ground term over $(N \backslash \{Z\}) \cup \{ x_1, \ldots, x_n \}$.

In addition, we require that there is at most one production rule starting with a given non-terminal and labelled by a given symbol.

The order of a scheme is the highest order (of the types) of its non-terminals.

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

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

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

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

Example 8

Consider the labelled scheme with labels $\{\lambda, f_1, f_2, a\}$, non terminals $Z, X, \overline{a} : o$, $H : (o, o)$, $\overline{f} : (o, o, o)$ and $F : ((o, o, o), o)$, initial symbol Z and production rules as above

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

Then the tree generated is the same as the one generated by the scheme of Example 6. See Figure 7 for the associated LTS.

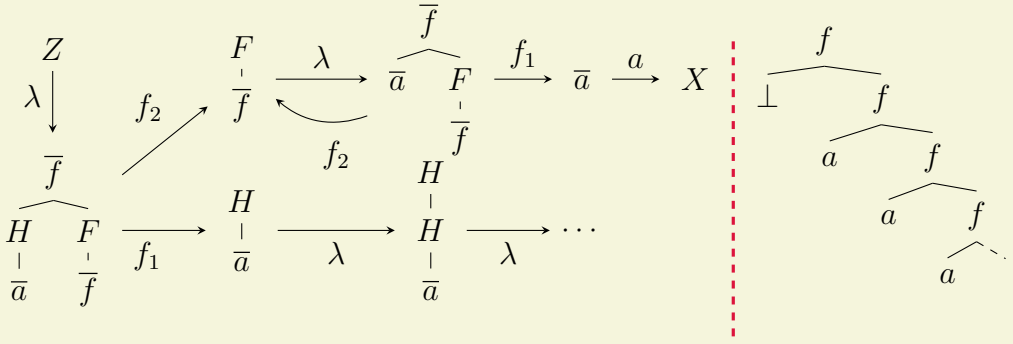


Figure 7 – The LTS associated with the labelled recursion of Example 8.

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

Theorem 3 [CS12a]

The recursion schemes and the labelled recursion schemes generate the same terms. Moreover the translations are linear and preserves order and arity.

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

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

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

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

■

5 Some Examples of Trees Generated by Schemes

In this section, we provide some examples of trees defined by labelled recursion schemes. Given a language L over Σ, we denote by $\mathrm{Pref}(L) = \{u \mid \exists v \in L \text{ s.t. } u \sqsubseteq v\}$ the tree in $\mathrm{Trees}^{\infty}(\Sigma)$ containing all prefixes of words in L. Note that in general $\mathrm{Pref}(L)$ does not defined a ranked tree. In the forthcoming examples we always consider trees of the form $\mathrm{Pref}(L)$ for some L and explain how to obtain it with a labelled recursion scheme. Then we present a recursion scheme that generates a tree whose branch language (*i.e.* the set of words read along the branches) is L (note that such a tree is not unique as one could flip subtrees without changing the branch language).

5.1 The Tree $\mathrm{Pref}(\{a^n b^n c \mid n \geqslant 1\})$

Let us start with the tree T_0 corresponding to the deterministic context-free language $\mathrm{Pref}(\{a^n b^n c \mid n \geqslant 1\})$. As it is the case for all prefix-closed deterministic context-free languages (see [Cou78a; Cou78b] or Theorem 8 at order 1), T_0 is generated by an order-1 scheme.

5.1.1 Generating T_0 with a Labelled Recursion Scheme

The tree T_0 is generated by the following labelled recursion scheme $\mathcal{S}_0'$.

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

with $Z, X, Y : o$ and $H, B : o \to o$. The LTS associated with $\mathcal{S}_0'$ is given in Figure 8.

5.1.2 Generating A Tree with Branch Language $\{a^n b^n c \mid n \geqslant 1\}$ with a Recursion Scheme

We now give a recursion scheme that generates a tree with branch language $\{a^n b^n c \mid n \geqslant 1\}$. For this it suffices to consider the order-1 recursion scheme $\mathcal{S}_0$ with non-terminals $\{Z : o,\ F : o \to o\}$, variable $x : o$, terminals $A = \{a, b, c\}$ of respective arities 2, 1 and 0, and the following production rules:

$$\begin{array}{rcl} Z & \to & F\,c \\ F\,x & \to & a\,(F\,(b\,x))\,(b\,x) \end{array}$$

The first steps of a derivation of $\mathcal{S}_0$ are given in Figure 9.

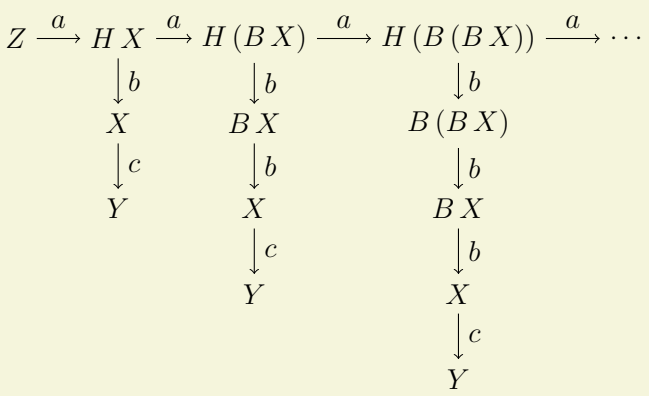


Figure 8 – The LTS associated with the labelled recursion scheme $\mathcal{S}_0'$ generating $T_0 = \mathrm{Pref}(\{a^n b^n c \mid n \geqslant 1\})$

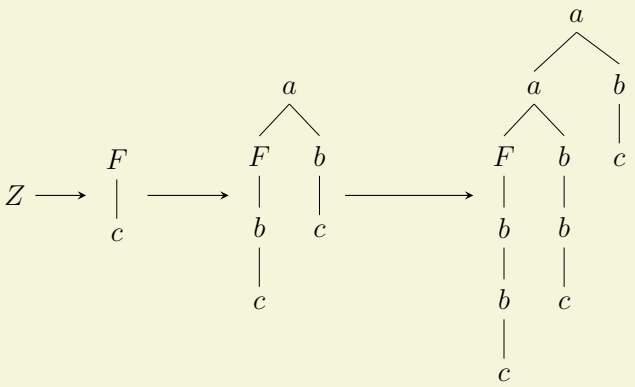


Figure 9 – First rewriting steps of $\rightarrow_{\mathcal{S}_0}$ for $\mathcal{S}_0$ generating a tree with branch language $\{a^n b^n c \mid n \geqslant 1\}$

5.2 The Tree $\mathrm{Pref}(\{a^n b^n c^n d \mid n \geqslant 1\})$

We consider now the tree $T_1 = \mathrm{Pref}(\{a^n b^n c^n d \mid n \geqslant 1\})$ and we describe a labelled recursion scheme and a recursion scheme generating T_1. Both have order-2 which cannot be lowered as only prefix-closed deterministic context-free languages are generated by order-1 schemes (see [Cou78a; Cou78b] or Theorem 8 at order 1).

5.2.1 Generating the Tree with a Labelled Recursion Scheme

Consider the order-2 labelled recursion scheme $\mathcal{S}_1'$ given by:

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

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

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

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

5.2.2 Generating a Tree with Branch Language $\{a^n b^n c^n d \mid n \geqslant 1\}$ with a Recursion Scheme

To generate a tree with branch language $\{a^n b^n c^n d \mid n \geqslant 1\}$ it suffices to consider the order-2 recursion scheme $\mathcal{S}_1$ with non-terminals $\{Z : o,\ F : (o \to o) \to (o \to o) \to o,\ C_p : (o \to o) \to (o \to o) \to o \to o\}$, variables $\{x : o,\ \varphi, \psi : o \to o\}$, terminals $A = \{a, b, c, d\}$ of arity 2 (a), 1 (b and c) and 0 (d), and the following production rules:

$$\begin{array}{rcl} Z & \to & F\, b\, c \\ F\, \varphi\, \psi & \to & a\,(F\,(C_p\, b\, \varphi)\,(C_p\, c\, \psi))\,(\varphi\,(\psi\, d)) \\ C_p\, \varphi\, \psi\, x & \to & \varphi\,(\psi\, x) \end{array}$$

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 a derivation of $\mathcal{S}_1$ are given in Figure 10.

5.3 The Tree Pref$(\{a^n c b^{2^n} d \mid n \geqslant 1\})$

Following the same ideas as for T_1, we build order-2 (labelled) schemes that define the tree $T_{\text{exp}} = \text{Pref}(\{a^n c b^{2^n} d \mid n \geqslant 1\})$.

5.3.1 Generating the Tree with a Labelled Recursion Scheme

Consider the order-2 labelled recursion scheme $\mathcal{S}'_{\text{exp}}$ given by:

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

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

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

5.3.2 Generating a Tree with Branch Language $\{a^n c b^{2^n} d \mid n \geqslant 1\}$ with a Recursion Scheme

Let $\mathcal{S}_{\text{exp}}$ be the order-2 recursion scheme with non-terminals $\{Z : o,\ F : (o \to o) \to o,\ C_p : (o \to o) \to (o \to o) \to o \to o\}$, variables $\{x : o,\ \varphi, \psi : o \to o\}$, terminals $A = \{a, b, c, d\}$ of arity 2 (a), 1 (b and c) and 0 (d), and the following production rules:

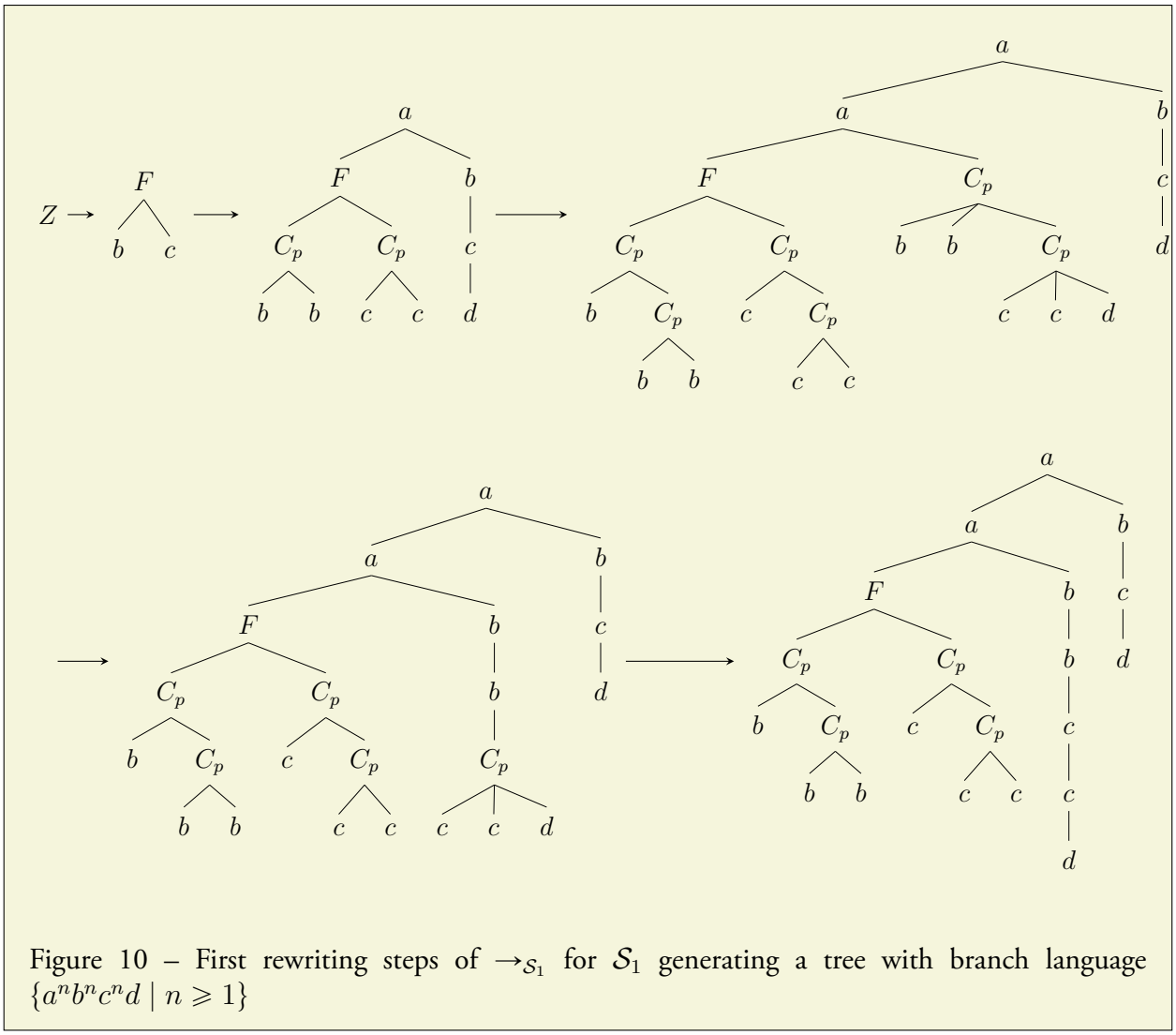


Figure 10 – First rewriting steps of $\rightarrow_{\mathcal{S}_1}$ for $\mathcal{S}_1$ generating a tree with branch language $\{a^n b^n c^n d \mid n \geqslant 1\}$

$$\begin{aligned} Z &\to F(C_p\, b\, b) \\ F\,\varphi &\to a\,(F\,(C_p\,\varphi\,\varphi))\,(c\,(\varphi\, d)) \\ C_p\,\varphi\,\psi\,x &\to \varphi\,(\psi\, x) \end{aligned}$$

Again, 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 a derivation of $\mathcal{S}_{\text{exp}}$ are given in Figure 11.

5.4 The Trees Corresponding to the Tower of Exponentials of Height k.

At order $k+1 \geqslant 1$, we can define the tree $T_{\text{exp}_k} = \text{Pref}(\{a^n\, c\, b^{\text{exp}_k(n)} \mid n \geqslant 1\})$ where we let $\text{exp}_0(n) = n$ and $\text{exp}_{k+1}(n) = 2^{exp_k(n)}$ for $k \geqslant 0$. We illustrate the idea by giving an order-3 labelled recursion scheme generating $T_{\text{exp}_2} = \text{Pref}(\{a^n\, c\, b^{2^{2^n}} \mid n \geqslant 1\})$.

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

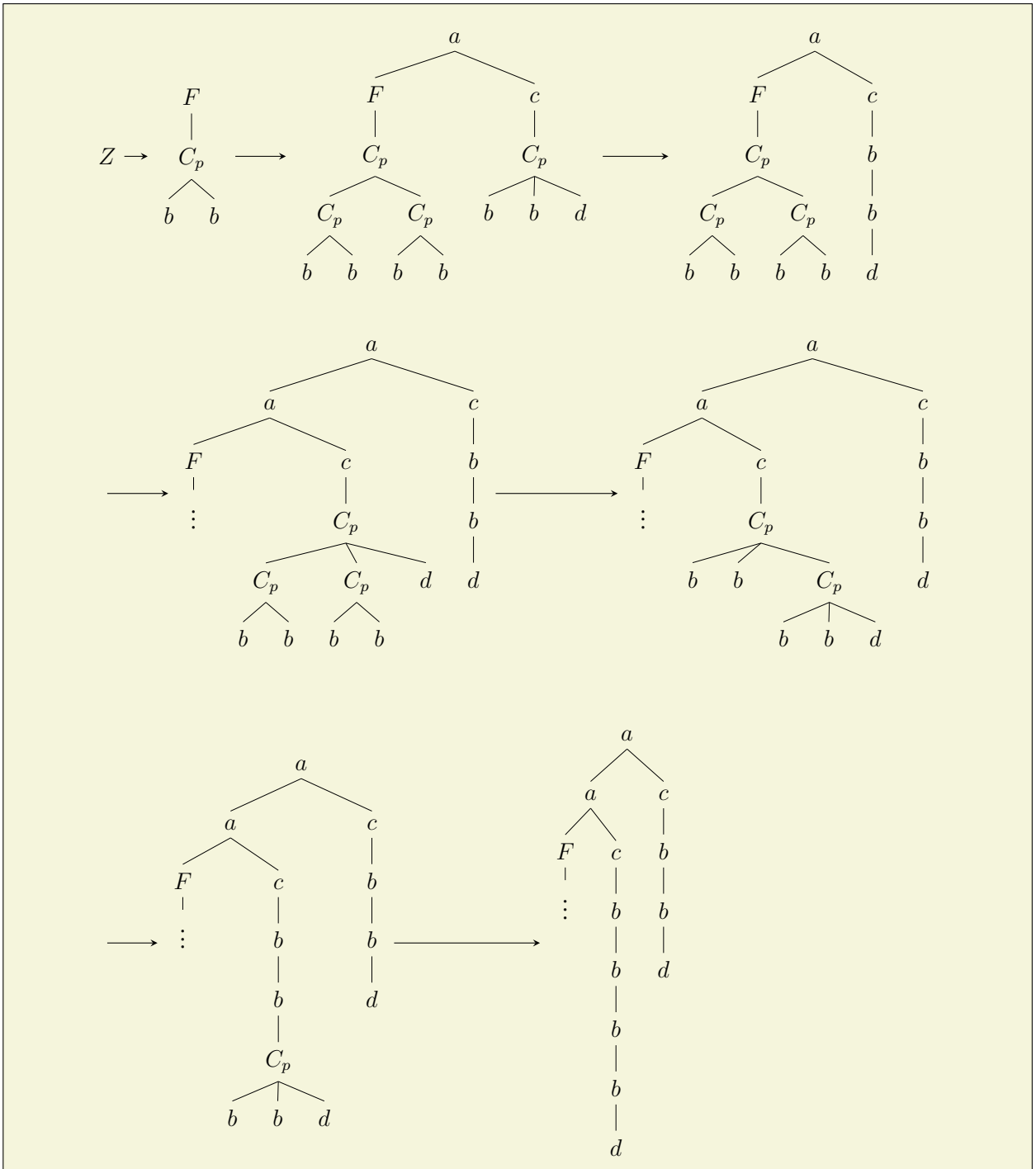


Figure 11 – First rewriting steps of $\rightarrow_{\mathcal{S}_{\text{exp}}}$ for $\mathcal{S}_{\text{exp}}$ generating a tree with branch language $\{a^n bc^{2^n} d \mid n \geqslant 1\}$

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

For all $n \geqslant 0$, we have:

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

5.5 The Tree of the Urzyczyn Language

All schemes presented so far satisfied a syntactic restriction, called the safety condition, that will be discussed in Section 8.6.2. Paweł Urzyczyn conjectured that (a slight variation) of the tree described below, though generated by an order-2 scheme, could not be generated by any order-2 scheme satisfying the safety condition. This conjecture was proved by Paweł Parys in [Par11] and later strengthened as he proved in [Par12] that, even if one allows an arbitrary order, no safe scheme can generate this tree.

The tree T_U has directions in $\{(,), \star\}$. A word over $\{(,)\}$ is ***well-bracketed*** if it has as many opening brackets as closing brackets and if, for every prefix, the number of opening brackets is greater than the number of closing brackets.

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

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

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

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

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

The scheme can be revisited as follows:

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

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

6 Collapsible Pushdown Automata

6.1 Higher-Order Stacks and their Operations

Higher-order pushdown automata were introduced by Maslov [Mas76] as a generalisation of pushdown automata. Recall first that a (order-1) pushdown automaton is a machine with a finite control together with an auxiliary storage given by a (order-1) stack whose symbols are taken from a finite alphabet. A higher-order pushdown automaton is defined in a similar way, except that it uses a higher-order stack as auxiliary storage. Intuitively, an order-n stack is a stack whose base symbols are order-$(n-1)$ stacks, with the convention that order-1 stacks are just stacks in the classical sense.

Fix a finite stack alphabet Γ and a distinguished ***bottom-of-stack symbol*** $\bot \notin \Gamma$. An order-1 stack is a sequence $\bot, a_1, \ldots, a_\ell \in \bot\Gamma^*$ which is denoted $[\bot a_1 \ldots a_\ell]_1$. An ***order-k stack*** (or a *k-stack*), for $k > 1$, is a non-empty sequence $s_1, \ldots, s_\ell$ of order-$(k-1)$ stacks which is written $[s_1 \ldots s_\ell]_k$. For convenience, we may sometimes see an element $a \in \Gamma$ as an order-0 stack, denoted $[a]_0$. We denote by Stacks_k the set of all order-k stacks and $\text{Stacks} = \bigcup_{k \geqslant 1} \text{Stacks}_k$ the set of all higher-order stacks. The ***height*** of the stack s denoted $|\, s \,|$ is simply the length of the sequence. We denote by $\text{ord}(s)$ the order of the stack s. We also use the notation $\bot_n$ for all $n \geqslant 0$ by letting $\bot_0 = \bot$ and $\bot_{k+1} = [\bot_k]_{k+1}$.

Example 9

The stack

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

is an order-3 stack of height 2.

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

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

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

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

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

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

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

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

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

Example 10

Let s be the order-3 stack of Example 9. Then we have

$$top_3(s) = [[\bot baa]_1[\bot bc]_1[\bot bab]_1]_2$$

and

$$pop_3(s) = [[[\bot baac]_1[\bot bb]_1[\bot bcc]_1[\bot cba]_1]_2]_3$$

Note that $pop_3(pop_3(s))$ is undefined.

We also have that

$$push_2(pop_3(s)) = [[[\bot baac]_1[\bot bb]_1[\bot bcc]_1[\bot cba]_1[\bot cba]_1]_2]_3$$

and

$$push_1^c(pop_3(s)) = [[[\bot baac]_1[\bot bb]_1[\bot bcc]_1[\bot cbac]_1]_2]_3$$

6.2 Stacks With Links and their Operations

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

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

Formally, ***order-n stacks with links*** over alphabet Γ are defined as order-n stacks [5] over alphabet $\Gamma \cup \Gamma \times \{2, \cdots, n\} \times \mathbb{N}$.

Example 11

The stack

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

is an order-3 stack with links.

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

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

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

In addition to the previous operations pop_i, $push_i$ and $push_1^a$, we introduce two extra operations: one to create links, and the other to collapse the stack by following a link.

Link creation is made when pushing a new stack symbol, and the target of an ℓ-link is always the $(\ell-1)$-stack below the topmost one. Formally, we define $\boldsymbol{push_1^{a,\ell}(s)} = push_1^{(a,\ell,h)}$ where we let $h = |top_\ell(s)| - 1$ and require that $h > 1$.

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

Finally we add to extra operations: the identity operation $\boldsymbol{id}$, *i.e.* $id(s) = s$ for all stacks, and $\boldsymbol{rew_1^a}$ that rewrites the top symbol (without changing the target of the link). Formally, we define $rew_1^a(s) = push_1^{(a,\ell,h)}(pop_1(s))$ where we let $top_1(s) = (b, \ell, h)$ for some $b \in \Gamma$.

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

Example 12

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

$$\begin{aligned}
push_1^{b,2}(s) &= [[[\bot\, a]_1]_2\ [[\bot]_1[\bot\, a\, b]_1]_2]_3 \\
rew_1^c\,(push_1^{b,2}(s)) &= [[[\bot\, a]_1]_2\ [[\bot]_1[\bot\, a\, c]_1]_2]_3 \\
collapse\,(push_1^{b,2}(s)) &= [[[\bot\, a]_1]_2\ [[\bot]_1]_2]_3 \\
\underbrace{push_1^{c,3}(push_1^{b,2}(s))}_{\theta} &= [[[\bot\, a]_1]_2\ [[\bot]_1[\bot\, a\, b\, c]_1]_2]_3.
\end{aligned}$$

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

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

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

We have

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

6.3 Higher-Order Pushdown Automata and Collapsible Pushdown Automata

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

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

In the special case where $\mathit{collapse} \notin \delta(q,\gamma,a)$ for all $q \in Q$, $\gamma \in \Gamma$ and $a \in \Sigma$, $\mathcal{A}$ is called a ***higher-order pushdown automaton*** (***HOPDA***). Note that in this case any operation $\mathit{push}_1^{a,\ell}$ can safely be replaced by push_1^a.

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

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

Example 13

Consider the following 2-HOPDA $\mathcal{A} = \langle \{a,b,c,d,\lambda\}, \{\bot,\alpha\}, \{q_a,q_b,q_c,q_d,\widetilde{q}_a,\widetilde{q}_b,\widetilde{q}_c\}, \delta, \widetilde{q}_a \rangle$ with δ as follows:

- $\delta(\widetilde{q}_a, \bot, a) = \delta(q_a, \alpha, a) = (q_a, \mathit{push}_1^\alpha)$;
- $\delta(q_a, \alpha, b) = (\widetilde{q}_b, \mathit{push}_2)$;
- $\delta(\widetilde{q}_b, \alpha, \lambda) = \delta(q_b, \alpha, b) = (q_b, \mathit{pop}_1)$;
- $\delta(q_b, \bot, c) = (\widetilde{q}_c, \mathit{pop}_2)$;
- $\delta(\widetilde{q}_c, \alpha, \lambda) = \delta(q_c, \alpha, c) = (q_c, \mathit{pop}_1)$;
- $\delta(q_c, \bot, d) = (q_d, \mathit{id})$;

Then $\mathcal{L}_\mathcal{A}$ is depicted in Figure 12.

Note that the tree generated by $\mathcal{A}$ is the tree $T_1 = \mathrm{Pref}(\{a^n b^n c^n d \mid n \geqslant 1\})$ considered in Section 5.2.

So far we focused on Lts and associated trees but another (equivalent) way to generate infinite objects (trees and graphs) is to label the elements of the domain of the Lts. This will be later useful when considering so-called CPDA-parity games and model-checking problem.

Let $\mathcal{A} = \langle \Sigma, \Gamma, Q, \delta, q_0 \rangle$ be an n-CPDA and let $\mathcal{L}_\mathcal{A} = \langle D, r, \Sigma, (\xrightarrow{a})_{a\in\Sigma} \rangle$ be the associated Lts. Consider a vertex-labelling function $\kappa : Q \times \Gamma \to A$ where A is a finite set of node labels. Then κ is extended as a function from $D \to A$ by letting $\kappa(q,s) = \kappa(q, \mathit{top}_1(s))$, *i.e.* a configuration is labelled according to its control state and topmost stack element. Hence, if we consider an n-CPDA together with such a vertex-labelling function κ we obtain an infinite (vertex-labelled edge-labelled directed) graph; if the labelling alphabet is ranked and if both the vertex and the edge labellings satisfy some extra natural conditions[6] one obtains a node-labelled ranked tree by first unfolding the infinite graph and then contracting all λ-edges[7].

[6] Namely one requires that any vertex labelled by a symbol of arity k has k neighbours that are reached by edges whose labels are $\{1,\cdots,k\}$.

[7] One also needs to label by $\bot$ those vertices that are the source of an infinite chain of λ edges.

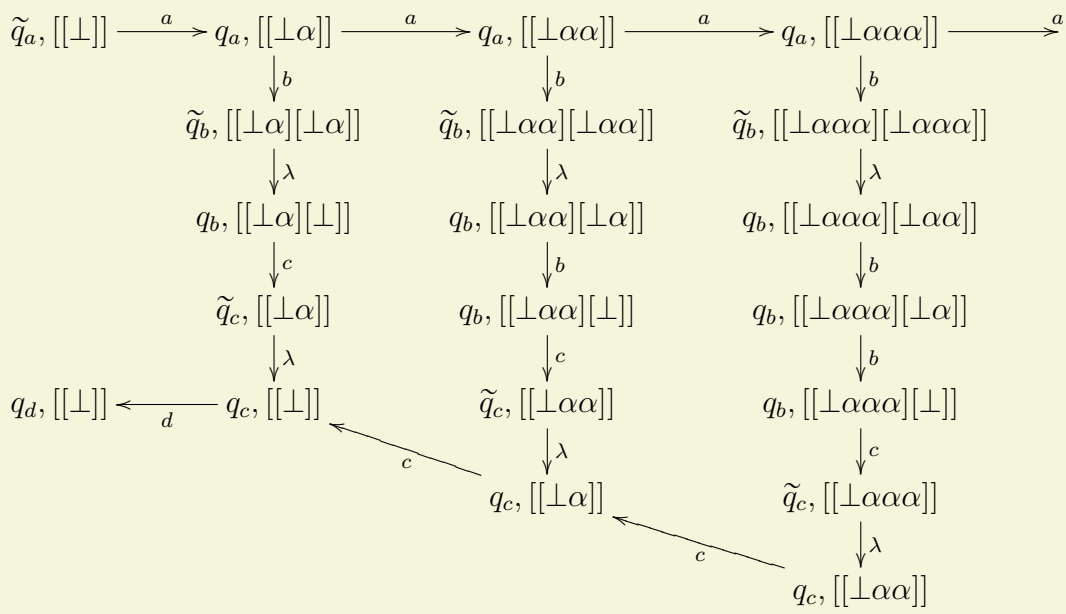


Figure 12 – The LTS $\mathcal{L}_{\mathcal{A}}$ associated with the HOPDA $\mathcal{A}$ of Example 13

When considering parity games, this approach will be useful. Indeed, in this setting, edge labels are not important and focusing on the graph is the same as considering the tree (games are somehow insensible to unfolding and can handle directly the contraction of λ-edges).

The following example revisit Example 13.

Example 14

Consider the following 2-HOPDA $\mathcal{A}' = \langle \{1, 2, \lambda\}, \{\bot, \alpha\}, \{q_a, q_b, q_c, q_d, \widetilde{q}_a, \widetilde{q}_b, \widetilde{q}_c\}, \delta, \widetilde{q}_a \rangle$ with δ as follows:

- $\delta(\widetilde{q}_a, \bot, \lambda) = \delta(q_a, \alpha, 2) = (q_a, \mathit{push}_1^{\alpha})$;
- $\delta(q_a, \alpha, 1) = (\widetilde{q}_b, \mathit{push}_2)$;
- $\delta(\widetilde{q}_b, \alpha, \lambda) = \delta(q_b, \alpha, 1) = (q_b, \mathit{pop}_1)$;
- $\delta(q_b, \bot, 1) = (\widetilde{q}_c, \mathit{pop}_2)$;
- $\delta(\widetilde{q}_c, \alpha, \lambda) = \delta(q_c, \alpha, 1) = (q_c, \mathit{pop}_1)$;
- $\delta(q_c, \bot, 1) = (q_\sharp, id)$;
- $\delta(q_\sharp, \bot, \_)$ is undefined.

Consider the vertex-labelling function κ defined by letting $\kappa(q_a, \_) = a$, $\kappa(q_b, \_) = b$, $\kappa(q_c, \_) = c$, $\kappa(q_d, \_) = d$, where _ stands for any stack symbol. Then $\mathcal{L}_{\mathcal{A}'}$ is depicted in Figure 13 (we indicate into parenthesis the vertex label when exists). It should be clear that if we unfold and contract the λ-edges we retrieve the same infinite ranked tree as previously.

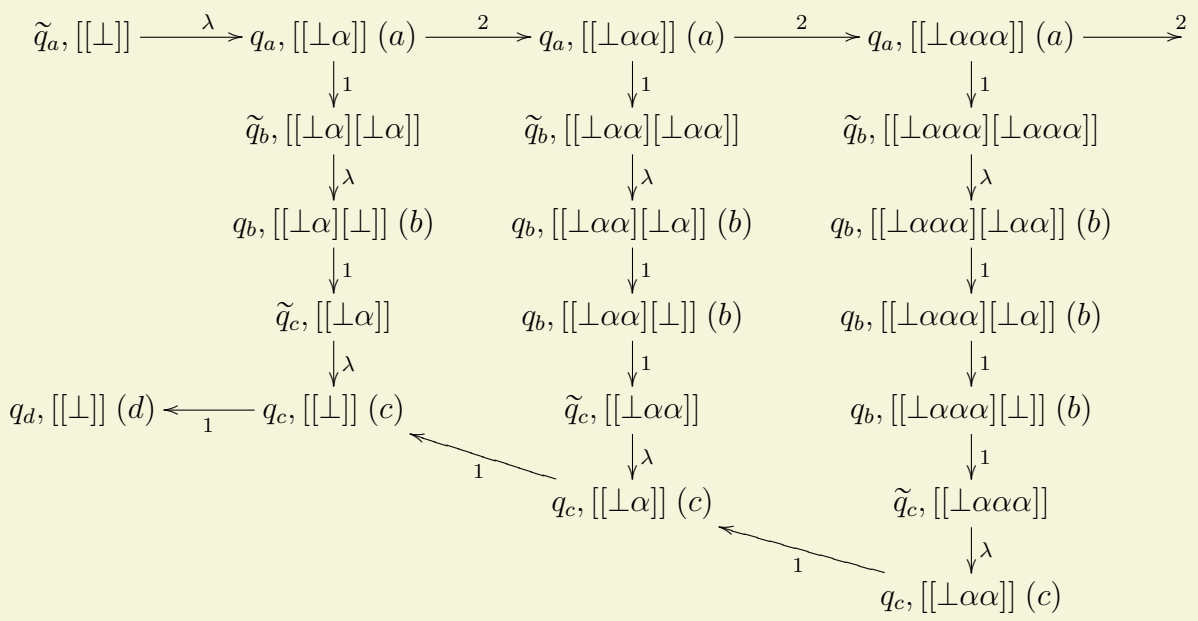


Figure 13 – The Lts $\mathcal{L}_{\mathcal{A}'}$ associated with the HOPDA $\mathcal{A}'$ of Example 14

7 Some Examples of Collapsible Pushdown Automata and Associated Trees

We already saw in Example 14 how to generate the tree $T_1 = \text{Pref}(\{a^n b^n c^n d \mid n \geqslant 1\})$ considered in Section 5.2. We now revisit some examples previously considered in the setting for schemes.

7.1 The Tree $\text{Pref}(\{a^n b^n c \mid n \geqslant 1\})$

This is a simple example as a standard pushdown automaton suffices. We do not formally describe it here but refer the reader to Figure 14.

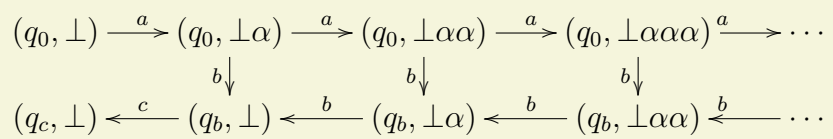


Figure 14 – The Lts $\mathcal{L}_{\mathcal{A}}$ associated with a pushdown automaton generating the tree $\text{Pref}(\{a^n b^n c \mid n \geqslant 1\})$

7.2 The Tree $\text{Pref}(\{a^n c b^{2^n} d \mid n \geqslant 1\})$

The case of the tree $\text{Pref}(\{a^n c b^{2^n} d \mid n \geqslant 1\})$ previously considered in Section 5.3 is more tricky. We do not give a formal description but rather describe the behaviour of the 2-HOPDA used to generate the tree. Our presentation is strongly influenced by the one given by Wöhrle in his PhD Thesis [Wöh05].

The main idea is to use on order-2 stack (without links) to encode a binary counter. The length of the counter is stored on the bottom order-1 stack, the "meaningful" configurations have a stack

which shape is a "triangle" (the 1-stacks in the 2-stack have their length iteratively decreasing by 1) and the value of the counter that is encoded is obtained by reading the topmost symbols of all order-1 stacks (the least important bit being the one from the topmost 1-stack). For instance $[[\bot 000\mathbf{0}]\,[\bot 00\mathbf{0}]\,[\bot 0\mathbf{1}]\,[\bot \mathbf{1}]]$ encodes the number whose binary decomposition is **0011**, *i.e.* 3.

Call q_0 the initial state. Then in q_0 the automaton can do a transition labelled by a after which a 0 is pushed on the stack while the state remains unchanged. In q_0 the automaton can also go to another state q_1 by a transition labelled c while making no modification on the stack. Hence, for any $n \geqslant 0$ one has

$$(q_0, [[\bot]]) \xrightarrow{a^n c} (q_1, [[\bot \underbrace{0 \cdots 0}_{n}]])$$

Then by performing a sequence of λ-transition the stack is transformed into a "triangle" (by performing successively $push_2$ and pop_1 until the top element is $\bot$ where one finally does a pop_2). This ends up in a configuration with state q_2. Hence, one has

$$(q_1, [[\bot \underbrace{0 \cdots 0}_{n}]]) \xrightarrow{\lambda^{2n-1}} (q_2, [[\bot \underbrace{0 \cdots 0}_{n}]\,[\bot \underbrace{0 \cdots 0}_{n-1}] \cdots [\bot 0]])$$

From state q_2 the automaton does a b-transitions followed by a (possibly empty) sequence of λ transitions: it performs pop_2 operations until it ends up in a stack whose top_1 element is 0, then this top_1 element becomes a 1 and the missing part of the "triangle" is then rebuilt in a similar way as previously. When no such element is found, the only possible action is a d-transition and then the automaton is blocked in state q_f. Hence one can do 2^n b-transitions before doing a d-transition.

As an example, this leads to the possible trace in the LTS associated to the automaton:

$$(q_0, [[\bot]]) \xrightarrow{a^3 c} (q_1, [[\bot 000]]) \xrightarrow{\lambda^*} (q_2, [[\bot 00\mathbf{0}]\,[\bot 0\mathbf{0}]\,[\bot \mathbf{0}]]) \xrightarrow{b\lambda^*} (q_2, [[\bot 00\mathbf{0}]\,[\bot 0\mathbf{0}]\,[\bot \mathbf{1}]])$$
$$\xrightarrow{b\lambda^*} (q_2, [[\bot 00\mathbf{0}]\,[\bot 0\mathbf{1}]\,[\bot \mathbf{0}]]) \xrightarrow{b\lambda^*} (q_2, [[\bot 00\mathbf{0}]\,[\bot 0\mathbf{1}]\,[\bot \mathbf{1}]]) \xrightarrow{b\lambda^*} (q_2, [[\bot 00\mathbf{1}]\,[\bot 0\mathbf{0}]\,[\bot \mathbf{0}]])$$
$$\xrightarrow{b\lambda^*} (q_2, [[\bot 00\mathbf{1}]\,[\bot 0\mathbf{0}]\,[\bot \mathbf{1}]]) \xrightarrow{b\lambda^*} (q_2, [[\bot 00\mathbf{1}]\,[\bot 0\mathbf{1}]\,[\bot \mathbf{0}]]) \xrightarrow{b\lambda^*} (q_2, [[\bot 00\mathbf{1}]\,[\bot 0\mathbf{1}]\,[\bot \mathbf{1}]])$$
$$\xrightarrow{b\lambda^* d} (q_f, [[\bot 001]])$$

7.3 The Trees Corresponding to the Tower of Exponentials of Height k.

Recall that in Section 5.4 we showed that order $k+1 \geqslant 1$ schemes permit to define the tree $T_{\exp_k} = \mathrm{Pref}(\{a^n\, c\, b^{\exp_k(n)} \mid n \geqslant 1\})$ where we let $\exp_0(n) = n$ and $\exp_{k+1}(n) = 2^{exp_k(n)}$ for $k \geqslant 0$. The same result holds if we replace schemes by HOPDA: for all $k \geqslant 0$ there is a $(k+1)$-HOPDA that generates $T_{\exp_k}$. Note that the cases $k = 0$ and $k = 1$ were considered in the above examples.

The idea is to generalise the technique of Wöhrle for the case $k = 1$ (see Section 7.2). Recall that the key idea was to generate after performing n a-transitions followed by one c-transition the configuration

$$(q_1, [[\bot \underbrace{0 \cdots 0}_{n}]])$$

and then to simulate a binary counter using n-bits and increment it from 0 to $2^n - 1$.

We illustrate the idea for $k = 2$ (the general case is then easily derived). We follow the same idea except that after performing n a-transitions followed by one c-transition and several λ transition we will end up in a configuration that permits to simulate a binary counter with 2^n bits that is then used in a similar way as previously.

The idea is to work with an order-3 HOPDA and to replace the b-transitions in the automaton for the case $k = 1$ by λ-transition that perform a $push_3$ operation on the stack. Hence after simulating all the previous b-transitions, one ends up in a configuration that contains 2^n order-2 stacks, each of them encoding a counter (whose values ranges from 0 to $2^n - 1$). As an example for $n = 3$:

$$
\begin{aligned}
&(q_0, [[[\bot]]_2]_3) \xrightarrow{a^3c} (q_1, [[[\bot 000]]_2]_3) \xrightarrow{\lambda^*} (q_2, [[[\bot 000]\,[\bot 00]\,[\bot 0]]_2]_3)\\
&\xrightarrow{\lambda^*} (q_2, [[[\bot 000]\,[\bot 00]\,[\bot 0]]_2[[\bot 000]\,[\bot 00]\,[\bot 1]]_2]_3)\\
&\xrightarrow{\lambda^*} (q_2, [[[\bot 000]\,[\bot 00]\,[\bot 0]]_2[[\bot 000]\,[\bot 00]\,[\bot 1]]_2[[\bot 000]\,[\bot 01]\,[\bot 0]]_2]_3)\\
&\dots\\
&\xrightarrow{\lambda^*} (q_2, [[\sigma_0]_2[\sigma_1]_2[\sigma_2]_2[\sigma_3]_2[\sigma_4]_2[\sigma_5]_2[\sigma_6]_2[\sigma_7]_2]_3)
\end{aligned}
$$

Where we let $\sigma_i = [\bot 00 i_2]\,[\bot 0 i_1]\,[\bot i_0]$ where $i_2 i_1 i_0$ is the binary representation of $0 \leqslant i \leqslant 7$.

Now consider the following variant: before doing a $push_3$ in the previous construction one does a $push_2$ followed by a pop_1 followed by a $push_1^0$. Then one has (for readability we omit the $\bot$ symbols from now):

$$(q_0, [[[\bot]]_2]_3) \xrightarrow{a^3 c \lambda^*} (q_2, [[\sigma_0\,[0]]_2[\sigma_1\,[0]]_2[\sigma_2\,[0]]_2[\sigma_3\,[0]]_2[\sigma_4\,[0]]_2[\sigma_5\,[0]]_2[\sigma_6\,[0]]_2[\sigma_7\,[0]]_2]_3)$$

Then by performing a sequence of λ-transition the 3-stack is transformed into a "triangle" (by performing successively $push_3$ and pop_2). As an example for $n = 3$, starting from the initial configuration after performing 3 a-transitions followed by a c-transition and a (large) number of λ-transition one ends up in a configuration whose stack is the following (we adopt a bi-dimensional representation for the 3-stack for ease of reading):

$$
\left[\begin{array}{l}
[\sigma_0\,[\mathbf{0}]]_2\\
[\sigma_0\,[0]]_2[\sigma_1\,[\mathbf{0}]]_2\\
[\sigma_0\,[0]]_2[\sigma_1\,[0]]_2[\sigma_2\,[\mathbf{0}]]_2\\
[\sigma_0\,[0]]_2[\sigma_1\,[0]]_2[\sigma_2\,[0]]_2[\sigma_3\,[\mathbf{0}]]_2\\
[\sigma_0\,[0]]_2[\sigma_1\,[0]]_2[\sigma_2\,[0]]_2[\sigma_3\,[0]]_2[\sigma_4\,[\mathbf{0}]]_2\\
[\sigma_0\,[0]]_2[\sigma_1\,[0]]_2[\sigma_2\,[0]]_2[\sigma_3\,[0]]_2[\sigma_4\,[0]]_2[\sigma_5\,[\mathbf{0}]]_2\\
[\sigma_0\,[0]]_2[\sigma_1\,[0]]_2[\sigma_2\,[0]]_2[\sigma_3\,[0]]_2[\sigma_4\,[0]]_2[\sigma_5\,[0]]_2[\sigma_6\,[\mathbf{0}]]_2\\
[\sigma_0\,[0]]_2[\sigma_1\,[0]]_2[\sigma_2\,[0]]_2[\sigma_3\,[0]]_2[\sigma_4\,[0]]_2[\sigma_5\,[0]]_2[\sigma_6\,[0]]_2[\sigma_7\,[\mathbf{0}]]_2
\end{array}\right]_3
$$

Then it is fairly clear how one encodes in such a configuration a binary counter of 2^n bits that is later used to count the number of b transitions before going to a blocking state, as in the case for $k = 1$ (see Section 7.2).

7.4 A "True" CPDA Example

So far we only gave examples of HOPDA (*i.e.* we did not make use of the *collapse* operation in the transition function). We now consider an example of a 2-CPDA.

Consider the following 2-CPDA $\mathcal{A}_t = \langle \{a, b, c, 2, 1\}, \{\bot, a, b\}, \{q_0, q_1, q_2\}, \delta, q_0 \rangle$ with δ as follows:

- $\delta(q_0, \bot, 2) = \delta(q_0, a, 2) = (q_1, \mathit{push}_2)$;
- $\delta(q_1, \bot, a) = \delta(q_1, a, a) = (q_0, \mathit{push}_1^{a,2})$;
- $\delta(q_1, \bot, b) = \delta(q_1, a, b) = (q_2, \mathit{push}_1^{b,2})$;
- $\delta(q_2, a, 1) = \delta(q_2, b, 1) = (q_2, \mathit{pop}_1)$;
- $\delta(q_2, a, c) = \delta(q_2, b, c) = (q_0, \mathit{collapse})$;

The LTS associated with $\mathcal{A}_t$ is given in Figure 15.

The CPDA $\mathcal{A}_t$ is a "true" one as we will later argue in Theorem 11 that the The LTS associated with $\mathcal{A}_t$ cannot be generated by any HOPDA.

7.5 The Tree of the Urzyczyn Language

In Section 5.5, we considered the tree T_U of the so-called Urzyczyn language. Recall that the language $U \subseteq \{(,), \star\}^*$ is defined as the set of words of the form $w\star^n$ where w is a prefix of a well-bracketed word and n is equal to $|w| - |u| + 1$ where u is the longest suffix of w which is well-bracketed. In other words, n equals 1 if w is well-bracketed, and otherwise it is equal to the index of the last unmatched opening bracket plus one. For instance, the words $()((()) \star\star\star\star$ and $()()()\star$ belong to U. The tree T_U is simply Pref(U).

We define a 2-CPDA $\mathcal{A}_U = \langle \{(,), \star\}, \{\bot, a, \sharp\}, \{q_0, q_1, q_2, q_3\}, \delta, q_0 \rangle$ that generates T_U. We let δ be as follows (and we allow to perform several actions on the stack on a single transition, which can easily be simulated by adding intermediate states; we use _ to denote an arbitrary stack symbol):

- $\delta(q_0, \bot, \lambda) = (q_1, \mathit{push}_1^{\sharp}; \mathit{push}_2; \mathit{pop}_1)$: this permits to have a marker to later detect when arriving on the deepest 1-stack;
- $\delta(q_1, \_, ( ) = (q_1, \mathit{push}_2; \mathit{push}_1^{a,2})$: on reading an opening bracket we duplicate the top 1-stack and add a symbol on it (with a 2-link);
- $\delta(q_1, a,)) = (q_1, \mathit{push}_2; \mathit{pop}_1)$: on reading a closing bracket we duplicate the top 1-stack and pop its top symbol;
- $\delta(q_1, a, \star) = (q_2, \mathit{collapse})$: on reading the first $\star$ we do a *collapse* if the top element is an a;
- $\delta(q_1, \bot, \star) = (q_3, \mathit{id})$: on reading the first $\star$ we leave the stack unchanged and go to a blocking state if the top element is $\bot$;
- $\delta(q_2, a, \star) = \delta(q_2, \bot, \star) = (q_2, \mathit{pop}_2)$: on reading the next $\star$ we do a pop_2 provided the top symbol is not #;

Let us give some intuition why $\mathcal{A}_U$ generates T_U. First from the initial configuration we do the following transition

$$(q_0, \texttt{[[}\bot\texttt{]]}_2) \xrightarrow{\lambda} (q_1, \texttt{[[}\bot\#\texttt{]}\ \texttt{[}\bot\texttt{]]}_2)$$

Then after reading the k first letters $b_1 \cdots b_k$ of a prefix of a welle-bracketed word one ends up in a configuration of the form

$$(q_1, \texttt{[[}\bot\#\texttt{]}\ \texttt{[}\bot\texttt{]}\sigma_1\sigma_2 \cdots \sigma_k\texttt{]}_2)$$

where the σ_i are order-1 stacks such that, for all i

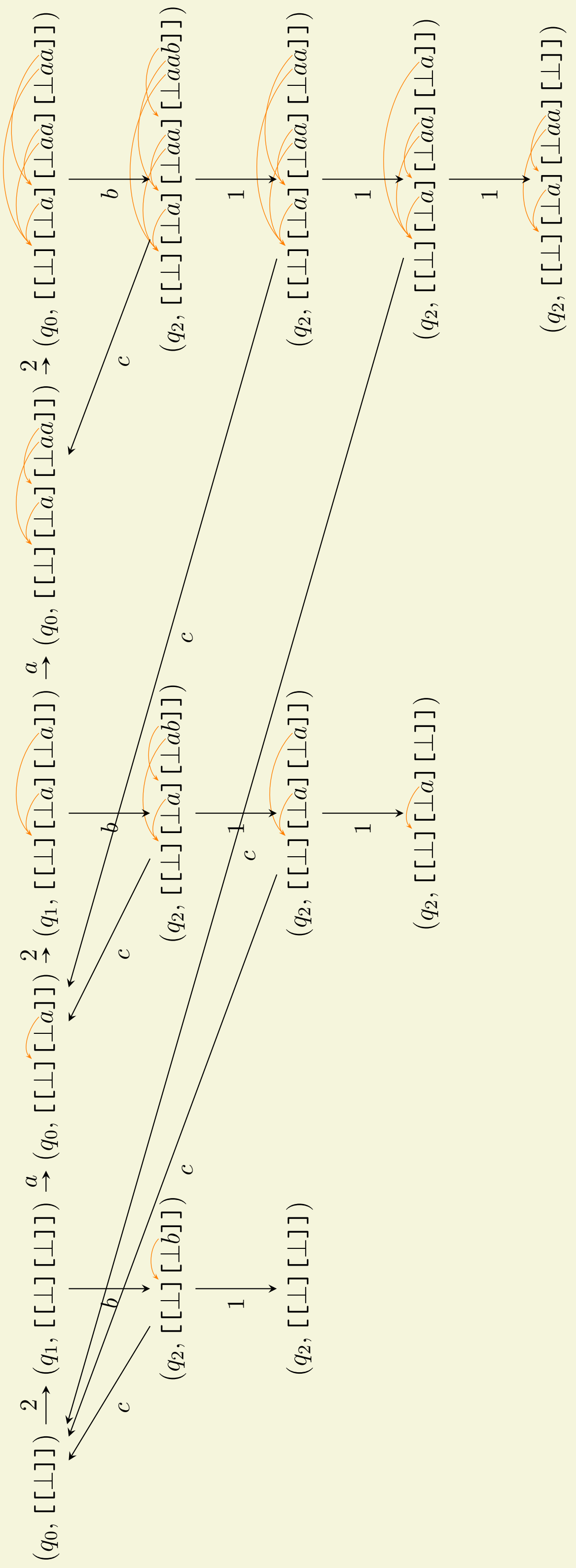


Figure 15 – The Lts $\mathcal{L}_{\mathcal{A}_t}$ associated with the CPDA $\mathcal{A}_t$ of Section 7.4

- $\sigma_i = [\perp \underbrace{a \cdots a}_{j_i}]$ where j_i is the number of unmatched brackets in $b_1 \cdots b_i$;

- if the topmost symbol of σ_i is an a (*i.e.* $b_1 \cdots b_i$ is not well-bracketed) then it has a 2-link and is such that $\mathit{collapse}(\texttt{[[}\bot\texttt{\#]}\ \texttt{[}\bot\texttt{]}\sigma_1\sigma_2\cdots\sigma_i\texttt{]}_2) = \texttt{[[}\bot\texttt{\#]}\ \texttt{[}\bot\texttt{]}\sigma_1\sigma_2\cdots\sigma_{j_i}\texttt{]}_2$ where j_i is the smallest integer such that $b_{j_i+2}\cdots b_i$ is well-bracketed (with the convention that it equals $i-1$ if no such integer exists, *i.e.* b_i is an opening bracket).

Next, if from $(q_1, \texttt{[[}\bot\texttt{\#]}\ \texttt{[}\bot\texttt{]}\sigma_1\sigma_2\cdots\sigma_k\texttt{]}_2)$ one performs a $\star$ transition, one has:

- If $b_1\cdots b_k$ is well-bracketed, $\sigma_k = \texttt{[}\bot\texttt{]}$ and thus it leads to a configuration with control state q_3 from which no more transition can be performed (which is fine as after doing a sequence of transitions coding a well-bracketed word, one should do a single $\star$-transition).

- If $b_1\cdots b_k$ is not well-bracketed, the $\star$-transition leads to configuration

$$(q_2, \texttt{[[}\bot\texttt{\#]}\ \texttt{[}\bot\texttt{]}\sigma_1\sigma_2\cdots\sigma_{j_k}\texttt{]}_2)$$

  from which j_k+1 extra successive $\star$-transitions can be performed until getting to the blocking configuration $(q_2, \texttt{[[}\bot\texttt{\#]]})$, which is fine as in this situation one should perform j_k+2 $\star$-transitions in total.

8 The Safe and the Unsafe Hierarchies

8.1 The Safety Constraint

We now consider a syntactic subfamily of (labelled) recursion schemes called the ***safe recursion schemes***. The *safety constraint* was introduced in [KNU01] but was already implicit in the work of Damm [Dam82] (see also [Mir06] for a detailed presentation). This restriction constrains the way variables are used to form argument subterms of the rules' right-hand sides. Note that this definition makes sense for both recursion schemes and labelled recursion schemes.

Definition 3 [KNU01]

A recursion scheme is ***safe*** if no right-hand side contains an argument-subterm of order k containing a variable of order strictly less than k. A recursion scheme that is not safe is called ***unsafe***.

Excepted the scheme $\mathcal{S}_U$ generating the tree of the Urzyczyn language (see Section 7.5), all examples we gave so far were safe. The scheme $\mathcal{S}_U$ is unsafe as the production $F\,\varphi\,x \xrightarrow{(} F(F\varphi x)y(Hy)$ contains in its right-hand side the argument subterm $F\varphi x : \mathrm{o} \to \mathrm{o}$ of order-1 which contains the variable $x : \mathrm{o}$ of order-0. Urzyczyn conjectured that (a slight variation) of the tree T_U generated by $\mathcal{S}_U$, though generated by an order-2 scheme, could not be generated by any *safe* scheme. This conjecture was recently proved by Parys (see [Par11; Par12] and Theorem 12 below).

***Remark* 4.** *In [KNU01; KNU02], the notion of safety is only defined for homogeneous schemes. A type is said to be* **homogeneous** *if it is either ground or equal to* $\tau_1 \to \cdots \to \tau_n \to \mathrm{o}$ *where the* τ_i*'s are homogeneous and* $\mathrm{ord}(\tau_1) \geqslant \cdots \geqslant \mathrm{ord}(\tau_n)$*. By extension, a scheme is homogeneous if all its non-terminal symbols have homogeneous types. For instance* $(\mathrm{o} \to \mathrm{o}) \to \mathrm{o} \to \mathrm{o}$ *is an homogeneous type whereas* $\mathrm{o} \to (\mathrm{o} \to \mathrm{o}) \to \mathrm{o}$ *is not. We will see in Proposition 1 that dropping the homogeneity constraint in the definition of safety does not change the family of generated trees.*

8.2 Hierarchies of Trees, Hierarchies of Labelled Transition Systems

As explained previously one can associate with any (labelled) recursion scheme a unique tree. Hence, schemes define a family of trees indexed by the order of the schemes. For any $n \geqslant 0$, we let ***n-UnsafeTrees*** denotes the set of trees generated by order-n (possibly unsafe) recursion schemes and ***n-SafeTrees*** denotes the set of trees generated by order-n *safe* recursion schemes.

We also let

$$\text{UnsafeTrees} = \bigcup_{n \geqslant 0} n\text{-UnsafeTrees}$$

and

$$\text{SafeTrees} = \bigcup_{n \geqslant 0} n\text{-SafeTrees}$$

denote the set of all trees generated by recursion schemes (*resp.* safe recursion schemes) of arbitrary order.

As explained previously one can associate with any *labelled* recursion scheme an Lts. Hence, labelled schemes define a family of Lts indexed by the order of the schemes. For any $n \geqslant 0$, we let ***n-UnsafeLTS*** denote the set of Lts generated by order-n labelled recursion schemes and ***n-SafeLTS*** denote the set of trees generated by order-n *safe* labelled recursion schemes.

We also let

$$\text{UnsafeLTS} = \bigcup_{n \geqslant 0} n\text{-UnsafeLTS}$$

and

$$\text{SafeLTS} = \bigcup_{n \geqslant 0} n\text{-SafeLTS}$$

denote the set of all trees generated by labelled recursion schemes (*resp.* safe recursion schemes) of any order.

Instead of using schemes to generate trees and Lts, one can use CPDA and HOPDA. For any $n \geqslant 0$, we let ***n-CPDATrees*** denote the set of trees generated by order-n CPDA and ***n-HOPDATrees*** denote the set of trees generated by order-n HOPDA.

We also let

$$\text{CPDATrees} = \bigcup_{n \geqslant 0} n\text{-CPDATrees}$$

and

$$\text{HOPDATrees} = \bigcup_{n \geqslant 0} n\text{-HOPDATrees}$$

denote the set of all trees generated by CPDA (*resp.* HOPDA) of any order.

For any $n \geqslant 0$, we let ***n-CPDALTS*** denote the set of Lts generated by order-n CPDA and ***n-HOPDALTS*** denote the set of trees generated by order-n HOPDA.

We also let

$$\text{CPDALTS} = \bigcup_{n \geqslant 0} n\text{-CPDALTS}$$

and

$$\text{HOPDALTS} = \bigcup_{n \geqslant 0} n\text{-HOPDALTS}$$

denote the set of all Lts generated by CPDA (*resp.* HOPDA) of any order.

8.3 The Safe Hierarchies & the Caucal Hierarchies

In [KNU01; KNU02], the motivation for considering the safety constraint was that, for generating trees, safe schemes can be translated into HOPDA and vice-versa. Namely, one has the following equi-expressivity theorem.

Theorem 4 [KNU02]

For any $n \geqslant 0$, one has

$$n\text{-SafeTrees} = n\text{-HOPDATrees}$$

Moreover the translations from schemes to HOPDA and vice-versa can be performed in polynomial time.

***Remark* 5.** *In the previous statement one implicitly requires that the safe schemes are homogeneous. This assumption can be dropped (see Corollary 2 below).*

The main motivation in [KNU01; KNU02] was to prove the decidability of MSO logic on SafeTrees. An alternative approach for the same quest was developed by Caucal in [Cau02] where he defined two hierarchies: one of trees and one of graphs/Lts . Roughly speaking, the hierarchies are defined by induction, by letting 0-CaucalLTS be the set of all finite deterministic Lts; then for any $n \geqslant 0$, one defines n-CaucalTrees as the set of trees $\mathrm{Tree}(\mathcal{L})$ where $\mathcal{L} \in n$-CaucalLTS, *i.e.* those trees generated by Lts in n-CaucalLTS, and $(n+1)$-CaucalLTS as those Lts obtained by applying an inverse rational mapping [8] to a tree in n-CaucalTrees.

The hierarchy of trees defined by Caucal coincides with the one defined by safe schemes/HOPDA.

Theorem 5 [Cau02]

For any $n \geqslant 0$, one has

$$n\text{-SafeTrees} = n\text{-CaucalTrees}$$

The hierarchy of Lts defined by Caucal coincides with the one defined by safe labelled schemes/HOPDA up to λ-contraction.

Theorem 6 [CW03]

For any $n \geqslant 0$, one has

$$\lambda - \mathrm{Contraction}(n\text{-HOPDALTS}) = n\text{-CaucalLTS}$$

As a consequence of [Dam79; Eng91] where strictness of the hierarchy of languages defined as traces of HOPDA is proved, it follows that the hierarchies of safe trees and safe Lts are strict as well

[8] A mapping $h : \Sigma \to (\Sigma \cup \overline{\Sigma})^*$ where we let $\overline{\Sigma} = \{\overline{\sigma} \mid \sigma \in \Sigma\}$ is *rational* if, for all $\sigma \in \Sigma$, $h(\sigma)$ is a rational language over $\Sigma \cup \overline{\Sigma}$. Let $\mathcal{L} = \langle D, r, \Sigma, (\xrightarrow{\sigma})_{\sigma\in\Sigma} \rangle$ be an Lts, then we let $s \xrightarrow{\overline{\sigma}} t$ iff $t \xrightarrow{\sigma} s$ and we define the binary relation $\xrightarrow{w}$ on D in the obvious way for all word $w \in (\Sigma \cup \overline{\Sigma})^*$.

The image of $\mathcal{L}$ by the inverse mapping h^{-1} is the Lts $h^{-1}(\mathcal{L}) = \langle D, r, \Sigma, (\overset{\sigma}{\rightsquigarrow})_{\sigma\in\Sigma} \rangle$ where $s \overset{\sigma}{\rightsquigarrow} t$ if and only if $\exists w \in h(\sigma)$ such that $s \xrightarrow{w} t$.

as the Caucal hierarchies. Also note that an alternative (and very elegant) proof of this statement is given in [BC10]: here the authors make no use of HOPDA but characterise the linear order that can be defined in the Caucal tree hierarchy and from this characterisation follows the strictness of the hierarchies.

Corollary 1 [Dam79; Eng91; BC10]

The hierarchies $(n\text{-SafeTrees})_n$ and $(n\text{-SafeLTS})_n$ — hence, the hierarchies defined by Caucal — are strict.

8.4 The Equi-Expressivity Theorem

It was a long-standing open problem to find a machine-based characterisation of those trees generated by recursion schemes. The answer in [KNU01; KNU02] and [Cau02] where only partial as the authors needed to restrict their attention to safe schemes. A first step toward a general solution was obtained by Knapik, Niwiński, Urzyczyn and Walukiewicz who proved in [KNUW05] that order-2 homogeneously-typed (but not necessarily safe) recursion schemes are equi-expressive with 2-CPDA (called in their work *panic automata*).

Theorem 7 [KNUW05]

One has

$$2\text{-UnsafeTrees} = 2\text{-CPDATrees}$$

Moreover the translations from schemes to CPDA and vice-versa can be performed in polynomial time.

Together with Matthew Hague, Andrzej Murawski and Luke Ong, we generalised this result to any order.

Theorem 8 (Equi-Expressivity Theorem) [HMOS08; CS12a]

For any $n \geqslant 0$, order-n recursion schemes and order-n CPDA are equi-expressive for generating trees and Lts. For any $n \geqslant 0$, one has

$$n\text{-UnsafeTrees} = n\text{-CPDATrees}$$

and

$$n\text{-UnsafeLTS} = n\text{-CPDALTS}$$

and the translations from schemes to HOPDA and vice-versa can be performed in polynomial time.

Moreover, given a scheme $\mathcal{S}$ the number of states in the equivalent CPDA $\mathcal{A}$ is linear in the maximal arity appearing in $\mathcal{S}$, and its alphabet is of size linear in the one of $\mathcal{S}$.

Note that in [HMOS08] the result for Lts is not formally stated as labelled recursion schemes were only introduced in [CS12a], but the result is implicit in [HMOS08]. However, let us mention that the proof of the equi-expressivity theorem that is given in [CS12a] uses simpler arguments when translating a scheme to a CPDA.

We do not give the proof of Theorem 8 but only mention some facts about it. The translation from CPDA to schemes is syntactical and it is immediate to note that starting from an HOPDA one can adapt the translation to end-up with a safe scheme (even in the strong sense of Damm, see Section 8.5 below). The converse translation (from schemes to CPDA) is more involved but an important point to stress here is that the resulting CPDA uses as a stack alphabet the set of all argument subterms appearing in right-hand side of production rules of the scheme and has a number of states that is linear in the maximal arity appearing in the scheme. This will be crucial when discussing the complexity of model-checking for trees generated by recursion schemes. We also invite the reader to revisit Section 1.2 where we gave at a high level the structure of a CPDA evaluating our toy example.

Note that the translation from recursion schemes to CPDA used in Theorem 8 when applied to a safe scheme leads to a link-free CPDA, *i.e.* a CPDA where any *collapse* (involving say a k-link) can be equivalently replaced by a pop_k.

Theorem 9 [CS12a]

The translation of Theorem 8 applied to a safe recursion scheme yields a link-free collapsible automaton.

Hence, we get the following corollary extending (by dropping the homogeneity assumption as well as focusing on LTS) a previous result from [KNU02] (see Theorem 4 above).

Corollary 2

For any $n \geqslant 0$, one has

$$n\text{-SafeTrees} = n\text{-HOPDATrees}$$

and

$$n\text{-SafeLTS} = n\text{-HOPDALTS}$$

Moreover the translations from schemes to HOPDA and vice-versa can be performed in polynomial time.

Recently Kartzow and Parys proved the strictness of the unsafe (equivalently, CPDA) hierarchies [KP12]. The proof is based on a pumping lemma for CPDA.

Theorem 10 [KP12]

The hierarchies $(n\text{-UnsafeTrees})_n$ and $(n\text{-UnsafeLTS})_n$, equivalently the hierarchies $(n\text{-CPDATrees})_n$ and $(n\text{-CPDALTS})_n$, are strict.

8.5 Back to Safety: Damm's View of Safety

The safety constraint may seem unnatural and purely *ad-hoc*. Inspired by the constraint of derived types of Damm, we introduce a more natural constraint, *Damm-safety*, which leads to the same family of trees [Dam82].

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

the terms φ, $\varphi\ f\ f$ and $\varphi\ f\ f\ c\ c$ can appear as argument subterms in a Damm-safe scheme but $\varphi\ f$ and $\varphi\ f\ f\ c$ are forbidden.

Definition 4 [Dam82]

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

***Remark* 6.** *The second constraint in the definition of Damm-safety can be reformulated as: all argument subterms of an argument subterm of order-k appearing in a right-hand side have at least order-k.*

Using Remark 6, it is easy to see that Damm-safety implies the safety constraint. However, the safety constraint, even when restricted to homogeneous schemes, is less restrictive than Damm-safety. Consider for instance a variable $x : o$ and non-terminals $G : o \to o \to o$ and $C : o$, then $G\, x$ cannot appear as an argument-subterm in a safe scheme but $G\, C$ can. As $G\, C$ does not satisfy Damm-safety constraint, safety is syntactically more permissive than Damm-safety. However unsurprisingly, any safe scheme can be transformed into an equivalent Damm-safe scheme of the same order. The transformation consists in converting the safe scheme into a higher-order pushdown automaton (Corollary 2) and then converting this automaton back to a scheme using the translation of [KNU02]. In fact, this translation of higher-order pushdown automata into safe schemes produces Damm-safe schemes.

Proposition 1

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

In particular, this proposition shows that any safe scheme can be transformed into an equivalent homogeneous one.

8.6 The Safe *vs* Unsafe Conjecture

A very natural question is whether safety is a genuine constraint. Originally safety was introduced for schemes seen as generator for trees but the question also makes sense for Lts (either defined by safe/unsafe labelled schemes or HOPDA/CPDA).

8.6.1 The Safe *vs* Unsafe Conjecture For Labelled Transition Systems

In [HMOS08] we proved that CPDA are more expressive than HOPDA for generating Lts and for this we consider the case of the CPDA $\mathcal{A}_t$ presented in Section 7.4.

Theorem 11 (Safe *vs* unsafe for Lts) [HMOS08]

There is no HOPDA of any order that generates the same Lts as $\mathcal{A}_t$.

Proof. The result will follows from Theorem 15 that states that $\mathcal{A}_t$ has an undecidable MSO theory. Indeed, as any Lts generated by an HOPDA has a decidable MSO theory, we directly conclude. ■

8.6.2 The Safe *vs* Unsafe Conjecture For Trees

Paweł Urzyczyn conjectured that (a slight variation) of the tree T_U described in Section 5.5, though generated by an order-2 (unsafe) scheme, could not be generated by any order-2 safe scheme. This conjecture was proved by Paweł Parys in [Par11] and later strengthened as he proved in [Par12] that even if one allows an arbitrary order no safe scheme can generate this tree. The proof is based on a pumping argument for HOPDA (and becomes technically very involved when dealing with arbitrary order).

Theorem 12 (Safe *vs* unsafe for trees) [Par11; Par12]

There is no HOPDA of any order that generates the tree T_U.

8.7 Summary

Figure 16 summarises the previously mentioned results concerning the safe and the unsafe hierarchies. In this figure we mean by n-Safe (*resp.* n-Unsafe) either n-SafeTrees $=$ n-HOPDATrees (*resp.* n-UnsafeTrees $=$ n-CPDATrees) when dealing of trees or n-SafeLTS $=$ n-HOPDALTS (*resp.* n-UnsafeLTS $=$ n-CPDALTS) when dealing of Lts.

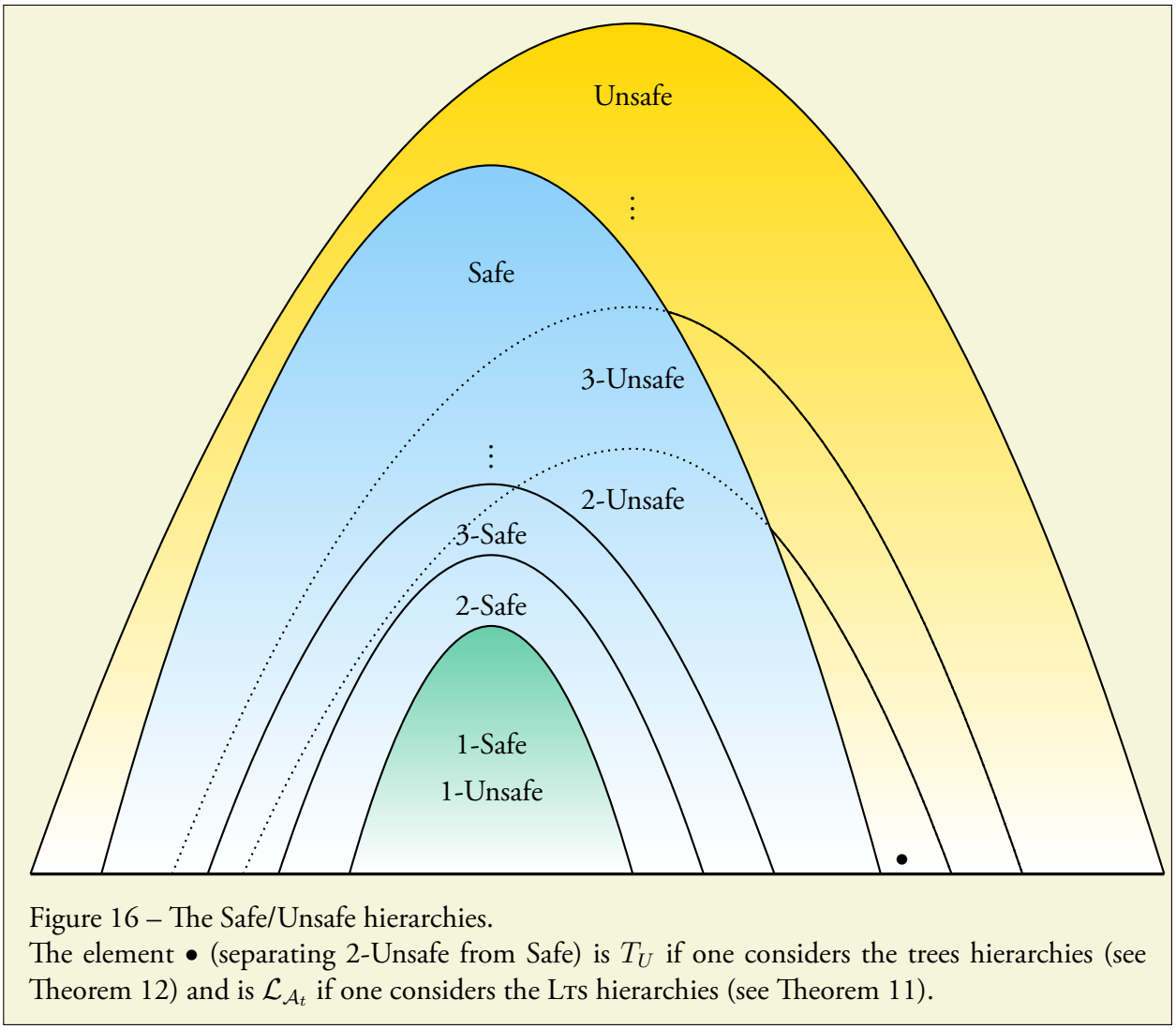


Figure 16 – The Safe/Unsafe hierarchies.
The element • (separating 2-Unsafe from Safe) is T_U if one considers the trees hierarchies (see Theorem 12) and is $\mathcal{L}_{\mathcal{A}_t}$ if one considers the Lts hierarchies (see Theorem 11).

9 A Glimpse Into Logic

9.1 Model-Checking

9.1.1 The Case of Trees

The quest of finding an alternative description of those trees generated by recursion schemes was led in parallel with the study of the decidability of the model-checking problem for the monadic second order logic (MSO) and the modal μ-calculus. Decidability of the MSO theories of trees generated by *safe* schemes was established by Knapik, Niwiński and Urzyczyn [KNU02] and independently by Caucal [Cau02] (who also proved a similar decidability result that holds on Lts as well; see Theorem 14).

The decidability for order-2 (possibly unsafe) schemes was obtained in [KNUW05] thanks to the equi-expressivity with so-called panic automata (see also [AMO05] for an equivalent result).

In 2006, Ong showed decidability of MSO for arbitrary recursion schemes [Ong06], and established that this problem is n-EXPTIME complete. This result was obtained using tools from innocent game semantics (in the sense of Hyland and Ong [HO00]) and does not rely on an equivalent automata model for generating trees. Note that, as the focus is on model-checking trees (at the root), considering μ-calculus rather than MSO is equivalent.

Since then, alternative proofs of Ong's MSO decidability results have been given: using parity games played on Lts generated by CPDA [HMOS08], using a typing approach [KO09] and using parity games on Krivine machine for the λY-calculus [SW11].

Theorem 13 [Ong06; HMOS08; KO09; SW11]

The trees in UnsafeTrees have decidable MSO theories.

9.1.2 The Case of Graphs/Lts

When working with Lts (equivalently with graphs) the landscape gets more complicated. For the safe hierarchy, the MSO logic is decidable — hence, the same holds for μ-calculus and fragments of MSO such as FO or FO(TC).

Theorem 14 [Cau02]

The graphs in the Lts hierarchy of Caucal (hence in SafeLTS) have decidable MSO theories.

When shifting to the unsafe case (*i.e.* considering graphs in UnsafeLTS = CPDALTS), the situation drastically changes.

Theorem 15 [HMOS08]

The Lts $\mathcal{L}_{\mathcal{A}_t}$ associated with the CPDA $\mathcal{A}_t$ (see Section 7.4) has an undecidable MSO theory (actually an undecidable FO(TC) theory).

Proof. Consider the following MSO interpretation[9] [Cou94] $\mathcal{I}$ of the configuration graph of the 2-CPDS $\mathcal{A}_t$ considered in Section 7.4.

$$\begin{aligned} \varphi_A(x,y) &= x \xrightarrow{C} y \ \wedge \ x \xrightarrow{R} y \\ \varphi_B(x,y) &= x \xrightarrow{1} y \end{aligned}$$

with $C = \overline{1}^* \, \overline{b} \, a \, 2 \, b \, 1^*$ and $R = c \, 2 \, a \, \overline{c} \ \vee \ \overline{1} \, c \, 2 \, a \, \overline{c} \, 1$ (here C is used to enforce that A-edges occur only between vertices from consecutive columns in the original structure while R is used to enforce that A-edges occurs only between vertices from consecutive rows in the original structure).

We observe that the image of $\mathcal{L}_{\mathcal{A}_t}$ by the interpretation $\mathcal{I}$ is the "infinite half-grid" (see Figure 17). As the infinite (half) grid has an undecidable MSO theory and as MSO interpretations

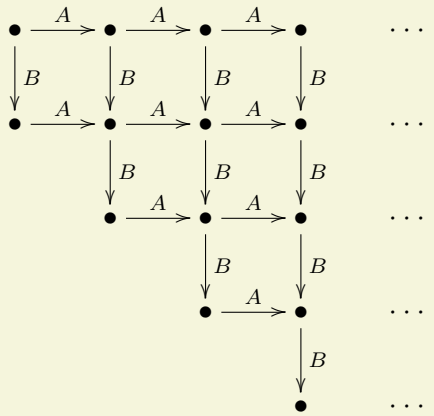


Figure 17 – The "infinite half-grid"

preserve MSO decidability we conclude that the MSO theory of $\mathcal{L}_{\mathcal{A}_t}$ is undecidable.

To refine the result to FO(TC) we simply remark that the interpretation I is FO(TC) definable and that the infinite (half) grid has an undecidable FO(TC) theory [WT07].

■

One can wonder about fragments of MSO weaker than FO(TC), *e.g.* FO(Reach) or FO. On a positive side, Kartzow proved the following result.

Theorem 16 [Kar10]

The graphs in the 2-CPDALTS = 2-UnsafeLTS have decidable FO(Reach) theories.

But moving to order-3 leads to undecidability, even if one restricts to FO.

[9] In this proof think of an interpretation as a collection of formulas of the form $\varphi_A(x,y)$. Applying such an interpretation to an LTS leads to a new LTS with the same domain but different transitions: there is an A-transition from x to y in the new LTS if and only if $\varphi_A(x,y)$ holds in the original LTS.

Theorem 17 [Bro12]

There is a graph in 3-CPDALTS = 3-UnsafeLTS with an undecidable FO theory.

Recall that *when considering* Lts *instead of trees*, the expressive power of FO (a weaker logic than the full MSO logic) differs from the one of μ-calculus (for instance one cannot express in μ-calculus the existence of a loop). Recall that, of course, μ-calculus also permits to express some properties that FO – and even FO(TC) – cannot, and that MSO is more expressive than μ-calculus.

It turns out that μ-calculus is decidable for any element in CPDALTS = UnsafeLTS. This result was proved in [HMOS08] using parity games played on Lts generated by CPDA. See Section 10 for more details on the proof techniques.

Theorem 18 [HMOS08]

The graphs in the CPDALTS = UnsafeLTS have decidable μ-calculus theories.

This result has been extended to deal with so-called *reflection* (equivalently *global* model-checking) and *selection* (equivalently *synthesis*). We discuss these properties in the next two sections.

Figure 18 summarises the decidability status of the model-checking problems for the safe/unsafe hierarchies of Lts.

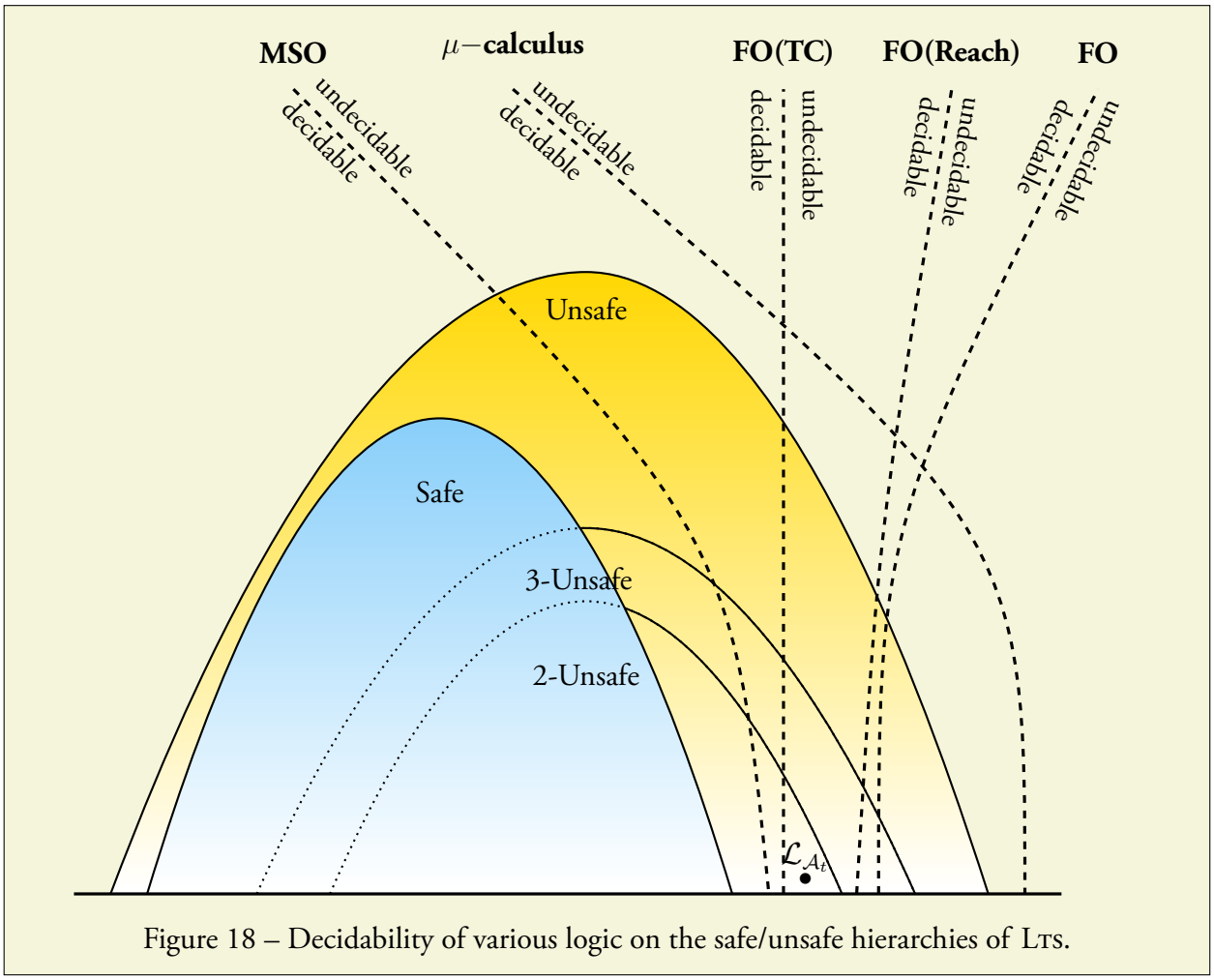


Figure 18 – Decidability of various logic on the safe/unsafe hierarchies of Lts.

9.2 Effective Reflection / Global Model-Checking

We now focus on a *global* version of model-checking where one asks for a finite description of all those elements where a given formula is true.

9.2.1 Definitions

Fix a Σ-node-labelled ranked tree t given by a recursion scheme or by a CPDA, and a logical formula φ (*e.g.* a μ-calculus formula, or an MSO formula with a single free first-order variable). We denote by $\|t\|_\varphi$ the set of nodes of t described by φ.

The ***local model-checking problem*** asks whether $u \in \|t\|_\varphi$ for a given node u. Decidability of this problem was first proved by Ong in [Ong06] (cf. Theorem 13). The ***global model-checking*** asks for a description of the set of positions where a given formula φ holds. We present two approaches to this problem, the exogenous one where the set is described by an external device (here a finite state automaton), and the endogenous one which is new and where the description is internalised by a recursion scheme with "polarized" labels.

Definition 5 (Global model-checking problem)

The ***global model-checking problem*** asks for a *finite* description of the set $\|t\|_\varphi$, if there is one. As $\|t\|_\varphi$ is in general an infinite set, there are several *non-equivalent* ways to represent it finitely. However there are two natural approaches.

- *Exogeneous:* Given a Σ-node-labelled tree $t : \mathrm{Dom} \to \Sigma$ and a formula φ, output a description by means of a word acceptor device recognising $\|t\|_\varphi \subseteq \mathrm{Dom}$.
- *Endogeneous:* Given a Σ-node-labelled tree $t : \mathrm{Dom} \to \Sigma$ and a formula φ, output a finite description of the $(\Sigma \cup \underline{\Sigma})$-node-labelled tree $t_\varphi : \mathrm{Dom} \to \underline{\Sigma}$ — 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 $u \in \|t\|_\varphi$ and $t_\varphi(u) = t(u)$ otherwise.

In case the tree t is generated by an order-n recursion scheme, it is natural to consider order-n CPDA both as words acceptors (see Remark 7 below) for $\|t\|_\varphi$ (in the exogenous approach) and as tree generator for t_φ (in the endogenous approach). In the latter case, thanks to the equi-expressivity Theorem 8, order-n schemes and CPDA can be used interchangeably.

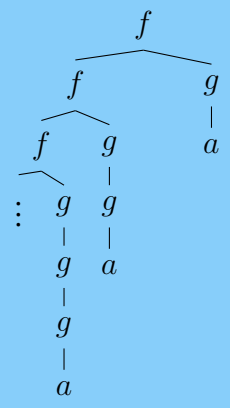
Remark* 7.** *So far, we considered CPDA as generators for* Lts *or trees. However, it is fairly simple to use them as an accepting device for word languages. Indeed, consider a CPDA $\mathcal{A} = \langle \Sigma, \Gamma, Q, \delta, q_0 \rangle$ together with a subset $F \subseteq Q$ of* final *states. Then the* **language accepted by $\mathcal{A}$**, *denoted* ***$L(\mathcal{A})$ *with final states F consists of the set of finite words $w \in \Sigma^*$ such that $(q_0, \bot_n) \overset{w}{\Longrightarrow} (f, \sigma)$ for some $f \in F$,* i.e. *the unique path in $\mathcal{L}_\mathcal{A}$ labelled by w and starting from the initial configuration ends in a configuration with a control state in F.*

Example 15

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

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

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



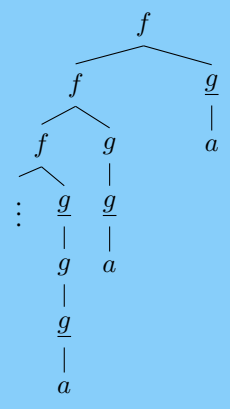
Let $\varphi = p_g \wedge \mu X.(\diamond_1 p_a \vee \diamond_1 \diamond_1 X)$, where p_g (resp. p_a) is a propositional variable asserting that the current node is labelled by g (resp. a), be the μ-calculus formula defining the nodes which are labelled by g such that the length of the (unique) path to an a-labelled node is odd.

An *exogenous* approach to the global model-checking problem is to output a 2-CPDA accepting the set $\|t\|_\varphi = \{1^n 21^k \mid n + k \text{ is odd}\}$, which in this special case is regular.

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

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

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



We now define a general concept, called *reflection*[10], and which expresses the ability to perform

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

the endogenous approach within a class of trees for a given logic.

Definition 6 (Reflection)

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

In case the class $\mathcal{C}$ is finitely presented (*i.e.* each element of $\mathcal{C}$ comes with a finite presentation, *e.g.* a scheme), we say that $\mathcal{C}$ is ***$\mathcal{L}$-effectively-reflective*** if $\mathcal{C}$ is $\mathcal{L}$-reflective and moreover one can effectively construct, for any (presentation of) $t \in \mathcal{C}$ 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 $\mathcal{C}$ into a presentation of its φ-reflection.

***Remark* 8.** *We will not discuss here the case of global model-checking for* Lts. *Indeed, we postpone it to Section 11 where the main ingredients will be introduced.*

9.2.2 μ-Calculus Effective Reflection

The following result is a generalisation of Theorem 13 that stated that μ-calculus model-checking is decidable for trees in UnsafeTrees. We obtained this result in [BCOS10] thanks to a characterisation of the winning region of parity games played on Lts generated by CPDA.

Theorem 19 (μ-Calculus Effective Reflection) [BCOS10]

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 transforms $(\mathcal{S}, \varphi)$ to an order-n CPDA $\mathcal{A}$ such that $L(\mathcal{A}) = \|t\|_\varphi$.

(2) There is an algorithm that transforms $(\mathcal{S}, \varphi)$ to an order-n recursion scheme that generates t_φ.

Before this result, we considered in [CMHOS08] the special case of *safe* schemes, leading to the following result (the second point was not in [CMHOS08] but easily follows).

Theorem 20 (μ-Calculus effective reflection: the safe case) [CMHOS08]

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

(1) There is an algorithm that transforms $(\mathcal{S}, \varphi)$ to an order-n HOPDA $\mathcal{A}$ such that $L(\mathcal{A}) = \|t\|_\varphi$.

(2) There is an algorithm that transforms $(\mathcal{S}, \varphi)$ to an order-n *safe* recursion scheme that generates t_φ.

***Remark* 9.** *Note that, both in Theorem 19 and Theorem 20,* (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 8, we can*

construct in polynomial time an order-n CPDA $\mathcal{A}$ which, together with a mapping $\kappa : Q \mapsto \Sigma \cup \underline{\Sigma}$, generates t_φ. Taking $\{q \in Q \mid \kappa(q) \in \underline{\Sigma}\}$ as a set of final states, $\mathcal{A}$ accepts $\|t\|_\varphi$.

9.2.3 MSO Effective Reflection

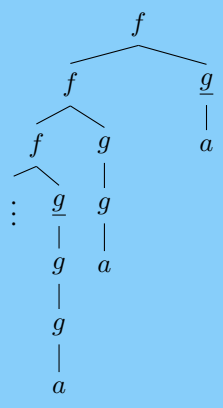
It is natural to ask if trees generated by schemes are reflective w.r.t. MSO. Indeed, modal μ-calculus and MSO are equivalent for expressing properties of a deterministic tree at the *root*, but not at other nodes (see *e.g.* [JW96]).

Example 16

Consider the following property φ (definable in MSO but not in μ-calculus) on nodes u of a tree: "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". Returning to the scheme of Example 15 one would expect the following answer to the global model-checking problem for the corresponding MSO formula:

- Exogeneous approach: $\|t\|_\varphi = \{1^n 2 \mid n \text{ is even}\}$
- Endogeneous approach:

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



The next theorem establishes the MSO reflection of UnsafeTrees and the proof is by a non-trivial reduction to the μ-calculus reflection of UnsafeTrees.

Corollary 3 (MSO effective reflection) [BCOS10]

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 transforms $(\mathcal{S}, \varphi)$ to an order-n CPDA $\mathcal{A}$ such that $L(\mathcal{A}) = \|t\|_\varphi$.

(2) There is an algorithm that transforms $(\mathcal{S}, \varphi)$ to an order-n recursion scheme that generates t_φ.

9.2.4 First Application of MSO Effective Reflection: Avoiding Divergent Computations

Consider again the scheme from Example 6 whose production rules were

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

The second rule $H\,z \to H\,(H\,z)$ is *divergent* (or *non-productive*), meaning that it generates a node labelled $\bot$ in the value tree. Consider now a scheme with production rules

$$\begin{aligned} Z &\to f\,(\boxtimes)\,(F\,f) \\ F\,\varphi &\to \varphi\,a\,(F\,\varphi) \end{aligned}$$

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

Actually, MSO effective reflection leads to the following general result, showing that one can always design an "equivalent" non-divergent scheme. See also [Had12; SW13b; Had13] for alternative proofs of this statement.

Corollary 4

Let $\mathcal{S}$ be an order-n recursion scheme with terminals A. Then one can construct another order-n scheme $\mathcal{S}_\bot$ of the same order with terminals $A \cup \{\boxtimes\}$ where $\boxtimes : o$ is a fresh symbol of arity 0 and such that

(i) $[\![\,\mathcal{S}_\bot\,]\!]$ does not contain any node labelled $\bot$;

(ii) $[\![\,\mathcal{S}\,]\!] = \tau([\![\,\mathcal{S}_\bot\,]\!])$ where $\tau : A \cup \{\boxtimes, \bot\} \to A \cup \{\bot\}$ is defined by letting $\tau(a) = a$ if $a \neq \boxtimes$ and $\tau(\boxtimes) = \bot$.

Proof. Let $\mathcal{S} = \langle\, A, N, \mathcal{R}, Z, \bot \,\rangle$ and let $@ : o \to o$ be a fresh terminal symbol of arity 1. Define $\mathcal{S}_@ = \langle\, A \cup \{@\}, N, \mathcal{R}_@, Z, \bot \,\rangle$ where $\mathcal{R}_@$ is the set of production rules $\{Fx_1 \cdots x_n \to @\,e \mid Fx_1 \cdots x_n \to e \text{ belongs to } \mathcal{R}\}$, *i.e.* we append a symbol $@$ whenever performing a rewriting step.

It is fairly simple to notice the following:

- $[\![\,\mathcal{S}_@\,]\!]$ does not contain any node labelled $\bot$;
- $[\![\,\mathcal{S}\,]\!]$ is obtained from $[\![\,\mathcal{S}_@\,]\!]$ by
  - (i) replacing any infinite branch (possibly starting from another node than the root) made only of nodes labelled by $@$ by a single node labelled by $\bot$;
  - (ii) contracting any finite path made of nodes labelled by $@$.

Now consider a formula φ stating that a node is labelled by $@$, is the source of an infinite branch made of nodes labelled by $@$ and is the son of a node not labelled by $@$ (*i.e.* the node is the first one in an infinite branch labelled by $@$). Thanks to Theorem 19 (actually Corollary 3), we obtain a new recursion scheme $\mathcal{S}_{@,\varphi}$ that generates $[\![\,\mathcal{S}_@\,]\!]_\varphi$.

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

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

(ii) in any production rule, replace any occurrence of a ground subterm of the form $@\,t$ by the ground term t.

Hence, the tree produced by $\mathcal{S}_\perp$ is obtained from $[\![\,\mathcal{S}_@\,]\!]$ by (i) replacing any infinite branch made of nodes labelled by @ by a single node labelled by $\boxtimes$ and (ii) by contracting any finite path made of nodes labelled by @. Therefore $\mathcal{S}_\perp$ is as expected.

■

9.2.5 Second Application of MSO Effective Reflection: an à la Caucal Result for Unsafe Schemes

A natural extension of the MSO reflection result (Corollary 3) is to use MSO to define new edges in the structure and not simply to mark certain nodes. This corresponds to the well-know mechanism of MSO-interpretations [Cou94]. Furthermore to obtain trees, we unfold the obtained graph from one of its nodes. As MSO-interpretations and unfolding are graph transformations that preserve the decidability of MSO, we obtain a tree with a decidable MSO-theory. Combining these two transformations provides a very powerful mechanism for constructing infinite graphs with a decidable MSO-theory. Recall that if we only use MSO-interpretations followed by unfolding to produce trees starting from the class of finite trees, we obtain the Caucal tree hierarchy (see Section 8.3).

We present here a definition of MSO-interpretations which is tailored to our setting. An ***MSO-interpretation*** over Σ-labelled ranked trees is given by a domain formula $\varphi_\delta(x)$, a formula $\varphi_\sigma(x)$ for each $\sigma \in \Sigma$ and a formula $\varphi_d(x,y)$ for each direction $d \in \mathrm{Dir}(\Sigma)$. When applied to a Σ-labelled tree t, $\mathcal{I}$ 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 σ iff u satisfies $\varphi_\sigma(x)$ in t. Similarly there exists an edge labelled by $d \in \{1,\dots,m\}$ (where m is the maximal arity of a symbol in Σ') from a vertex u to a vertex v iff the pair (u,v) satisfies the formula $\varphi_d(x,y)$ in t.

***Remark* 10.** *In all generality the alphabet of the output structure could be different from that on the input structure.*

We say that $\mathcal{I}$ is *well-formed* if for all Σ-labelled trees t, every vertex u of $\mathcal{I}(t)$ is labelled by exactly one $\sigma \in \Sigma$ and has exactly one out-going edge for each direction in $\{1,\dots\varrho(\sigma)\}$. Here we restrict our attention to well-formed interpretations,[11] which ensures that after unfolding of the interpreted graph, we obtain a Σ-labelled ranked tree.

Example 17

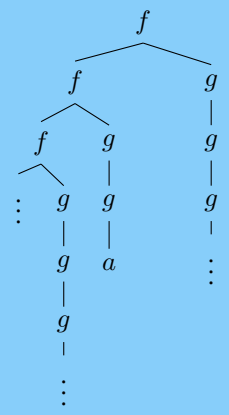
We revisit examples 15 and 16. Consider the scheme of Example 15 (recall that it generates a tree with branch language $\{f^n g^n a \mid n \geqslant 1\}$) and the formula φ of Example 16 (recall that φ was true in a node u iff u is the right son of an f-labelled node and there is a path from u to an 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

[11] Given an MSO-interpretation $\mathcal{I}$, we can decide if it is well-form. In fact, we can construct an MSO-formula $\varphi_\mathcal{I}$ which holds on the complete binary tree iff $\mathcal{I}$ is well-formed [Rab69].

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 15 and then unfolding it from its root, we obtain the tree on the right which is generated by the scheme on the left.

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



More generally, we have the following result.

Theorem 21

Let t be a Σ-labelled ranked tree generated by an order-n recursion scheme $\mathcal{S}$ and let $\mathcal{I}$ be a well-formed MSO-interpretation. The unfolding of $\mathcal{I}(t)$ from any vertex u can be generated by an order-$(n+1)$ recursion scheme.

***Remark* 11.** *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 [KNU02; Cau02]. The answer is negative. Indeed, a positive answer for 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 the result of Parys on the safe/unsafe conjecture (see Theorem 12).*

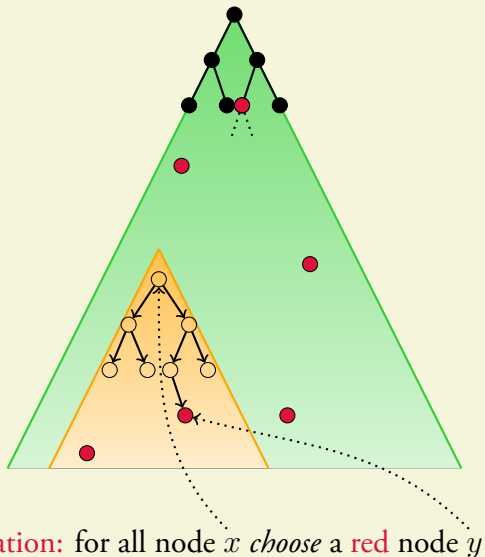
9.3 Effective Selection Property/ Synthesis

We now focus on a more general problem than global model-checking, known as the *synthesis* problem. For simplicity we start by a case study motivated by the famous question on the definability of choice functions on the infinite binary tree [GS83].

9.3.1 A Case Study: Choice Functions

We start with a case study example to illustrate the effective selection property, namely the one of 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 $\{0,1\}^*$ and we identify direction 0 with the left and direction 1 with the right). Assume that t satisfies the following extra property (we say that t is *well-formed*): every subtree contains a red node (which is equivalent to say that every subtree contains infinitely many red nodes). Our goal is for all node x to *choose/select* a red node y in the subtree $t[x]$ rooted at x. See Figure 19 for an illustration.



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

Figure 19 – Choice function

Definition 7 (Choice function, MSO choice function)

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 order-1 free variables such that

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

Gurevich and Shelah proved that such a function may not always exists. This result was later re-proven in a more elementary way by Carayol and Löding who moreover exhibited a tree in SafeTrees for which finding an MSO choice function fails.

Theorem 22 [GS83; CL07]

There is a well-formed tree $t \in$ SafeTrees 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 20 for an illustration.

Definition 8 (Partition defining a choice function)

Consider a partition $X_0 \uplus X_1$ of the set of nodes of t, and think of the nodes in X_0 (*resp.* X_1) as those where one should first go down to the left (*resp.* right) in order to find a red node. A partition (X_0, X_1) describes a choice function iff the following holds. For any node $x \in t$, define the sequence $x_0, x_1, x_2, \ldots$ by letting $x_0 = x$ and $x_{i+1} = x_i d_i$ where $d_i = 0$ if $x_i \in X_0$ and $d_i = 1$ if $x_i \in X_1$: *i.e.* $x_0, x_1, x_2, \ldots$ if the sequence of nodes visited starting from x and following the directions indicated by $X_0 \uplus X_1$. Then for some k, x_k is red.

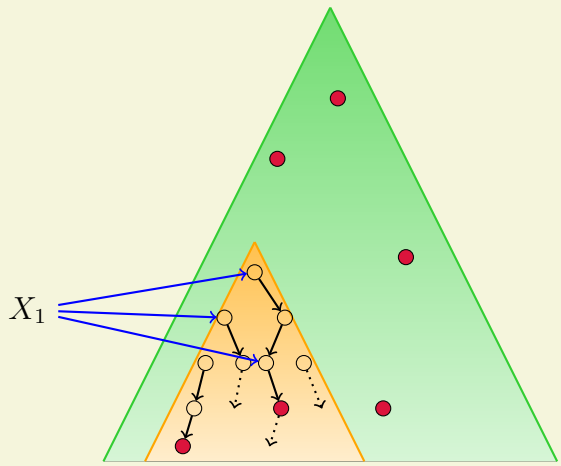


Figure 20 – Partition defining a choice function

***Remark* 12.** *Note that a partition $X_0 \uplus X_1$ always exists on a well-formed tree. Indeed, for every node u one considers 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 $u \in X_0$ otherwise one lets $u \in X_1$.*

It directly follows from Theorem 22 that a partition defining a choice function cannot be defined in MSO.

Corollary 5

There is a well-formed tree $t \in$ SafeTrees such that, for all MSO formula φ, the sets $X_0 = \{x \mid \varphi(x)$ holds$\}$ and $X_1 = \{x \mid \varphi(x)$ does not hold$\}$ do not define a choice function on t.

An *exogenous* approach to the previous problem is as follows.

Definition 9 (Exogeneous approach for choice function on well-formed trees)

Given a Σ-node-labelled well-formed tree $t : \text{Dom} \to \Sigma$, output a description by means of a word acceptor device of a subset $X \subseteq \text{Dom}$ of nodes such that $(X_0, \text{Dom}\backslash X_0)$ defines a choice function for t.

An *endogenous* approach to the previous problem is as follows.

Definition 10 (Endogeneous approach for choice function on well-formed trees)

Given a Σ-node-labelled well-formed tree $t : \text{Dom} \to \Sigma$, output a finite description of a $(\Sigma \times \{0,1\})$-node-labelled tree $t_{choice} : \text{Dom} \to \Sigma \times \{0,1\}$ such that t_{choice} and t have the same domain, and such that $X_0 = \{x \mid x \in \text{Dom}$ and $t_{choice}(x) = (\sigma, 0)$ for some $\sigma \in \Sigma\}$ and $X_1 = \{x \mid x \in \text{Dom}$ and $t_{choice}(x) = (\sigma, 1)$ for some $\sigma \in \Sigma\}$ define a choice function for t.

Contrasting with the impossibility result stated in Corollary 5 we have the following surprising result that follows from a more general result (see Theorem 23 below).

Corollary 6 [CS12a]

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

(1) There is an algorithm that builds starting from $\mathcal{S}$ an order-n CPDA $\mathcal{A}$ such that $(L(\mathcal{A}), \mathrm{Dom}(t) \backslash L(\mathcal{A}))$ defines a choice function for t.

(2) There is an algorithm that transform $\mathcal{S}$ to an order-n recursion scheme $\mathcal{S}_{choice}$ that generates a tree t_{choice} defining a choice function for t.

***Remark* 13.** *As for the reflection property (see Remark 9) we note that in the previous statement (2) implies (1).*

9.3.2 Effective Selection Property

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

Definition 11 (MSO selection problem, Selector. Exogeneous approach)

Let $\varphi(X_1, \cdots, X_\ell)$ be an MSO formula with ℓ second-order free variables, and let t 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 $U_1, \cdots, U_l \subseteq \mathrm{Dom}(t)$ such that the formula $\varphi(X_1 \leftarrow U_1, \ldots, X_\ell \leftarrow U_\ell)$ holds in t.

We now give the endogenous approach. The idea is to describes subsets of nodes $U_1, \cdots U_\ell$ that makes a given formula $\varphi(X_1, \cdots, X_\ell)$ true by marking every node with a tuple of ℓ Booleans (and a node u belongs to U_i iff the i-th Boolean is 1 in the tuple labelling u). See Figure 21 for an illustration of the following definition when $\ell = 2$.

Definition 12 (MSO selection problem, Selector. Endogeneous approach)

Let $\varphi(X_1, \cdots, X_\ell)$ be an MSO formula with ℓ second-order free variables, and let t 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 if so to output a tree t_φ labelled by the ranked alphabet $\Xi = \Sigma \times \{0,1\}^\ell$ (we take $\varrho(a, (b_1, \ldots, b_\ell)) = \varrho(a)$) such that the following holds:

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

2. The formula $\varphi(X_1 \leftarrow U_1, \ldots, X_\ell \leftarrow U_\ell)$ holds in t where $\forall 1 \leqslant i \leqslant \ell$, $U_i = \{u \in t \mid b_u(i) = 1\}$.

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

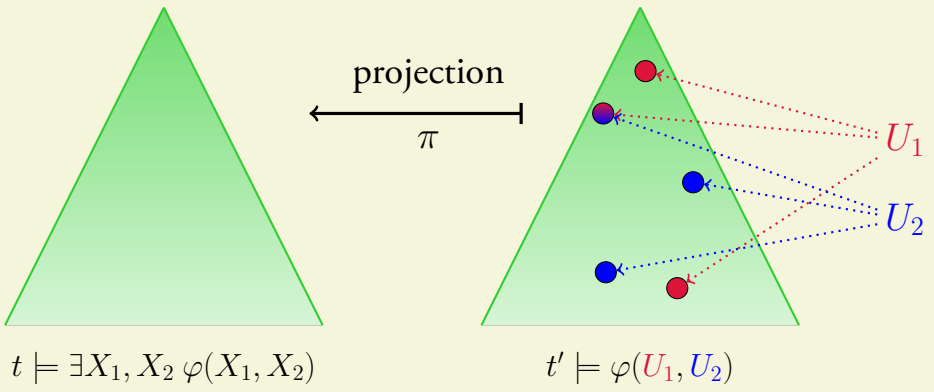


Figure 21 – The MSO selection problem

Example 18 (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 order-2 free variable).

Example 19 (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 an order-1 free variable and recall that the global model-checking asks for a description (endogenous or exogenous) of the set $\|t\|_\varphi = \{u \in \mathrm{Dom}(t) \mid \varphi(u) \text{ holds}\}$. Now let

$$\psi(X) \stackrel{def}{=} x \in X \Leftrightarrow \varphi(x)$$

Then $\exists X\ \psi(X)$ is always true and there is a unique U such that $\psi(U)$ holds, and this U is equal to $\|t\|_\varphi$. Hence an answer to the MSO selection problem for $\psi(X)$ on t leads to a solution to the global model-checking for φ on t.

Example 20 (Choice function)

We now explain how to use the MSO selection problem to obtain a partition defining an choice function on a well-formed tree t. Consider a formula $\varphi(X_0, X_1)$ that expresses the following

- X_0 and X_1 form a partition of the nodes of t.
- For all node x, there exist a red node z below x and a (finite) subset U of nodes that form 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 $y \in X_0$ (*resp.* $y \in X_1$).

Formally,

$$\begin{aligned}\varphi(X_0, X_1) \stackrel{def}{=} \quad & [\forall x,\ x \in X_0 \Leftrightarrow x \notin X_1] \mid \wedge \mid [\forall x,\ \exists z\ \exists U \\ & (x < z \wedge z \in U \wedge red(z)) \wedge (\forall y \in U,\ x \leqslant y \leqslant z) \wedge \\ & (\forall y \in U,\ y \neq z \Rightarrow \\ & \quad ((y \in X_0 \Leftrightarrow succ_0(y) \in U) \wedge (y \in X_1 \Leftrightarrow succ_1(y) \in U))) \\ &]\end{aligned}$$

One easily verifies that a solution (U_0, U_1) of the selection problem for $\varphi(X_0, X_1)$ 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 13 (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.

We obtained the following result. The proof is highly non-trivial and requires a precise characterisation of winning strategies in parity games played on Lts generated by CPDA (the key argument is that winning strategies can be embedded into the CPDA generating the term). We will discuss this later in Section 10. Also note that it contrasts with the impossibility result given by Corollary 5 (we discuss this in the next subsection).

Theorem 23 (Effective MSO selection property)[CS12a; Had13]

(Labelled) Recursion schemes as well as CPDA have the effective MSO selection property.

***Remark* 14.** *A similar statement for* safe *schemes can be deduced from [Fra06; Car06; CS08]. However the machinery for general schemes is much more involved.*

***Remark* 15.** *Theorem 23 together with Example 19 directly imply Corollary 3 (MSO reflection for schemes).*

***Remark* 16.** *The proof of Theorem 23 given by Axel Haddad in his PhD [Had13] is very different from the one we gave in [CS12a]. 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 (that in a sense has been normalised). The great advantage of Haddad's approach (built on top of the intersection types approach by Kobayashi and Ong [KO09]) is to work directly on schemes and to succeed to provide as a selector a scheme obtained from the original one by adding duplicated and annotated versions of the terminals.*

9.3.3 Selection *vs* Reflection

One may be surprised by the contrast that exists between the positive result stated in Theorem 23 and the impossibility result of Corollary 5.

In Example 19 we explained how one can reduce the MSO reflection to the MSO selection. 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 order-2 free variable. If a reduction from selection to reflexion would exist it would mean that, given any formula $\varphi(X)$ and any scheme $\mathcal{S}$ such that $\exists X\, \varphi(X)$ holds in $[\![\mathcal{S}]\!]$, then there exists another (possibly much more complicated) formula $\psi(x)$ such that the set $U = \|[\![\mathcal{S}]\!]\|_\varphi$ is such that the formula $\varphi(X \leftarrow U)$ holds in $[\![\mathcal{S}]\!]$.

Note here that (1) we do not even ask for effectivity in constructing ψ from φ and (2) 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 20; note by the way that one only needs a single free variable here as X_1 can safely be defined as the complement of X_0). Therefore if one could reduce selection to reflection, it would mean that there is an MSO formula $\psi(x)$ that defines a choice function, which contradict Corollary 5.

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

10 CPDA Parity Games: Decidability, Winning Regions & Winning Strategies

We now consider parity games played on Lts defined by CPDA. The interest for studying such game is double. First, it fits in a line of research on parity games on infinite states systems that was initiated in 1996 by Walukiewicz when considering pushdown games. Second, it is motivated by the application for the logical questions for trees generated by recursion schemes discussed in Section 9.

10.1 CPDA Parity Games

Recall that we defined general notions on parity games in Chapter 1, Section 4. We now specialise these definitions to those games played on Lts defined by CPDA.

Definition 14 (CPDA Parity Games)

Let $\mathcal{A} = \langle \Sigma, \Gamma, Q, \delta, q_0 \rangle$ be an order-n CPDA and let $\mathcal{L}_\mathcal{A} = \langle D, r, \Sigma, (\xrightarrow{a})_{a\in\Sigma} \rangle$ be the Lts associated with $\mathcal{A}$. If we forget about edge labels, we can see $\mathcal{L}_\mathcal{A}$ as a (directed) graph $G_\mathcal{A} = (V, E)$ where the set of vertices $V = D$ is the set of configurations of $\mathcal{A}$ and where $(v_1, v_2) \in E$ iff $v_1 \xrightarrow{a} v_2$ for some input letter $a \in \Sigma$.

Let $Q_E \uplus Q_A$ be a partition of Q and let Col : $Q \longrightarrow$ Colours $\subset \mathbb{N}$ be a colouring function (over states). Altogether they define a partition $V_E \uplus V_A$ of V whereby a vertex belongs to V_E iff its control state belongs to Q_E, and a colouring function Col : $V \to$ Colours where a vertex is assigned the colour of its control state. The structure $\mathcal{G} = (G_\mathcal{A}, V_E, V_A)$ defines an arena and the pair $\mathbb{G} = (\mathcal{G}, \text{Col})$ defines a parity game that we call an ***n-CPDA parity game***.

***Remark* 17.** *One may wonder in the previous definition from where the partition $Q_E \uplus Q_A$ and the colouring function* Col *are coming. In general one is interested in checking whether a given tree described by a CPDA satisfies a given MSO formula, or equivalently whether it is accepted by a parity tree automaton. The latter problem can then be expressed as solving a parity game played on a product of the CPDA and the underlying finite structure of the parity tree automaton: colours are inherited from the one used in the tree automaton while the partition $Q_E \uplus Q_A$ is coming from the dynamics of the tree automaton. For more insight on this, we suggest the reader to consult Section 2.3 in the third chapter of this document.*

Example 21

Consider the 2-CPDA from Section 7.4 (See also Figure 15). Assuming $Q_E = \{q_0, q_1\}$, $Q_A = \{q_2\}$ and $\text{Col}(q_0) = 1$, $\text{Col}(q_1) = 2$ and $\text{Col}(q_2) = 0$ one obtains the arena depicted in Figure 22

(where yellow is for colour 0, blue is for colour 1 and pink is for colour 2). Trivially Éloïse wins in this game from any position.

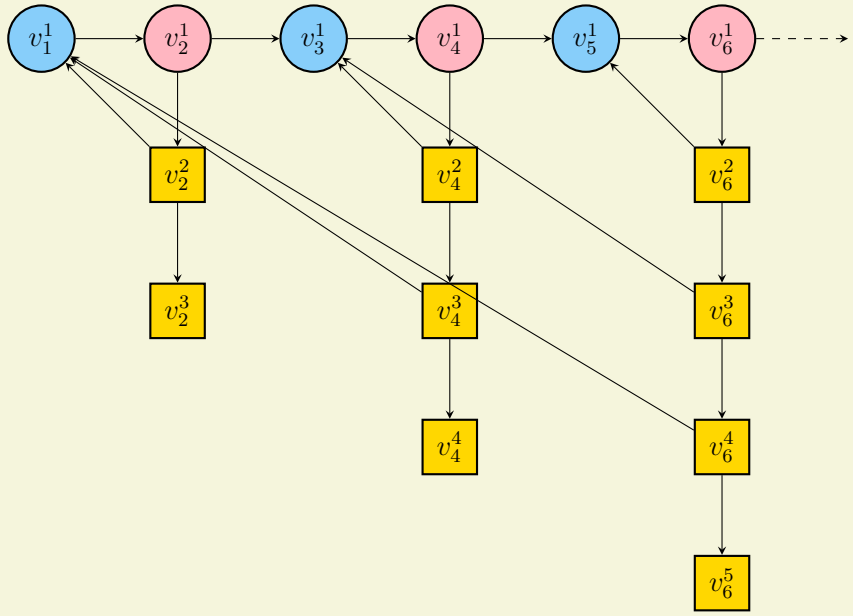


Figure 22 – The arena of the CPDA-game of Example 21

Given an n-CPDA parity game, there are three main algorithmic questions:

1. Decide whether $(q_0, \bot_n)$ is winning for Éloïse.
2. Provide a finite description of the winning region for Éloïse.
3. If $(q_0, \bot_n)$ is winning for Éloïse, provide a description of a winning strategy for Éloïse from $(q_0, \bot_n)$.

***Remark* 18.** *Note that the first question is equivalent to the following one: given a vertex $v \in V$ decide whether v is winning for Éloïse. Indeed, one can always design a new n-CPDA parity game that simulates the original one except that from the initial configuration the players are first forced to go to v, from where the simulation really starts.*

To answer the second question, we will introduce the notion of *regular sets of stacks*, and to answer the third one we will consider *strategies realised by n-CPDA transducers*.

10.2 Some History and Known Results

We briefly review the known results on CPDA parity games (and subclasses). See Table 1 for a summary.

The first paper explicitly considering pushdown games (*i.e.* order-1 CPDA games) is [Wal96]: an optimal algorithm for deciding the winner is given (ExpTime-complete) as well as a construction of a strategy realised by a synchronised pushdown automaton. However decidability can be derived from the MSO decidability of pushdown graphs [MS85] in combination with the existence of positional

winning strategies in parity games on infinite graphs [EJ91]: indeed one can write an MSO formula stating the existence of a positional winning strategy for Éloïse (see *e.g.* [Cac03a] for such a formula). We gave a very similar construction in [Ser04a]. Another approach using two-way alternating parity tree automata was developed by Vardi [Var98]. The winning region was characterised in [Ser03; Cac02] and later in [Hag08; HO11] using saturation techniques.

Cachat first considered HOPDA parity games in [Cac03b] providing an optimal algorithm for deciding the winner (n-ExpTime-complete). As for pushdown games, decidability can be derived from the MSO decidability of higher-order pushdown graphs [Cau02] in combination with the existence of positional winning strategies in parity games on infinite graphs [EJ91]. We gave a simpler proof in [CMHOS08] that permits moreover to characterise the winning region and to construct a synchronised n-HOPDA realising a winning strategy. Also see [CS08] for an approach extending the techniques of [Var98] to higher order, and [BM04; HO08] for saturation techniques (for reachability winning condition).

Order-2 CPDA parity games were considered in [KNUW05] where an optimal algorithm for deciding the winner (2-ExpTime-complete) was given. We later solved the general case in [HMOS08]. We characterised the winning region in [BCOS10] and the winning strategies in [CS12a] (even if the results are implicit in [HMOS08]). Finally, in [BCHS12], for the case of the reachability winning condition, we extended the approach of [HO08] and obtained an algorithm based on saturation to compute the winning region, and on top of this algorithm we developed the C-SHORe tool [BCHS13].

10.3 Regular Sets of Stacks with Links

We start by introducing a class of automata with a finite state-set that can be used to recognise sets of stacks with links.

Let s be an order-n stack with links. We first associate with $s = [s_1 \cdots s_\ell]_n$ a well-bracketed word of depth n, $\widetilde{s} \in (\Gamma \cup \{\bot, [,]\})^*$ by forgetting about links:

$$\widetilde{s} := \begin{cases} [\widetilde{s_1} \cdots \widetilde{s_\ell}] & \text{if } n \geqslant 1 \\ s & \text{if } n = 0 \text{ (\textit{i.e.} } s \in \Gamma) \end{cases}$$

In order to account for the link structure, we define a partial function $target(s) : \{1, \cdots, |\widetilde{s}|\} \to \{1, \cdots, |\widetilde{s}|\}$ that assigns to every position in $\{1, \cdots, |\widetilde{s}|\}$ the index of the end of the stack targeted by the corresponding link (if exists: indeed this is undefined for positions with symbol $\bot$, $[$ or $]$). Thus with s is associated the pair $\langle \widetilde{s}, target(s) \rangle$; and with a set S of stacks is associated the set $\widetilde{S} = \{\langle \widetilde{s}, target(s) \rangle \mid s \in S\}$.

Example 22

Consider the stack (recall that 1-links are not depicted)

$$s = [[[\bot\,\alpha]_1]_2\ [[\bot]_1[\bot\,\alpha\,\beta\,\gamma]_1]_2]_3$$

Then

$$\widetilde{s} = [[[\bot\,\alpha]]\ [[\bot][\bot\,\alpha\,\beta\,\gamma]]]$$

and $target(s) = \tau$ where $\tau(5) = 4$, $\tau(14) = 13$, $\tau(15) = 11$ and $\tau(16) = 7$.

	Pushdown	n-HOPDA	n-CPDA
Winning strategy	Realised by a synchronised pushdown automaton [Wal96; Ser04a]	Realised by a synchronised n-HOPDA [CMHOS08; CS08]	Realised by a synchronised n-CPDA [HMOS08; CS12a]
Winning region	Regular [Ser03; Cac02; Hag08; HO11]	Regular [CMHOS08; CS08] See also [BM04; HO08] for reachability using saturation methods	Regular [BCOS10] See also [BCHS12] for reachability using saturation methods
Solving	Decidable [MS85] + [EJ91] ExpTime-complete [Wal96; Var98; Ser04a] PSpace-complete for 1-counter [Ser06b]	Decidable [Cau02] + [EJ91] n-ExpTime-complete [Cac03b; CMHOS08]	n-ExpTime-complete [HMOS08] See also [KNUW05] for a previous study at order-2

Table 1 – Known results on CPDA parity games and subclasses

We consider *deterministic* finite automata working on such representations of stacks. The automaton reads the word $\widetilde{s}$ from left to right (that is, from bottom to top). On reading a letter that does not have a link (i.e. $target$ is undefined on its index) the automaton updates its state according to the current state and the letter; on reading a letter that has a link, the automaton updates its state according to the current state, the letter and the state it was in after processing the targeted position. A run is accepting if it ends in a final state. One can think of these automata as a deterministic version of Stirling's *dependency tree automata* [Sti09] restricted to words.

Definition 15 (Regular set of stacks with links) [BCOS10]

Formally, an ***automaton*** is a tuple $\mathcal{B} = \langle\, Q, A, q_{in}, F, \delta \,\rangle$ where Q is a finite set of states, A is a finite input alphabet (in our case $A = \Gamma \cup \{\bot, [,]\}$), $q_{in} \in Q$ is the initial state, $F \subseteq Q$ is a set of final states and $\delta : (Q \times A) \cup (Q \times A \times Q) \to Q$ is a transition function. With a pair $\langle\, u, \tau \,\rangle$ where $u = a_1 \cdots a_n \in A^*$ and τ is a partial map from $\{1, \cdots n\} \to \{1, \cdots n\}$, we associate a *unique* run $r = r_0 \cdots r_n$ as follows:

- $r_0 = q_{in}$;
- for all $0 \leqslant i < n$, $r_{i+1} = \delta(r_i, a_{i+1})$ if $i + 1 \notin Dom(\tau)$;

- for all $0 \leqslant i < n$, $r_{i+1} = \delta(r_i, a_{i+1}, r_{\tau(i+1)})$ if $i+1 \in Dom(\tau)$.

The run is *accepting* just if $r_n \in F$, and the pair (u, τ) is *accepted* just if the associated run is accepting.

To recognise configurations instead of stacks, we use the same machinery but now add the control state at the end of the coding of the stack. We code a configuration (p, s) as the pair $\langle\, \widetilde{s} \cdot p, target(s) \,\rangle$ (hence the input alphabet of the automaton also contains a copy of the control state of the corresponding CPDA).

Finally, we say that a set K of n-stacks over alphabet Γ is ***regular*** just if there is an automaton $\mathcal{B}$ such that for every n-stack s over Γ, $\mathcal{B}$ accepts $\langle\, \widetilde{s}, target(s) \,\rangle$ iff $s \in K$. Regular sets of configurations are defined in the same way.

Example 23

Fix some order $n > 1$ and let L be the set of order-n stacks (on stack alphabet $\Gamma = \{a, b\}$) such that any k-link for $k > 1$ is pointing to a $(k-1)$-stacks whose topmost element is an a. For instance, the following stack belongs to L

$$\sigma_1 = [\,[\,[\,\bot\, a\,]_1\,]_2\ [\,[\,\bot b\,]_1\,[\,\bot b\, a\,]_1\,[\,\bot\, a\, b\, b\,]_1\,]_2\,]_3$$

but the next one does not

$$\sigma_2 = [\,[\,[\,\bot\, a\,]_1\,]_2\ [\,[\,\bot b\,]_1\,[\,\bot b\, a\,]_1\,[\,\bot\, a\, b\, b\,]_1\,]_2\,]_3$$

We claim that L is regular. Indeed, the idea is to consider an automaton with state set $Q = \{r, p_a, p_b, q_a, q_b, p_\bot, q_\bot\}$ and that behaves as follows:

- r is a sink (non accepting) state.
- On reading an $x \in \{a, b\}$ with a link targeting a position where the automaton was on some p_y the automaton goes to p_x. This corresponds to the case of 1-link where no condition is required on the targeting position: we remember the letter x in the control state.
- On reading a] from state p_x or q_x (for some $x \in \{a, b, \bot\}$) the automaton goes to q_x: we remember the topmost letter of the stack in the control state as long as we read].
- On reading an $x \in \{a, b\}$ with a link targeting a position where the automaton was on some q_y the automaton goes to p_x if $y = a$ and to r if $y = b$. This corresponds to the case of a k-link with $k > 1$ where one needs to check what was the topmost letter in the targeted stack.
- On reading either $\bot$ or [the automaton goes to state $p_\bot$: we re-initialise the state to start processing the next stack.

The initial state is $p_\bot$ and the final states are $F = \{q_a, q_b, q_\bot\}$.

Formally:

- $\delta(r, \_) = \delta(r, \_, \_) = r$;
- $\delta(s, x, p\_) = p_x$ for all $x \in \{a, b\}$ and all $s \neq r$;

- $\delta(p_x,]) = \delta(q_x,]) = q_x$;
- $\delta(s, x, q_a) = p_x$ for all $x \in \{a, b\}$ and all $s \neq r$;
- $\delta(s, x, q_b) = r$ for all $x \in \{a, b\}$ and all $s \neq r$;
- $\delta(s, \bot) = \delta(s, [) = p_\bot$ for all $s \neq r$.

On σ_1 the (accepting) run is

$$p_\bot \xrightarrow{[} p_\bot \xrightarrow{[} p_\bot \xrightarrow{[} p_\bot \xrightarrow{\bot} p_\bot \xrightarrow{a} p_a \xrightarrow{]} q_a \xrightarrow{]} q_a \xrightarrow{[} p_\bot \xrightarrow{[} p_\bot \xrightarrow{\bot} p_\bot \xrightarrow{b} p_b \xrightarrow{]} q_b \xrightarrow{[} p_\bot \xrightarrow{\bot} p_\bot \xrightarrow{b} p_b \xrightarrow{a} p_a \xrightarrow{]} q_a \xrightarrow{[} p_\bot \xrightarrow{\bot} p_\bot \xrightarrow{a} p_a \xrightarrow{b} p_b \xrightarrow{b} p_b \xrightarrow{]} q_b \xrightarrow{]} q_b \xrightarrow{]} q_b$$

On σ_2 the (rejecting) run is

$$p_\bot \xrightarrow{[} p_\bot \xrightarrow{[} p_\bot \xrightarrow{[} p_\bot \xrightarrow{\bot} p_\bot \xrightarrow{a} p_a \xrightarrow{]} q_a \xrightarrow{]} q_a \xrightarrow{[} p_\bot \xrightarrow{[} p_\bot \xrightarrow{\bot} p_\bot \xrightarrow{b} p_b \xrightarrow{]} q_b \xrightarrow{[} p_\bot \xrightarrow{\bot} p_\bot \xrightarrow{b} r \xrightarrow{a} r \xrightarrow{]} r \xrightarrow{[} r \xrightarrow{\bot} r \xrightarrow{a} r \xrightarrow{b} r \xrightarrow{b} r \xrightarrow{]} r \xrightarrow{]} r \xrightarrow{]} r$$

Regular sets of stacks (resp. configurations) form an effective Boolean algebra. Indeed, closure under complement comes from the fact that we consider *deterministic* automata. Closure under union or intersection is achieved by considering a Cartesian product, as in the case of finite automata on finite words.

Proposition 2 [BCOS10]

Let H, K be regular sets of n-stacks over an alphabet Γ. Then $L \cup K$, $L \cap K$ and $Stacks(\Gamma) \backslash L$ are also regular (here $Stacks(\Gamma)$ denotes the set of all stacks over Γ). The same holds for regular sets of configurations.

***Remark* 19.** *When focusing on stacks* without *links (equivalently configurations of HOPDA) the notion of regular set of stacks/configurations coincide with the classical notion of regular languages on finite words (indeed as there is no link our automata are nothing else but standard finite state automata on finite words). In particular, regularity of a set of stacks is a special (strict) case of* rational *set of stacks in the sense of Carayol [Car05; Car06].*

10.4 CPDA Strategies

Let $\mathcal{A} = \langle \Sigma, \Gamma, Q = Q_E \uplus Q_A, \delta, q_0 \rangle$ be an order-n CPDA defining an arena $\mathcal{G} = (G_\mathcal{A}, V_E, V_A)$ and let $\mathbb{G} = (\mathcal{G}, \mathrm{Col})$ be a corresponding n-CPDA parity game.

We aim at defining a notion of n-CPDA *transducer* that provides a description for strategies in $\mathbb{G}$, that is describes a function from partial plays in $\mathbb{G}$ into V.

Consider a partial play $\Lambda = v_0 v_1 \cdots v_\ell$ in $\mathbb{G}$ where $v_0 = (q_0, \bot_n)$. An alternative description of Λ is by the sequence $(q_1, op_1) \cdots (q_\ell, op_\ell) \in (Q \times Op_n(\Gamma))^*$ such that $v_i = (q_i, \sigma_i)$ for all $1 \leqslant i \leqslant \ell$ and $\sigma_i = op_i(\sigma_{i-1})$ (with the convention that $\sigma_0 = \bot_n$). We may in the following use implicitly this representation of Λ when needed. Similarly, one can represent a strategy as a (partial) function $\varphi : (Q \times Op_n(\Gamma))^* \to Q \times Op_n(\Gamma)$, the meaning being that in a partial play Λ ending in some vertex (q, σ) if $\varphi(\Lambda) = (q', op)$ then the player moves to $(q', op(\sigma))$.

Definition 16 (n-CPDA transducer realising a strategy)

An ***n-CPDA transducer realising a strategy*** in $\mathbb{G}$ is a tuple $\mathcal{S} = \langle \Sigma, S, s_0, \delta_{\mathcal{S}}, \tau, \sigma_0 \rangle$ where Σ is a stack alphabet, S is a finite set of states, $s_0 \in S$ is the initial state,

$$\delta_{\mathcal{S}} : S \times \Sigma \times (Q \times Op_n(\Gamma)) \to S \times Op_n(\Sigma)$$

is a *deterministic* transition function and

$$\tau : S \times \Sigma \to Q \times Op_n(\Gamma)$$

is a function (note that we do not require τ to be total).

A configuration of $\mathcal{S}$ is a pair (s, σ) where s is a state and σ is an n-stack over Σ; the initial configuration of $\mathcal{S}$ is $(s_0, \bot_n)$. With a configuration (s, σ) is associated, when defined, a (unique) move in $\mathbb{G}$ given by $\tau(s, \mathit{top}_1(\sigma))$. A partial play $\Lambda = (q_1, op_1) \cdots (q_\ell, op_\ell)$ in $\mathbb{G}$ induces a (unique, when defined) ***run*** of $\mathcal{S}$ which is the sequence

$$(s_0, \sigma_0)(s_1, \sigma_1) \cdots (s_\ell, \sigma_\ell)$$

where $(s_0, \sigma_0) = (s_0, \bot_n)$ is the initial configuration of $\mathcal{S}$ and for all $0 \leqslant i \leqslant \ell - 1$ one has $\delta_{\mathcal{S}}(s_i, \mathit{top}_1(\sigma_i), (q_{i+1}, op_{i+1})) = (s_{i+1}, op'_{i+1})$ with $\sigma_{i+1} = op'_{i+1}(\sigma_i)$. In other words, the control state and the stack of $\mathcal{S}$ are updated accordingly to δ.

We say that $\mathcal{S}$ is ***synchronised*** with $\mathcal{A}$ iff for all $(s, a, (q, op)) \in S \times \Sigma \times (Q \times Op_n(\Gamma))$ such that $\delta(s, a, (q, op)) = (s', op')$ is defined one has that op and op' are of the same kind, *i.e.* either they are both a pop_k (for the same k) or both a push_k (for the same k) or both a $\mathit{push}_{\overline{1}}^{,e}$ (the symbol pushed being possibly different but the order of the link being the same) or both *collapse* or both $\mathit{rew}_{\overline{1}}$ (the new symbol after rewritting being possibly different) or both *id*. In particular, if one defines the ***shape*** of a stack σ as the stack obtained by replacing all symbols appearing in σ by a fresh symbol $\sharp$ (but keeping the links) one has the following.

Proposition 3

Assume that $\mathcal{S}$ is synchronised with $\mathcal{A}$. Then, for any partial play Λ in $\mathbb{G}$ ending in a configuration with stack σ, the run of $\mathcal{S}$ on Λ, when exists, ends in a configuration with stack σ' such that σ and σ' have the same shape.

Definition 17 (Strategy realised by an n-CPDA transducer)

The ***strategy realised by $\mathcal{S}$*** is the (partial) function $\varphi_{\mathcal{S}}$ defined by letting $\varphi_{\mathcal{S}}(\Lambda) = \tau((s, \mathit{top}_1(\sigma)))$ where (s, σ) is the last configuration of the run of $\mathcal{S}$ on Λ.

We say that Éloïse ***respects*** $\varphi_{\mathcal{S}}$ during a partial play $\Lambda = (q_1, op_1) \cdots (q_\ell, op_\ell)$ in $\mathbb{G}$ iff for all $0 \leqslant i \leqslant \ell - 1$ whenever the last configuration in $(q_1, op_1) \cdots (q_i, op_i)$ belongs to V_E one has $(q_{i+1}, op_{i+1}) = \varphi_{\mathcal{S}}((q_1, op_1) \cdots (q_i, op_i))$

We say that $\varphi_{\mathcal{S}}$ is ***well-defined*** iff for any partial play $\Lambda = (q_1, op_1) \cdots (q_\ell, op_\ell)$ where Éloïse respects $\varphi_{\mathcal{S}}$ whenever the last vertex (q_ℓ, σ_ℓ) in Λ belongs to V_E one has $\varphi_{\mathcal{S}}(\Lambda) \in \Delta(q, \mathit{top}_1(\sigma_\ell))$, *i.e.* the move given by $\varphi_{\mathcal{S}}$ is a valid one.

***Remark* 20.** *In [CS12a] when considering CPDA strategies as well as CPDA for generating parity games, we assume that instead of performing a single stack operation per transition one actually does a top-rewriting operation (or the identity) followed by another stack operation (possibly the identity). This is mainly for technical reasons in the proof (and for complexity reasons) and does not change the expressive power of CPDA for generating trees or graphs (indeed, a λ-transition can be used to simulate the application of two stack operations). In the present document, as we do not go into the proof details, we omit this technicality.*

Strategies realised by a synchronised n-CPDA are of special interest. Indeed, assume that one considers a game defined by an n-CPDA $\mathcal{A}$ and constructs an n-CPDA transducer $\mathcal{S}$ synchronised with $\mathcal{A}$ and realising a winning strategy for Éloïse from the initial configuration. It is then natural to consider a "synchronised product" of $\mathcal{A}$ with $\mathcal{S}$ which is again an n-CPDA (remember that $\mathcal{A}$ and $\mathcal{S}$ are synchronised) and that generates an Lts similar to the one generated by $\mathcal{A}$ except that only those transitions consistent with the strategy described by $\mathcal{S}$ are kept. This argument will be later developed in the (sketch of) proof of Theorem 23 in Section 11.2.

10.5 Main Result & Some Words About its Proof

The following result gives a complete landscape for CPDA parity games.

Theorem 24 [HMOS08; CMHOS08; BCOS10; CS12a]

Let $\mathcal{A} = \langle \Sigma, \Gamma, Q = Q_E \uplus Q_A, \delta, q_0 \rangle$ be an n-CPDA and let $\mathbb{G}$ be an n-CPDA parity game defined from $\mathcal{A}$. Then one has the following results.

(1) Deciding whether $(q_0, \bot_n)$ is winning for Éloïse is an n-ExpTime complete problem. More precisely, the overall complexity is n-times exponential in $|Q|$ and in the number of colours but *polynomial* in $|\Gamma|$;

(2) The winning region for Éloïse (*resp.* for Abélard) is regular. Moreover one can compute an automaton that recognises it.

(3) If $(q_0, \bot_n)$ is winning for Éloïse then one can effectively construct an n-CPDA transducer $\mathcal{S}$ *synchronised* with $\mathcal{A}$ realising a well-defined *winning* strategy $\mathcal{S}$ for Éloïse in $\mathbb{G}$ from $(q_0, \bot_n)$.

The proof of Theorem 24 requires a lot of machinery that we briefly outline. The proof goes by reducing the order, ending up in a game played on a finite graph.

We want to have a notion of "reduction" between games. Typical requirements to say that a game $\mathbb{G}$ reduces to a game $\mathbb{G}'$ should be the following (we will then specialise this to the setting of CPDA parity games):

(1) The game $\mathbb{G}'$ can be computed from $\mathbb{G}$ (possibly leading a blow-up in the size).

(2) There is a computable function ν, that associates with any vertex v in $\mathbb{G}$ a vertex $\nu(v)$ in $\mathbb{G}'$ such that v is winning in $\mathbb{G}$ iff $\nu(v)$ is winning in $\mathbb{G}'$.

(3) There is a computable function ζ, that associates with any vertex v in $\mathbb{G}$ that is winning for Éloïse and any winning strategy φ of Éloïse in $\mathbb{G}$ from v, a strategy $\zeta(\varphi)$ that is winning for Éloïse in $\mathbb{G}'$ from $\nu(v)$.

Indeed, (1) and (2) permit to compute the winning positions in $\mathbb{G}$ provided one knows how to compute them in $\mathbb{G}'$, while (1) and (3) do the same thing but for strategies.

Before formalising this concept for the special case of CPDA parity games, recall that we are interested in showing regularity of the winning region and we want to construct a CPDA transducer synchronised with the one describing the arena and that realises a winning strategy.

Definition 18 (Game reduction for CPDA parity games)

Assume $\mathcal{A}$ is an n-CPDA defining a parity game $\mathbb{G}$ and that $\mathcal{A}'$ is an n'-CPDA defining a parity game $\mathbb{G}'$ (note that $\mathcal{A}$ and $\mathcal{A}'$ may have different orders). Then one says that $\mathbb{G}$ ***reduces*** to $\mathbb{G}'$, denoted $\mathbb{G} \preccurlyeq \mathbb{G}'$ if the following holds (see Figure 23):

(1) The CPDA $\mathcal{A}'$ can be computed from $\mathcal{A}$ (possibly leading a blow-up in the size).

(2) There is a computable function ν, that associates with any configuration v of $\mathcal{A}$ a configuration $\nu(v)$ of $\mathcal{A}'$ such that:

  (i) For any configuration v of $\mathcal{A}$, v is winning in $\mathbb{G}$ iff $\nu(v)$ is winning in $\mathbb{G}'$.

  (ii) The image by ν of the initial configuration v_0 of $\mathcal{A}$ is the initial configuration of $\mathcal{A}'$.

  (iii) If the set of winning configurations for Éloïse in $\mathbb{G}'$ is regular, then the set of winning configurations for Éloïse in $\mathbb{G}$ is regular as well.

(3) If there is an n'-CPDA transducer $\mathcal{S}'$ synchronised with $\mathcal{A}'$ realising a well-defined winning strategy for Éloïse in $\mathbb{G}'$ from the initial configuration $\nu(v_0)$, then one can effectively construct an n-CPDA transducer $\mathcal{S}$ synchronised with $\mathcal{A}'$ realising a well-defined winning strategy for Éloïse in $\mathbb{G}'$ from the initial configuration v_0.

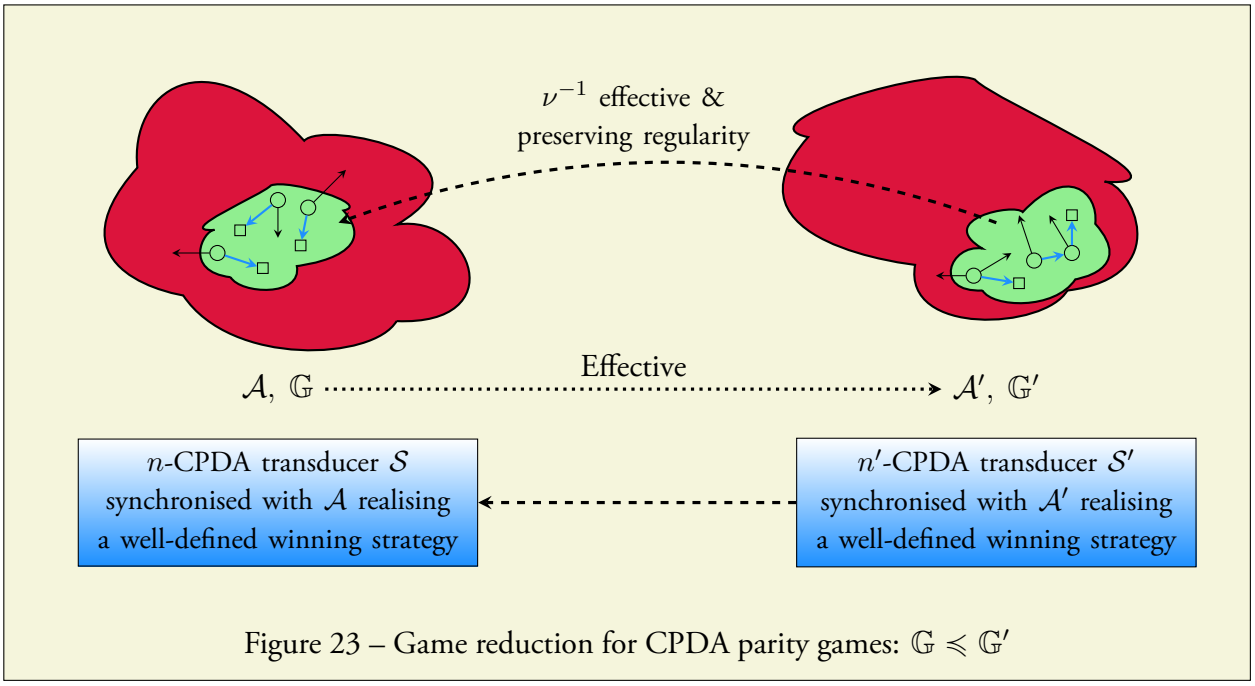


Figure 23 – Game reduction for CPDA parity games: $\mathbb{G} \preccurlyeq \mathbb{G}'$

The proof of Theorem 24 is by induction on the order of the CPDA $\mathcal{A}$. When the order is 0, the game $\mathbb{G}$ is finite and is solved using standard techniques. When the order is $n > 0$ one does a

sequence of 3 reductions:

$$\mathbb{G} \preccurlyeq \mathbb{G}_{\text{rk}} \preccurlyeq \mathbb{G}_{\text{lf}} \preccurlyeq \widetilde{\mathbb{G}}$$

where (we detail the concept below)

(1) $\mathbb{G}_{\text{rk}}$ is an n-CPDA parity game generated by an n-CPDA verifying some specific property, namely to be *rank-aware*.

(2) $\mathbb{G}_{\text{lf}}$ is an n-CPDA parity game generated by an n-CPDA that is order-n link-free, *i.e.* it never creates n-links.

(3) $\widetilde{\mathbb{G}}$ is an $(n-1)$-CPDA parity game.

The game $\widetilde{\mathbb{G}}$ is exponentially larger than the game $\mathbb{G}$, which gives the n-ExpTime upper-bound. The matching lower bound was already proved for HOPDA game in [CW07] but we gave a much simpler argument in [HMOS08].

We conclude this section by giving some more details on the proof ideas and side results.

Intuitively, an n-CPDA is ***rank-aware*** if it comes with a function $\xi : Q \to \text{Colours}$ (Q denotes the set of states) such that the following holds. For any sequence $(q_0, \sigma_0) \xrightarrow{a_1} (q_1, \sigma_1) \xrightarrow{a_2} \cdots \xrightarrow{a_i} (q_i, \sigma_i)$ if the topmost symbol of σ_i has an n-link then $\xi(q_i)$ is the smallest colour visited since the creation of the (original copy) of this link. The proof that one can built for any n-CPDA game $\mathbb{G}$ an n-CPDA game $\mathbb{G}_{\text{rk}}$ such that $\mathbb{G} \preccurlyeq \mathbb{G}_{\text{rk}}$ is a non-trivial generalisation of a similar result given in [KNUW05] at order-2. Indeed, it requires to introduce a lot of machinery that we omit in this document.

Then, the idea to build $\mathbb{G}_{\text{lf}}$ such that $\mathbb{G}_{\text{rk}} \preccurlyeq \mathbb{G}_{\text{lf}}$ is the following (we call $\mathcal{A}_{\text{rk}}$ the underlying n-CPDA generating $\mathbb{G}_{\text{rk}}$ and $\mathcal{A}_{\text{lf}}$ the underlying n-CPDA generating $\mathbb{G}_{\text{lf}}$). The n-CPDA $\mathcal{A}_{\text{lf}}$ *simulates* $\mathcal{A}_{\text{rk}}$ as follows. As long as no stack operation of the form $push_1^{\alpha,n}$ is performed $\mathcal{A}_{\text{lf}}$ behaves as $\mathcal{A}_{\text{rk}}$. Now, in order to simulate a stack operation of the form $push_1^{\alpha,n}$ the players go into the following "negotiation":

- Éloïse has to choose a vector $(R_0, \ldots, R_d)$, here we assume that $\text{Colours} = \{0, \ldots, d\}$, where each $R_i \subseteq Q_{\text{rk}}$ is a subset of states of the CPDA $\mathcal{A}_{\text{rk}}$. This choice means that she is claiming that she has a strategy such that if the n-link created by pushing α is eventually used for collapsing the stack then the control state after collapsing will belong to R_i where i is meant to be the smallest colour from the creation of the link to the collapse of the stack (equivalently it will be the rank — as computed in $\mathcal{A}_{\text{rk}}$ thanks to function ξ — just before collapsing). Note that the R_i are arbitrary sets because Éloïse does not have full control over the play (and in general cannot force R_i to be a singleton).

- Then Abélard may choose to anticipate and simulate the *collapse*: in this case he chooses a set R_i and a state $q \in R_i$ and performs a pop_n and goes to state q (through an intermediate state of colour i). If he does not want to simulate a *collapse* then one stores $(R_0, \ldots, R_d)$ into the stack since its truth may be checked later in the play.

Note that we no longer create n-links. Now if at some point a player wants to simulate a *collapse* that would have involved an n-link, the topmost symbol comes with a vector $(R_0, \ldots, R_d)$ previously claimed by Éloïse. As the simulated CPDA $\mathcal{A}_{\text{rk}}$ is rank-aware, one can check whether the original claim of Éloïse is consistent with the actual play thanks to function ξ.

The last step, building an order-$(n-1)$ CPDA game $\widetilde{\mathbb{G}}$ such that $\mathbb{G}_{\text{lf}} \preccurlyeq \widetilde{\mathbb{G}}$ is achieved by situating the techniques in a general and abstract framework of (order-1) pushdown automata whose stack

alphabet is a *possibly infinite* set (we originally introduced this concept in [CMHOS08] where we studied HOPDA parity games).

Definition 19 (Abstract pushdown automaton) [CMHOS08]

An ***abstract pushdown automaton*** is a tuple $\mathcal{P} = \langle \Sigma, A, Q, \delta, q_0 \rangle$ where Σ is a finite input alphabet, A is a (possibly *infinite*) set called an ***abstract pushdown alphabet*** and containing a bottom-of-stack symbol denoted $\bot \in A$, Q is a finite set of states, $q_0 \in Q$ is an initial state and

$$\delta : Q \times A \times \Sigma \to Q \times A^{\leqslant 2}$$

is the transition relation. We additionally require that for all $q \in Q$, all $a \neq \bot$ and all $s \in \Sigma$, $\Delta(q, a, s) \subseteq Q \times (A \backslash \{\bot\})^{\leqslant 2}$, and for all $q \in Q$ and $s \in \Sigma$, $\Delta(q, \bot, s) \subseteq Q \times (\{\bot\} \cup \{\bot a \mid a \in A\})$, *i.e.* the bottom-of-stack symbol can only occur at the bottom of the stack, and is never popped nor rewritten. A configuration of $\mathcal{P}$ is a pair (q, σ) with $q \in Q$ and $\sigma \in \bot(A \backslash \{\bot\})^*$.

***Remark* 21.** *In general an abstract pushdown automaton cannot be finitely described, as the domain of δ is infinite and no further assumption is made on δ.*

Example 24

An order-1 pushdown automaton is an abstract pushdown automaton whose stack alphabet is finite.

An abstract pushdown automaton $\mathcal{P}$ induces an Lts whose domain is the set of configurations of $\mathcal{P}$ and whose transitions are defined by the transition relation δ, *i.e.*, from a vertex $(q, \sigma \cdot a)$ one has an s-transition to $(q', \sigma \cdot \sigma')$ whenever $(q', \sigma') = \Delta(q, a, s)$.

Example 25

Link free n-CPDA (*i.e.* n-CPDA that do not create n-links) are special cases of abstract pushdown automata. Indeed, let $n > 1$ and consider such an order-n CPDA $\mathcal{A} = \langle \Sigma, \Gamma, Q, \delta, q_0 \rangle$. Let A be the set of all order-$(n-1)$ stacks over Γ, and for every $p \in Q$, $a \in A$ with $\gamma = \mathit{top}_1(a)$ and $s \in \Sigma$, we define $\delta'(p, a, \sigma)$ to be

$$\delta'(p, a, \sigma) = \begin{cases} (q, \varepsilon) & \text{if } \delta(q, \gamma, s) = (q, \mathit{pop}_n) \\ (q, a \cdot a) & \text{if } \delta(q, \gamma, s) = (q, \mathit{push}_n) \\ (q, \mathit{op}(a)) & \text{if } \delta(q, \gamma, s) = (q, \mathit{op}) \text{ with } \mathit{op} \notin \{\mathit{pop}_n, \mathit{push}_n\} \end{cases}$$

It easily follows that $\mathcal{A}$ and the abstract pushdown automaton $\langle \Sigma, A, Q, \delta', q_0 \rangle$ generates isomorphic Lts.

Considering a partition $Q_E \uplus Q_A$ of Q between Éloïse and Abélard together with a colouring function $\mathrm{Col} : Q \to$ Colours one defines a parity game on the Lts generated by $\mathcal{P}$. Such a game is called an ***abstract pushdown game***.

It follows from Example 24 and Example 25 that both pushdown games and link-free CPDA games fit in the setting of abstract pushdown games.

In [Wal96] and later in [Ser04a] a game reduction from a pushdown parity game to a parity game on a finite arena is given. It turns out that the construction and the proof of the validity of the reduction do not make any assumption of the stack alphabet being finite. Hence, it remains valid for any abstract pushdown games as we explained in [CMHOS08] but of course the reduced game is no longer finite.

Of special interest is the following result. In particular the second point permits to achieve the third step in the proof schema of Theorem 24, *i.e.* building an order-$(n-1)$ CPDA game $\widetilde{\mathbb{G}}$ such that $\mathbb{G}_{\mathrm{lf}} \preccurlyeq \widetilde{\mathbb{G}}$.

Proposition 4 [CMHOS08; BCOS10]

- Applying the game reduction of [Wal96; Ser04a] to an n-HOPDA parity game (seen as an abstract pushdown game) leads to a reduced game that is an (exponentially larger) $(n-1)$-HOPDA parity game
- Applying the game reduction of [Wal96; Ser04a] to a link-free n-CPDA parity game (seen as an abstract pushdown game) leads to a reduced game that is an (exponentially larger) $(n-1)$-CPDA parity game

***Remark* 22 ([CMHOS08]).** *Proposition 4 leads to an alternative simpler proof of the decidability of HOPDA parity games [Cac03b].*

11 Back to Logic

We now discuss corollaries of Theorem 24. The proofs of these corollaries mainly rely on well known connections between parity games and μ-calculus, see *e.g.* [AN01; Wil01; Wal04].

11.1 Labelled Transition Systems

Decidability of μ-calculus model-checking for Lts generated by CPDA (Theorem 18) follows directly from point (1) of Theorem 24. Concerning global model-checking, one can deduce from point (2) of Theorem 24 (and some extra work) the following result.

Corollary 7 [BCOS10]

The μ-calculus definable sets on Lts generated by CPDA are exactly the regular sets of configurations.

In the setting of HOPDA the following result generalises previous work [BM04; HO08] and also contrasts with similar (but different) results for MSO [Car05; Car06] (for which one should replace μ-calculus by MSO and "regular" by "rational")

Corollary 8 [CMHOS08]

The μ-calculus definable sets on Lts generated by HOPDA are exactly the regular sets of configurations.

By showing that CPDA are closed under regular tests[12] one obtains the following result.

Theorem 25 (μ-Calculus effective reflection for CPDALTS) [BCOS10]

Let $\mathcal{A}$ be an n-CPDA and let φ be a μ-calculus formula. There is an algorithm that transforms any n-CPDA $\mathcal{A}$ and any μ-calculus formula φ into another n-CPDA $\mathcal{A}_\varphi$ such that the Lts generated by $\mathcal{A}_\varphi$ is isomorphic to the Lts generated by $\mathcal{A}$ on which the elements where φ holds have been marked.

11.2 Trees

Concerning trees generated by schemes or CPDA, the decidability of μ-calculus model-checking (Theorem 13) follows directly from point (1) of Theorem 24.

If we combine the careful analysis of the complexity of solving CPDA parity games together with the complexity of the size of a CPDA obtained from a recursion scheme, we obtain the same *fixed-parameter tractable* complexity as in [KO09].

Corollary 9 [CS12a]

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

The μ-calculus effective reflection (Theorem 19) is a consequence of the Equi-expressivity theorem (Theorem 8) together with Theorem 25. Finally, the effective selection property (Theorem 23 recalled below) is a consequence of point (3) of Theorem 24 (namely the existence of a strategy realised by a synchronised CPDA-transducer) together with the Equi-expressivity theorem. We now give some sketch of proof for sake of completeness.

Theorem 23 (Effective MSO selection property)[CS12a]

(Labelled) Recursion schemes as well as CPDA have the effective MSO selection property.

Sketch of proof. Let $\varphi(X_1, \cdots, X_\ell)$ be a monadic second order formula with ℓ second-order free variables, and let $\mathcal{S} = \langle \Sigma, N, \mathcal{R}, Z, \bot \rangle$ be an order-n (labelled) recursion scheme.

Relying on Theorem 8, we consider an n-CPDA $\mathcal{A}$ that generates the same tree as $\mathcal{S}$.

Thanks to the well-known equivalence between logic and tree automata, there is a nondeterministic parity 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[13] that acceptance of a tree by a nondeterministic parity 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 Abélard 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 Abélard

[12] *I.e.* given a CPDA and a regular set of configurations one can construct another CPDA that behaves as the first one except that its control states have an extra component that permits at any moment to detect whether the projection by forgetting this extra component of the actual configuration belongs to the given regular set of configurations.

[13] For a more detailed exposition see Section 2.3 of Chapter 3.

moves the pebble to some son and update the state attached to the pebble according to the transition chosen by Éloïse. In case the pebble reaches a leaf, the play ends and Éloïse wins iff the state is final (we have final states in the tree automaton to handle finite branches); otherwise the play is infinite and Éloïse wins iff the smallest infinitely visited priority is even.

For the tree t_φ, the underlying arena of the previous game is essentially a synchronised product of the 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_φ 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, it yields to 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 iff 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.

Apply Theorem 24 to $\mathbb{G}$. Then either Éloïse has no winning strategy from the initial configuration $(q_0, \perp_n)$ and we are done (there is no selector). Otherwise one can effectively construct an n-CPDA transducer $\mathcal{T}$ *synchronised* with $\mathcal{A}'$ realising a well-defined *winning* strategy for Éloïse in $\mathbb{G}$ from $(q_0, \perp_n)$. As $\mathcal{A}'$ and $\mathcal{T}$ are synchronised, we can consider their synchronised product; call it $\mathcal{A}''$. Hence in $\mathcal{A}''$ the configurations contain extra informations (coming from $\mathcal{T}$); in particular, for any configuration, if the control state from the $\mathcal{A}'$ component is controlled by Éloïse, then the control state from the $\mathcal{T}$ component provides 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 $\mathcal{T}$: 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.

Finally, as we can transform $\mathcal{A}_\varphi$ back to a labelled recursion scheme, we get $\mathcal{S}_\varphi$ as expected. ■

12 Saturation Techniques and the C-SHORe Tool

As a motivation for this section, think back to the toy example of Section 1. The general goal was to check the correctness of a program and in our example, this meant ensuring that the program never attempts to commit an error. The roadmap was first to transform the recursion scheme $\mathcal{S}$ into an equivalent CPDA $\mathcal{A}$ and then to reason directly on the latter. Of course, as one is interested in checking the (non-) existence of a bad execution, the natural way to check correctness is to consider the synchronised product of $\mathcal{A}$ with a finite word automaton $\mathcal{B}$ describing all those forbidden behaviours. This product yields a CPDA and one is left with checking whether there is a path from the initial configuration to a configuration in which the $\mathcal{B}$'s component is an accepting state (in that case there is an execution of the CPDA that is violating the specification).

Hence, the general problem we address is the following. Take as an input a CPDA $\mathcal{A}$ (typically obtained from a scheme and a description by a finite word automaton of forbidden behaviours) and a set Bad of configurations of $\mathcal{A}$ (typically those that are reached after a forbidden behaviour), and output a description of $Pre^*_{\mathcal{A}}(Bad)$ of all configurations of $\mathcal{A}$ from which there exists a path to an element in Bad. Note that if $\mathcal{A}$ and Bad are coming from a program and a specification, the program is correct if and only if the initial configuration of $\mathcal{A}$ does not belong to $Pre^*_{\mathcal{A}}(Bad)$.

It is easily seen that if Bad is a regular set of stacks (in the sense of Section 10.3) so does $Pre^*_{\mathcal{A}}(Bad)$ and this latter set can be computed using Theorem 24 as it is nothing else but the

winning region of a one-player CPDA reachability game. However, the solution provided by Theorem 24 (if we unravel the previously sketched proof…) constructs first a huge object (of size a tower of exponential of height $(n-1)$ as we start with a one-player game) on which later a reachability game is then solved. Therefore, the worse case time and space complexities are always reached which is not tractable in practical situations.

For this reason, motivated by eventually implementing a model-checker, we considered in [BCHS12] a saturation method to solve the problem. The general idea is first to note that $Pre^*_{\mathcal{A}}(Bad)$ can be defined as the smallest set such that

- $Pre^*_{\mathcal{A}}(Bad) \supseteq Bad$ and
- $Pre^*_{\mathcal{A}}(Bad) \supseteq \{(q,s) \mid \exists (q',s') \in Pre^*_{\mathcal{A}}(Bad) \text{ and } (q,s) \rightarrow (q',s')\}$.

Now assume that Bad is a regular set of configurations described by an automaton $\mathcal{B}_0$ (as explained in Section 10.3). Then (see Figure 24 for an illustration) a ***saturation algorithm*** consists in iterating a *saturation function* Π — adding new transitions to $\mathcal{B}_0$ — until a fixed point is reached, *i.e.* we cannot find any more transition to add. Hence, this leads to construct a (finite) sequence of automata $\mathcal{B}_0, \ldots, \mathcal{B}_k$ defined by letting $\mathcal{B}_{i+1} = \Pi(\mathcal{B}_i)$ until $\mathcal{B}_{i+1} = \mathcal{B}_i$. The saturation algorithm is correct if one has $Pre^*_{\mathcal{A}}(L(\mathcal{B}_0)) = L(\mathcal{B}_k)$, where $L(\mathcal{B}_i)$ denotes the set of configurations of $\mathcal{A}$ accepted by $\mathcal{B}_i$.

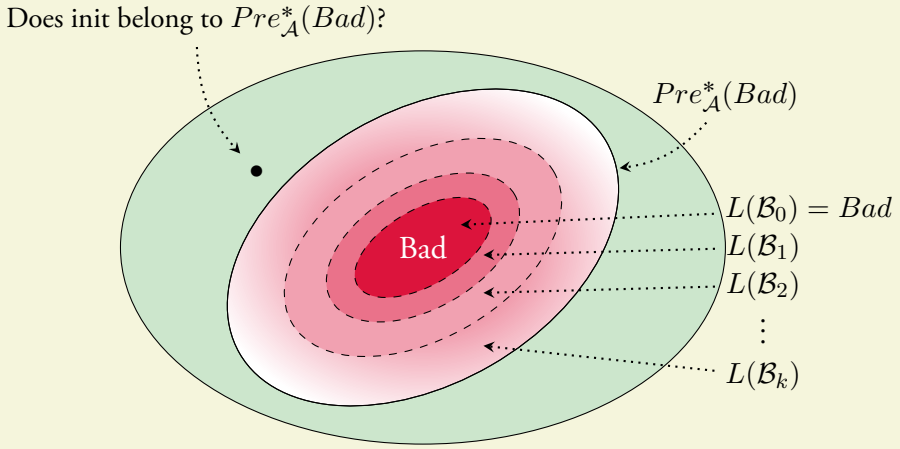


Figure 24 – A saturation algorithm computing the set $Pre^*_{\mathcal{A}}(Bad)$.

In [BCHS12] we presented a saturation algorithm computing the set $Pre^*_{\mathcal{A}}(L(\mathcal{B}_0))$. Note that for that we had to consider an variant of automata recognising stacks with links where one reads the stack in a top-down manner (hence the automata need to be non-deterministic and actually for simplicity we use *alternating* automata). We do not give here the saturation function as it would require to introduce the new model of automata recognising stacks with links and lead to many technicalities.

Theorem 26 [BCHS12]

There is a saturation based algorithm that takes as an input a CPDA $\mathcal{A}$ and an automaton on stacks with links $\mathcal{B}$ and outputs an automaton on stacks with links $\mathcal{B}'$ such that $L(\mathcal{B}') = Pre^*_{\mathcal{A}}(L(\mathcal{B}))$.

Moreover the algorithm runs in $(n-1)$-ExpTime and in all cases is linear in the size of the stack alphabet.

***Remark* 23.** *Regarding complexity one should first stress that in many situations (see later a discussion on the C-SHORe tool) the running time is much faster than the worse case* $(n-1)$-ExpTime *bound. Note that the fact that it works in linear time with respect to the size of the stack alphabet means that it is fixed-parameter tractable. Also recall that in the translation from a recursion scheme to a CPDA the number of states is bounded by the arity of the program (and that the stack alphabet is linear in the size of the scheme): hence, the speed of the algorithm is not strongly impacted by the length of the program / size of the scheme; the costly parameter being the arity of the program which is small in realistic situations.*

***Remark* 24.** *In [BCHS12] we actually give a more general saturation algorithm than the one of Theorem 26 as we consider alternating CPDA (*i.e. *we consider a two-player reachability game and compute its winning region).*

***Remark* 25 (Related work — Saturation techniques).** *The saturation technique has proved popular in the literature. It was introduced by Bouajjani et al. [BEM97] and Finkel et al. [FWW97] and based on a string rewriting algorithm by Benois [Ben69]. It has since been extended to Büchi games [Cac03a], parity and μ-calculus conditions [HO10], and concurrent systems [SES08; Ati10], as well as weighted pushdown systems [RSJM05]. In addition to various implementations (by tools such as Moped [Sch02] and PDSolver [HO10]), efficient versions of these algorithms have also been developed [EHRS00; SSE06].*

12.1 C-SHORe Tool and Optimisations

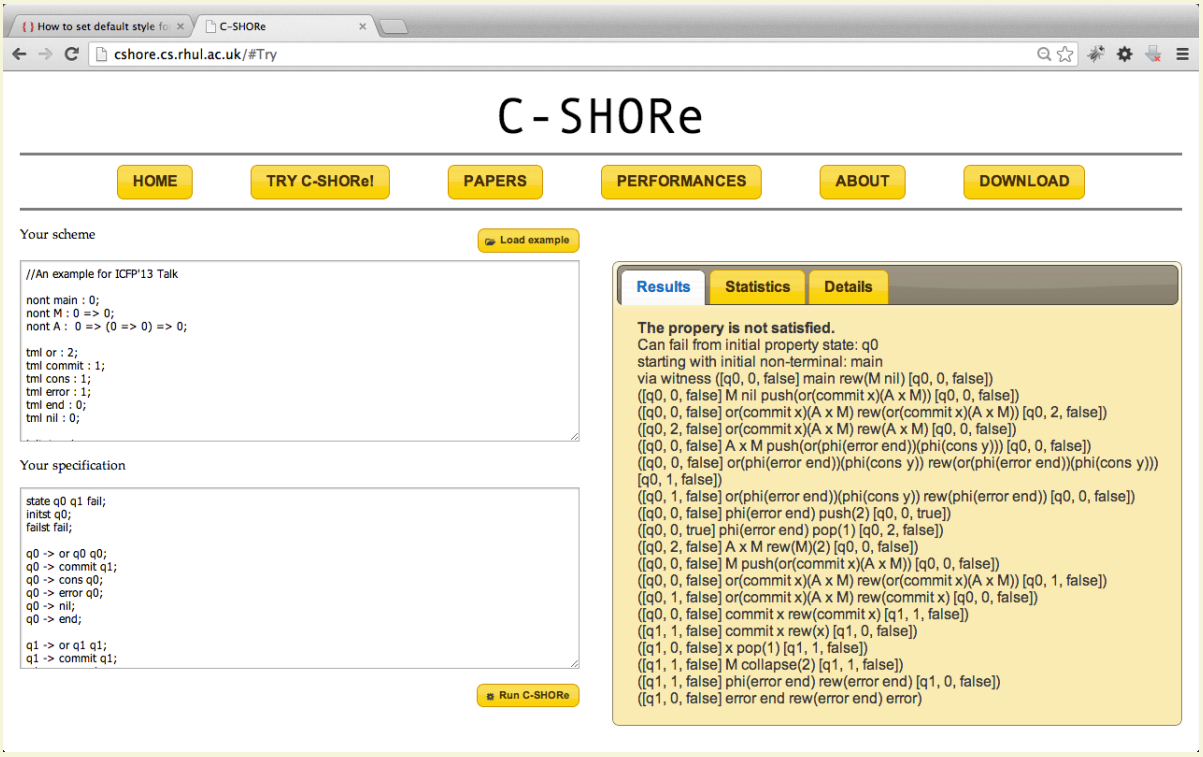


Figure 25 – Screenshot from C-SHORe web interface (see `http://cshore.cs.rhul.ac.uk`

Following [BCHS12] Matthew Hague and Christopher Broadbent developed the tool C-SHORe that checks whether whether a CPDA can reach from its initial configuration a state with an error state (hence, it typically permits to check safety property of programs described as a recursion scheme as explained previously). The tool is currently hosted at `http://cshore.cs.rhul.ac.uk`. This tool was presented in [BCHS13] together with many substantial modifications and additions to the algorithm from [BCHS12], leading to several novel practical and theoretical contributions:

1. An approximate *forward* reachability algorithm providing data
   a) allowing for the CPDA to be pruned so that saturation receives a smaller input;
   b) employed by a modified saturation algorithm to guide its *backward* search.

   This is essential for termination on most of the benchmarks.
2. A method for extracting witnesses to reachability[14].
3. A complete reworking of the saturation algorithm of [BCHS12] that speeds up the fixed-point computation[15].
4. Experimental results showing that our approach compares well with TRecS, GTRecS(2) and TravMC.

We now briefly discuss related works as well as points 1 and 4 from the above list (a reader interested in points 2 and 3 should read [BCHS13]; indeed, explaining them here would require to give the full saturation algorithm hence, require many extra definitions).

12.1.1 Checking Properties of Recursion Schemes in Practice: Related Work

Naoki Kobayashi's TRecS [Kob09a] tool, which checks properties expressible by a deterministic trivial Büchi automaton (all states accepting), was the first to check properties of recursion schemes "in practice". It works by determining whether a recursion scheme is typable in an intersection-type system characterising the property to be checked [Kob09b]. In a bid to improve scalability, a number of other algorithms have subsequently been designed and implemented such as Kobayashi et al.'s GTRecS2 [Kob11a; Kob12] and Neatherway et al.'s TravMC [NRO12] tools, all of which remain based on intersection type inference.

This work is the basis of various techniques for verifying functional programs. Kobayashi et al. have developed MoCHi [KSU11] that checks safety properties of (OCaML) programs, and EHMTT Verifier [UTK10] for tree processing programs. Both use a model-checker for recursion schemes as a central component. Similarly, Ramsay and Ong [OR11] provide a verification procedure for programs with pattern matching employing recursion schemes as an abstraction.

Despite much progress, even the state-of-the-art TRecS does not scale to recursion schemes big enough to model realistically sized programs; achieving scalability while accurately tracking higher-order control-flow is a challenging problem.

C-SHORe offers an automata-theoretic perspective on this challenge, providing a fresh set of tools that contrast with previous intersection-type approaches.

[14] Indeed, the proof of soundness in [BCHS12] is not constructive and proving the correctness (and in particular termination) of counter-example extraction is non-trivial.

[15] We introduce an efficient method of computing the fixed point in the saturation algorithm, inspired by Schwoon et al.'s algorithm for alternating (order-1) pushdown systems [SSE06]. Rather than checking all CPDA rules at each iteration, we fully process *all* consequences of each new transition at once. New transitions are kept in a set Δ_{new} (implemented as a stack), processed, then moved to a set Δ_{done}, which forms the transition relation of the final stack automaton. In most cases, new transitions only depend on a single existing transition, hence processing the consequences of a new transition is straightforward. The key difficulty is the $push_k$ rules as they depend on *sets* of existing transitions.

12.1.2 Initial Forward Analysis

For a CPDA $\mathcal{A}$ with initial configuration $Init$ call $Post^*_{\mathcal{A}}(Init)$ the set of configurations reachable by $\mathcal{A}$ from $Init$. Note that this set is in general not a regular set of configurations hence, cannot be represented precisely by an automaton [BM04] (for instance using $push_2$, we can create the stack $[[a^n]_1[a^n]_1]_2$ from $[[a^n]_1]_2$ for any $n \geqslant 0$).

It is generally completely impractical to compute $Pre^*_{\mathcal{A}}(Bad)$ in full (most non-trivial examples considered in our experiments would time-out). For our saturation algorithm to be usable in practice, it is therefore essential that the search space is restricted, which we achieve by means of an initial forward analysis of the CPDA. Ideally we would compute only $Pre^*_{\mathcal{A}}(Bad) \cap Post^*_{\mathcal{A}}(Init)$. Since this cannot be represented by an automaton, we instead compute a sufficient approximation T (ideally a *strict* subset of $Pre^*_{\mathcal{A}}(Bad)$) where (see Figure 26):

$$Pre^*_{\mathcal{A}}(Bad) \;\cap\; Post^*_{\mathcal{A}}(Init) \;\subseteq\; T \;\subseteq\; Pre^*_{\mathcal{A}}(Bad)$$

The initial configuration will belong to T if and only if it can reach a configuration in Bad. Computing such a T is much more feasible.

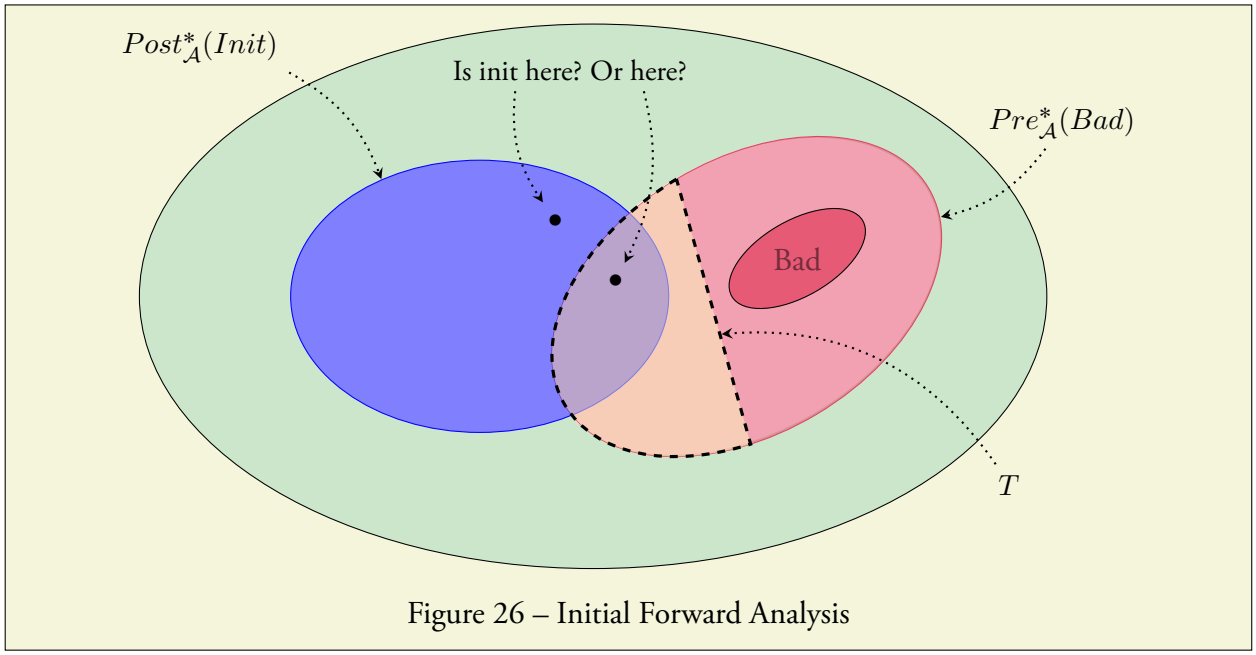


Figure 26 – Initial Forward Analysis

We first compute an over-approximation of $Post^*_{\mathcal{A}}(Init)$. For this we use a *summary algorithm* [SP81] (that happens to be precise at order-1) from which we extract an over-approximation of the set of CPDA transition rules that may be used on a run to Bad. Let $\mathcal{A}'$ be the (smaller) CPDA containing only these rules. That is, we remove all rules that we know cannot appear on a run to Bad. We could thus take $T = Pre^*_{\mathcal{A}'}(Bad)$ (computable by saturation for $\mathcal{A}'$) since it satisfies the conditions above. This is what we meant by *"pruning"* the CPDA — 1. a) in the list on page 85.

However, we further improve performance by computing an even smaller T — 1. b) in the list on page 85. We extract contextual information from our over-approximation of $Post^*_{\mathcal{A}}(Init)$ about how pops and collapses might be used during a run to Bad. Our $\mathcal{A}'$ is then restricted to a model $\mathcal{A}''$ that "guards" its rules by these contextual constraints. Taking $T = Pre^*_{\mathcal{A}''}(Bad)$ we have a T smaller than $Pre^*_{\mathcal{A}'}(Bad)$, but still satisfying our sufficient conditions. $\mathcal{A}''$ is what we call a *"guarded CPDS"* (defined in the next subsection). We cannot compute $Pre^*_{\mathcal{A}''}(Bad)$ precisely for a guarded CPDS, but we can adjust saturation to compute T such that $Pre^*_{\mathcal{A}''}(Bad) \;\subseteq\; T \;\subseteq\; Pre^*_{\mathcal{A}'}(Bad)$. This set thus also satisfies our sufficient conditions.

12.1.3 Experimental Results

In [BCHS13][16] we compared C-SHORe with the current state-of-the-art verification tools for higher-order recursion schemes: TRecS [Kob09a], GTRecS2 [Kob12] (the successor of [Kob11a]), and TravMC [NRO12]. Benchmarks are from the TRecS and TravMC benchmark suites, plus several larger examples provided by Naoki Kobayashi (generated with MoCHi [KSU11]). The majority of the TravMC benchmarks were translated into recursion schemes from an extended formalism, higher-order recursion schemes with Case statements, using a script by Kobayashi. For fairness, all tools in our experiments took a pure higher-order recursion scheme as input. However, the authors of TravMC report that TravMC performs faster on the original higher-order recursion schemes with Case examples than on their higher-order recursion schemes translations.

In all cases, the benchmarks consist of a higher-order recursion scheme (generating a computation tree) and a property automaton. In the case of C-SHORe, the property automaton is a regular automaton describing branches of the generated tree that are considered errors. Thus, following the intuition developed at the beginning of Section 12, we can construct a reachability query over a CPDA, where the reachability of a control state q_{bad} indicates an erroneous branch. All other tools check co-reachability properties of higher-order recursion schemes and thus the property automaton describes only valid branches of the computation tree. In all cases, it was straightforward to translate between the co-reachability and reachability properties.

Of 26 benchmarks, C-SHORe performed best on 5 examples. In 6 cases, C-SHORe was the slowest. In particular, C-SHORe does not perform well on two examples that belong to a class of benchmarks that stress higher-order model-checkers and indicate that our tool currently does not always scale well. However, C-SHORe seems to show a more promising capacity to scale on larger HORS produced by tools such as MoCHi [KSU11], which are particularly pertinent in that they are generated by an actual software verification tool. We also note that C-SHORe timed out on the fewest examples despite not always terminating in the fastest time.

It is also very important to note that C-SHORe and GTRecS2 are the only implemented fixed-parameter tractable algorithms in the literature for HORS model-checking of which we are aware (both TRecS and TravMC have worst-case run-times non-elementary in the size of the recursion scheme). Moreover, C-SHORe generally performs much better than GTRecS2. Thus not only does C-SHORe's performance seem promising when compared to the competition, there is also theoretical reason to suggest that the approach could in principle be scalable, in contrast to some of the alternatives. Thus initial work justifies further investigation into saturation based algorithms for higher-order model-checking.

Finally, we remark that without the forwards analysis sketched in Section 26, all shown examples except one timed out. We also note that we did not implement a naive version of the saturation algorithm (*i.e.* one without the optimisation mentioned in point 2 in the list on page 85) where after each change to the stack with links automaton, each rule of the CPDA is checked for further updates. However, experience implementing PDSolver [HO10] (for order-1 pushdown systems) indicates that the naive approach is at least an order of magnitude slower than the techniques [SSE06] we generalised for our purpose.

13 Open Problems and Perspectives

We now list several natural problems as well as perspectives related to the topics we have discussed in the present chapter.

[16] See also `http://cshore.cs.rhul.ac.uk` for up-to-date results.

13.1 Comparing the Models and Approaches

In this document, when studying model-checking problems for recursion schemes, we exclusively considered the approach using CPDA. However, alternative approaches have been developed and been very successful. The first one was the one by Ong using the notion of traversals and game semantics [Ong06]. Some years later, following initial ideas by Aehlig [Aeh06], Kobayahsi [Kob09b] and Kobayashi and Ong [KO09] developed an approach using intersection types which lead to another proof of the decidability of MSO and is at the basis of the tools GTRecS2 [Kob11a; Kob12] andTravMC [NRO12]. Finally, in [SW11] Salvati and Walukiewicz used Krivine machines [Kri07] as an abstract model to represent the sequence of rewriting of a scheme (actually, the authors work with the equivalent formalism of the λY-calculus). They also provide in [SW12] a translation from λY-calculus to CPDA.

Hence, a thankless but important task would be to compare and to evaluate the pro and cons of the following approaches:

- traversal and game semantics;
- collapsible pushdown automata;
- intersection types;
- λY-calculus and Krivine machines.

13.2 Pushing the Decidability of Model-Checking

In a sense, Theorem 18 can be considered as the strongest result in its line of research as (i) CPDALTS are one of the most general class of Lts with a decidable μ-calculus model-checking, and (ii) μ-calculus is one of the most expressive logics that is decidable on CPDALTS. Hence, pushing the decidability result of Theorem 18 can be done in two directions, both representing an exciting challenge.

- A first approach can be to generalise the class of CPDALTS while preserving μ-calculus decidability. So far there are no natural candidate for this question. Moreover, one should extend the class in a natural way to preserve the connections with applications: a typical way would be to add a natural feature from functional higher-order languages that is not yet captured by the setting of recursion schemes/CPDA (note that using different techniques such an approach was considered in [OR11] for programs with pattern matching employing recursion schemes as an abstraction).

- A second approach is to consider a more expressive logic than μ-calculus that would admit a decidable model-checking problem for the class of CPDALTS. Of course, due to Theorem 15, this logic must not permit to capture FO(TC). A natural candidate here is to consider the extension of μ-calculus where one allows back-modalities (equivalently one considers the usual μ-calculus but on an enriched Lts obtained by adding for every relation $\xrightarrow{a}$ a relation $\xrightarrow{\overline{a}}$ defined by $s \xrightarrow{\overline{a}} t$ if and only if $t \xrightarrow{a} s$). We conjecture that this logic is still decidable against the class CPDALTS, possibly requiring an extra exponential blow-up.

13.3 Regularities

The notion of regularity used in this document can capture the μ-calculus definable sets of configurations, in particular for HOPDA (see Theorem 20). But in the setting of HOPDA, MSO-definable sets are also making sense (recall that HOPDA have decidable MSO theories) and they have been captured by a stronger notion of regularity independently introduced by Carayol [Car05] and Fratani [Fra06]. Hence a natural question is the decidability of the following problem: given a strong regular set (in the sense of [Car05; Fra06]), decide whether it is regular (in the sense of this document). Moreover, it is known from [Fra06] that positional winning strategies of higher-order pushdown parity game can be described using strong regular sets (*i.e.* for each transition, the set of configurations from which the strategy plays the transition is a strong regular set) and a k-ExpTime algorithm was given in [CS08]. Hence, one can also wonder whether it is possible to describe positional winning strategies using our weaker notion of regularity.

Another related question is whether one can adapt to CPDA the notion of regularity defined by Carayol and Fratani. Of course, as CPDA have MSO undecidable theories, one would either loose effectivity or connection between this stronger notion of regularity and the MSO definability. A natural candidate is to introduce symmetrical stack operations as in the setting of HOPDA.

13.4 Shape Preserving Transformation

A very promising line of research is the quest for *shape preserving transformation*. It was initiated by Axel Haddad in his PhD [Had13] and still needs to be pursued. As an example, consider the following scheme transformation that from a scheme $\mathcal{S}$ requires to construct another scheme $\mathcal{S}_\varphi$ that generates a selector (for some given formula φ) in the tree generated by $\mathcal{S}$. Such a transformation is *shape preserving* if the scheme $\mathcal{S}_\varphi$ is obtained from the original one by adding duplicated and annotated version of the terminals (*i.e.* the symbols used in $\mathcal{S}_\varphi$ are of the form x^θ where x is a symbol used in $\mathcal{S}$ and θ is an annotation). As the structure of the new scheme is an extension of the original one, it permits to easily retrieve the original scheme from the new scheme.

There is one immediate interest in this approach. Assume that the scheme $\mathcal{S}$ was designed by some programmer (or more realistically was "compiled" from a program written by a human). Assume that this programmer specifies property φ and ask for $\mathcal{S}_\varphi$. Again one can think of $\mathcal{S}_\varphi$ as a selector but for this illustration think of it as some weaker object more specifically as a scheme generating a tree reflecting property φ (and think of φ as a property expressing wrong behaviours of the program). From the answer $\mathcal{S}_\varphi$ the programmer wants to correct his original program, *e.g.* by cutting those pieces of codes leading to incorrect behaviours (the ones expressed by φ). If the resulting scheme $\mathcal{S}_\varphi$ is obtained as presented in this document, *i.e.* via a round-trip through CPDA, it has no more similarities with the original scheme $\mathcal{S}$ and thus the programmer would get totally lost. But if one can do the process by a shape preserving transformation (see [Had13] for such a solution), then there is a reasonable hope that the programmer can modify his code as expected.

So far the work of Haddad is purely at the theoretical level but as it relies on the intersection type approach of Kobayashi and Ong [KO09] and as this approach was later successfully used in the tools GTRecS2 [Kob11a; Kob12] andTravMC [NRO12] there is some hope that, for a reasonable subset of properties, one can enrich one of these tools to perform shape preserving transformation for *e.g.* reflection (or in a first step for a basic kind of properties like the one we discussed in Section 9.2.4 that asks for removing divergent computations and that Haddad also tackled in his PhD).

13.5 *À la* Caucal Hierarchies for Recursion Schemes and Transfer Theorem

The approach developed by Caucal (see Section 8.3) in [Cau02] is probably one of the most elegant in the field. Unfortunately, as stated in Theorem 5 (resp. Theorem 6) the trees (resp. Lts) hierarchy only permits to capture the trees (resp. Lts) generated by *safe* schemes.

Of course as the Lts in the class CPDALTS do not have MSO decidable theories, at least one of the operations used to define these new hierarchies should not preserve MSO decidability.

On a related topic one can also investigate *transfer theorems*, *i.e.* to look for transformations (say from schemes to schemes) that preserve the MSO decidability (*i.e.* the resulting scheme has a decidable MSO theory if the original one had and this is effective). See [SW13a] for recent results on this topic.

13.6 An Equi-Expressivity Result for IO-Schemes

The equi-expressivity Theorem (See Theorem 8) states that CPDA are equi-expressive with schemes for generating trees when one considers the Outermost-Innermost evaluation policy (equivalently the unrestricted policy) for schemes. As mentionned in Remark 3, other evaluation policies for scheme make sense, in particular the Innermost-Outermost (IO) policy which is a scheme analog of the "call-by-value" evaluation of programs (while OI corresponds to "call-by-name").

Of course, a transformation from IO-schemes to OI-schemes exists (see [Had13] for a shape preserving transformation doing this) and therefore one can rely on them to handle algorithmic questions on IO-schemes. However, it would be interesting to work directly on IO-schemes using an automata model (or a Krivine machine in the spirit of the work of Salavti and Walukiewicz [SW12]): it would not only bring more understanding on how IO-schemes behave but it may also lead to define a new, intrinsically interesting, class of automata.

Hence, finding an IO-counterpart to the equi-expressivity Theorem is, in our opinion, a challenging and natural open problem.

13.7 The Equivalence Problem for Higher-Order Recursion Schemes

Recall that Nivat defined a notion of equivalence for program schemes: two schemes are equivalent if and only if they compute the same function under *any* interpretations. Later, Courcelle and Nivat [CN78] showed that two schemes are equivalent if and only if they generate the same infinite term tree. Then, following the work by Courcelle [Cou78a; Cou78b] the equivalence problem for schemes is known to be interreducible to the problem of decidability of language equivalence between deterministic pushdown automata (DPDA). Research on the equivalence for program schemes was halted until Sénizergues [Sén97; Sén02] established decidability of DPDA equivalence (see also simplifications by Stirling [Sti01; Sti00a] and Jancar [Jan12]) which therefore also solved the scheme equivalence problem.

Thanks to the equi-expressivity theorem, the equivalence problem for higher-order recursion schemes is interreducible to the problem of decidability of language equivalence between deterministic collapsible pushdown automata. Even if this problem is probably extremely hard, we believe that it is one of the most exciting open problem in the field and we expect that the recent simplification introduced by Jancar in the DPDA equivalence problem may lead some hint for generalisation to the higher-order setting (a first step being focussing on order-2 CPDA rather than general CPDA where some work by Stirling has already been done [Sti06]).

13.8 Graph Grammars for Collapsible/Higher-Order Pushdown Graphs

Deterministic graph grammars are a form of graph rewriting systems generalising to graphs the well-known notion of context-free grammars on words. In this case, grammar rules no longer express the replacement of a non-terminal letter by a string of terminal and non-terminal letters, but that of a non-terminal arc (or more generally hyperarc) by a finite graph (or hypergraph) possibly containing new non-terminals, thus generating larger and larger graphs. A graph grammar is deterministic if there is only one production rule for every non-terminal hyperarc. Consequently, from a given axiom, a grammar does not generate a set of graphs but a unique infinite graph. See [Cau07] for a reference on this topic.

These deterministic graph grammars correspond to the finite systems of equations over HR graph operators originally defined by Courcelle [Cou89]. The graphs they generate are called regular or HR-equational and they encompass the configuration graphs of pushdown automata. The link between deterministic graph grammars and the graphs defined by pushdown automata can be traced back to the work of Muller and Schupp [MS85].

A motivation for considering graph grammars instead of pushdown automata is that they provide a more natural (and graphical) description of the structure of the graphs. This is actually very convenient for instance when working with positional strategies which are nothing else but sub-graphs of the game graph and permits to develop saturation algorithms. In particular in an unpublished work with Arnaud Carayol and Didier Caucal [CCS14] we gave a saturation algorithm (made of only 5 simple rules) that takes as an input a graph grammar describing a two-player reachability game and produces as an output a graph grammar describing the same game on which is "marked" a positional winning strategy for the first player (following the terminology of this document this is an endogenous approach of the problem of computing a positional winning strategy in a reachability game described by a deterministic graph grammar).

This is to compare with the work in [BCHS12] where we have shown that saturation methods can be used to compute the winning region of CPDA reachability games (remember that the techniques do not permit to compute a winning strategy). Hence, one way of merging these two works would consist in first defining a notion of graph grammars capturing collapsible/higher-order pushdown graphs and then extending our saturation based algorithm for this richer notion.

13.9 Saturation Techniques and Tools

The saturation algorithm we presented in Section 12 lead an efficient and scalable implementation (the tool C-SHORe). This practical success is an invitation to enrich the tool and to keep developing saturation methods to handle CPDA. A first step should be to deal with more general properties than reachability, in particular Büchi conditions and possibly later parity condition. The main difficulties here are in the nesting of fixpoints (smallest and greatest fixpoints): one is theoretical (but previous work on weaker models [Hag08; HO11] suggests that it is doable; the difficulty here is probably in finding the right framework to write "simple" proofs) and another one is practical (it is not clear how one can increase the complexity of the tool while preserving its scalability). Another step should be to deal with more general problems than (global) model-checking, typically synthesis: here the difficulty is that there is no known saturation algorithm even on pushdown automata that computes a positional winning strategy (moving to graph grammars could be an option. See Section 13.8).

More generally, higher-order program analysis is an increasingly active and exciting topic in which the methods developed in this document perfectly fit. The next step to get closer to practice is probably to work on a "real" language (maybe not an existing one but one that would be designed

for that purpose and that, contrarily to recursion schemes, would look like a real programming language). One should also investigate abstract interpretation methods to handle variables ranging on infinite domains.

Finally, still in the quest for real applications, one should consider how to handle standard features like concurrency (see promising work by Hague on that topic [Hag13]) or probabilistic choices.

13.10 Boundedness Conditions

A natural winning condition in a pushdown game is to require that the stack height along a play is unbounded. One can show that if Éloïse has a winning strategy in such a game then she has one that is positional and permits to strictly unbound the stack height (*i.e.* the stack height converges to $+\infty$) (see [CDT02; Ser04a]). Again on pushdown graphs there was some work where one considers as a winning condition a Boolean combination of a boundedness condition on the stack together with a parity condition [BSW03; Gim04]. The motivation is fairly clear: the parity component specifies some ω-regular specification on the system represented by the pushdown game while the (un)boundedness condition specifies some bound on the memory used by the system.

In this spirit, it sounds natural to investigate, as a natural example of a non-regular specification on CPDA, those CPDA games where the winning condition requires that the stack is bounded/unbounded. Of course as one now deals with higher-order stacks, unboundedness can be declined in several ways. Let us mention two of them.

- Strict unboundedness: the sequence of higher-order stacks visited along the play admits a limit that is infinite.
- Unboundedness: the sequence of higher-order stacks visited along the play contains stacks of arbitrarily large size.

One can easily note the following result.

Proposition 5 [Serre, unpublished]

Winning a HOPDA game with an unboundedness condition may require infinite memory.

Concerning (a slight variant of) strict unboundedness one can prove the following result.

Theorem 27 [Serre, unpublished]

One can decide in a CPDA game whether Éloïse has a winning strategy to ensure (a *slight variant* of) strict unboundedness.

Whether Theorem 27 can be extended to other notions of unboundedness is open as well as whether one can consider combinations of unboundedness conditions and parity conditions while preserving decidability.

One special interest in studying unboundedness conditions, come from their connection with a result obtained in [Ser04b; Ser06a]. More specifically, in that work one contribution was to provide languages of ω-words denoted by $L(\mathcal{A}_1 \rhd \cdots \rhd \mathcal{A}_n \rhd \mathcal{A}_{n+1})$ and defined as follows. Let $n \geqslant 0$ be some integer and let us consider a collection $\mathcal{A}_1, \ldots, \mathcal{A}_n$ of *deterministic* pushdown automata (if $n = 0$ this collection is considered to be empty). Let $\mathcal{A}_{n+1}$ be a *deterministic* pushdown automaton equipped with a parity acceptance condition. On input alphabet of $\mathcal{A}_1, \ldots, \mathcal{A}_{n+1}$, we require

the following *stack consistency* property: for all $1 \leqslant i \leqslant n$, the input alphabet of $\mathcal{A}_{i+1}$ is the stack alphabet of $\mathcal{A}_i$. Let A be the input alphabet of $\mathcal{A}_1$. We associate with $\mathcal{A}_1, \ldots, \mathcal{A}_n, \mathcal{A}_{n+1}$ a language of infinite words on the alphabet A, that we denote $L(\mathcal{A}_1 \rhd \cdots \rhd \mathcal{A}_n \rhd \mathcal{A}_{n+1})$, and which is defined as follows (see Figure 27 for an illustration when $n = 3$).

- If $n = 0$, $L(\mathcal{A}_1 \rhd \cdots \rhd \mathcal{A}_n \rhd \mathcal{A}_{n+1}) = L(\mathcal{A}_{n+1})$ is the language accepted by $\mathcal{A}_{n+1}$.
- If $n > 0$, $L(\mathcal{A}_1 \rhd \cdots \rhd \mathcal{A}_n \rhd \mathcal{A}_{n+1})$ is the set of infinite words u_0 on the alphabet A such that:
  - When $\mathcal{A}_1$ reads u_0, its stack is strictly unbounded and therefore the sequence of stack contents converges to some limit u_1.
  - $u_1 \in L(\mathcal{A}_2 \rhd \cdots \rhd \mathcal{A}_n \rhd \mathcal{A}_{n+1})$.

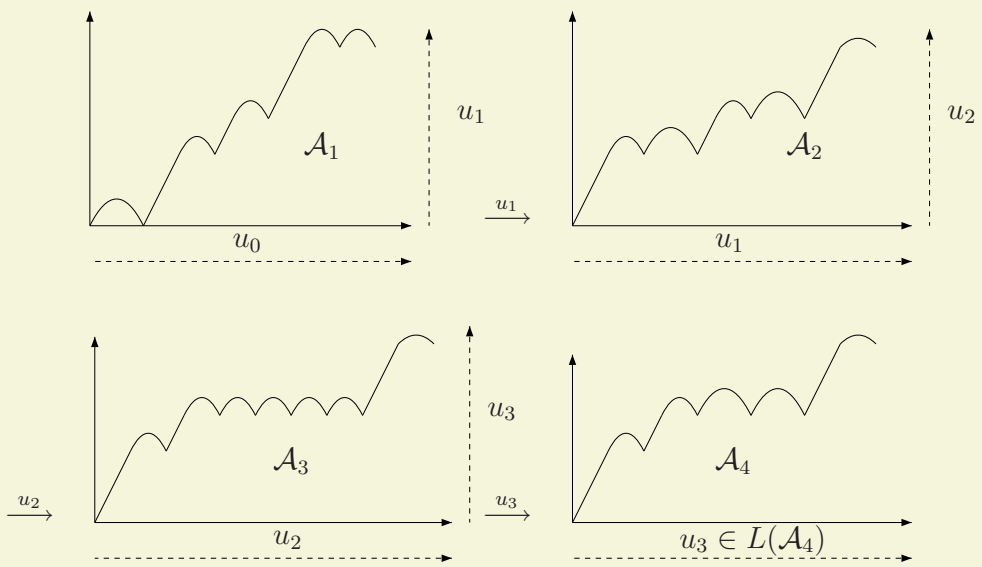


Figure 27 – Definition of the language $L(\mathcal{A}_1 \rhd \mathcal{A}_2 \rhd \mathcal{A}_3 \rhd \mathcal{A}_4)$

Equivalently, a word $u_0 \in A^\omega$ belongs to $L(\mathcal{A}_1 \rhd \cdots \rhd \mathcal{A}_n \rhd \mathcal{A}_{n+1})$ if and only if:

- For all $1 \leqslant i \leqslant n$, when $\mathcal{A}_i$ reads u_{i-1}, its stack is strictly unbounded and the sequence of stack contents converges to some limit u_i.
- $\mathcal{A}_{n+1}$ accepts u_n.

In [Ser04b; Ser06a] we proved that the languages of the form $L(\mathcal{A}_1 \rhd \cdots \rhd \mathcal{A}_n \rhd \mathcal{A}_{n+1})$ permit to obtain sets that are Π^0_{n+2}-complete in the Borel hierarchy (defined in the usual way on infinite words) and moreover if we use those languages to define a winning condition in a game played on a finite graph[17], then one can decide the winner (and it is an $(n+1)$-ExpTime complete problem). This result is interesting because it permits to exhibit a family of *decidable* winning conditions of arbitrarily "complicated" topological complexity.

[17] *i.e.* we map the vertices to letters in A and therefore we associate with a play a word in A^ω: the play is won by Éloïse if and only if its image belongs to $L(\mathcal{A}_1 \rhd \cdots \rhd \mathcal{A}_n \rhd \mathcal{A}_{n+1})$.

A natural question was whether the languages $L(\mathcal{A}_1 \rhd \cdots \rhd \mathcal{A}_n \rhd \mathcal{A}_{n+1})$ could be defined in a single pass fashion (as the original definition is mainly sequential: one first runs $\mathcal{A}_1$ and then $\mathcal{A}_2$ on the limit stack of the previous computation and so on…). The following theorem answers this question positively.

Theorem 28 [Serre, unpublished]

For all $n \geqslant 0$ and all language of the form $L(\mathcal{A}_1 \rhd \cdots \rhd \mathcal{A}_n \rhd \mathcal{A}_{n+1})$, there is an order-$n$ deterministic HOPDA $\mathcal{B}$ equipped with the strict unboundedness condition such that the set of words accepted by $\mathcal{B}$ is equal to $L(\mathcal{A}_1 \rhd \cdots \rhd \mathcal{A}_n \rhd \mathcal{A}_{n+1})$.

Moreover, as one can use Theorem 27 for the CPDA $\mathcal{B}$, one can easily deduce the decidability result presented in [Ser04b; Ser06a].

"*You're* the living example of the person who *never* think for himself!" "Am I really?" he said, laughing. "Yes! You're the most conformist man I ever met! All you do is what's expec-expec-expected of you" "That's terrible too?" "It's not *thinking*, D-d-dad! It isn't! It's being a s-s-stupid aut-aut-aut-aut-aut-automaton! A r-r-r-r-robot!"

Philip Roth, *American Pastoral*.

Tree Automata and Imperfect Information Games

3

1 Overview and Historical Background

This chapter considers, in an automata theoretic approach, languages of infinite node-labelled binary trees. An important motivation in computer science for studying infinite trees is that they provide a mathematical model for discrete transition systems that may be *branching*. Here, branching refers to the fact that some states may have several distinct direct successors (this can be due *e.g.* to nondeterminism in the system, to behaviours of an environment the systems interacts with or to nature for randomised systems): a path in such a tree represents a sequence of transitions of the system and the label of a node in this path specifies the properties of the system at that point. Therefore, a set of infinite trees describes a class of systems and, conversely, if one is interested in describing a class of systems in terms of their allowed behaviours it can be done by a set of infinite trees.

1.1 Finite Automata on Infinite Trees, Logic and Games

There are several natural ways of describing sets of infinite trees. One is *logic* as explained in Section 5 of Chapter 1 : with a logical formula is associated the set of all structures (say restricted to infinite trees) for which the formula holds. Then, there are many possible choices for the logics: let us mention again the monadic second order logic and the μ-calculus as well as more verification-oriented logics such as *Computation Tree Logic* (CTL) [CE81] and its extension CTL* [EH86] (for an excellent reference on this topic, see [BK08], in particular Chapter 6).

Another way to describe sets of infinite trees is using *finite automata*. Finite automata on infinite trees (that extend both automata on infinite words and automata on finite trees) were originally introduced by Rabin in [Rab69] to prove the decidability of the monadic second order logic over the full binary tree. Indeed, Rabin proved that for any MSO formula, one can construct a tree automaton such that it accepts a non empty language if and only if the original formula holds at the root of the full binary tree. These automata were also successfully used by Rabin in [Rab72] to solve Church's synthesis problem [Chu62], that asks for constructing a circuit based on a formal specification (typically expressed in monadic second order logic) describing the desired input/output behaviour. His approach was to represent the set of all possible behaviours of a circuit by an infinite tree (directions are used to code the inputs while node labels along a branch code the outputs) and to reduce the synthesis problem to the emptiness of a tree automaton accepting all those trees encoding circuits satisfying the specification.

Since then, automata on infinite trees and their variants have been intensively studied and found many applications, in particular in logic. Connections between automata on infinite trees and logic

are discussed *e.g.* in the excellent surveys [Tho97; VW07].

Roughly speaking a finite automaton on infinite trees is a finite memory machine that takes as input an infinite node-labelled binary tree and processes it in a top-down fashion as follows. It starts at the root of the tree in its initial state, and picks (possibly nondeterministically) two successor states, one per son, according to the current control state, the letter at the current node and the transition relation. Then the computation proceeds in parallel from both sons, and so on. Hence, a run of the automaton on an input tree is a labelling of this tree by control states of the automaton, that should satisfy the local constraints imposed by the transition relation. A branch in a run is accepting if the ω-word obtained by reading the states along the branch satisfies some acceptance condition (typically an ω-regular condition such as a Büchi or a parity condition). Finally, a tree is accepted by the automaton if *there exists* a run over this tree in which *every* branch is accepting. An ω-regular tree language is a tree language accepted by some tree automaton equipped with a parity condition.

A fundamental result of Rabin is that ω-regular tree languages form a Boolean algebra [Rab69]. The hard part in his proof is complementation, and since the publication of this result in 1969, it has been a challenging problem to simplify the proof. A much simpler one was obtained by Gurevich and Harrington in [GH82] making use of two-player perfect information games for checking membership of a tree in the language accepted by the automaton (note that the idea of using games to prove this result was already proposed by Büchi in [Büc77]): Éloïse (a.k.a. *Automaton*) builds a run on the input tree while Abélard (a.k.a. *Pathfinder*) tries to exhibit a rejecting branch in the run. The proof of Gurevich and Harrington was followed by many others trying to simplify the original proof of Rabin [Muc85; YY93; Zei94; MS95; EJ91; Mos91; Kla94]. See the introduction of [Zie98] for a remarkably precise comparison between those works.

Another fruitful connection between automata and games is for the emptiness checking. In a nutshell the emptiness problem for an automaton on infinite trees can be modelled as a game where Éloïse builds an input tree together with a run while Abélard tries to exhibit a rejecting branch in the run. Hence the emptiness problem for tree automata can be reduced to solving a *finite* two-player game of perfect information. Beyond these results, the tight connection between automata and games is one of the main tools in automata theory (see *e.g.* [Tho97; GTW02; Löd11]). Note that the connection between logic and games has also be intensively studied (see *e.g.* [Wil01; GTW02; Wal04]).

The focus in this chapter is on automata and games rather than on logic. Indeed, our initial approach is to modify the way we associate a language with a tree automaton rather than on designing variants of existing logics. The latter could be a further step but it does not seem to be easily doable in a clean and convincing way. Hence, rather than looking for an equivalent logical framework we design games that permit to solve natural problems as *e.g.* emptiness checking.

1.2 Three Levers to Define New Classes of Tree Automata

Recall that in the usual setting of non-deterministic tree automata, a tree is accepted if and only if there exists a run on which every branch is accepting. Hence, there are three natural levers on which one can act to define alternative families of tree automata / tree languages.

- The first lever is *local* with respect to the run: it is the condition required for a branch to be accepting. The reasonable options here being all classical ω-regular conditions: reachability (the branch contains some final state), Büchi (the branch contains infinitely many final states) or parity (the smallest colour appearing infinitely often in the branch is even). For the parity condition one can define the so-called Mostowski hierarchies [Mos85] by looking at those

tree languages accepted when one uses only colours in the interval $[i, j]$ (such a language is said to be i, j-*feasible*; one easily sees that we can always assume that $i = 0$ or $i = 1$; Büchi condition corresponds to the case where $i = 0$ and $j = 1$ while co-Büchi condition is the case where $i = 1$ and $j = 2$). The Mostowski hierarchy is strict for all standard models of tree automata: deterministic [Wag77] (the hierarchy is strict already for infinite words when one consider deterministic automata), non-deterministic [Niw86] and alternating [Arn99] (combined with [Bra98; Len96]). Hence, allowing more colours gives more power to the automata regardless on their nature (deterministic, non-deterministic or alternating).

- The second lever is *global* with respect to the run: it is the condition required for a run to be accepting. The usual definition is that *all* branches must be accepting for the run to be accepting. In this chapter we relax this condition by specifying (in different natural ways) *how many* branches should be accepting.

  One can do this either by *counting* the number of accepting branches (*e.g.* infinitely many, uncountably many) or by counting the number of rejecting branches(*e.g.* finitely many, at most countably many) : this leads to the notion of automata with cardinality constraints. The idea of allowing a certain amount of rejecting branches in a run was already considered by Beauquier, Nivat and Niwiński in [BNN91; BN95], where it was required that the number of accepting branches in a run belongs to a specified set of cardinals Γ. In particular, they proved that if Γ consists of all cardinals greater than some γ, then one obtains a regular tree language. The approach used in [BNN91; BN95] is based on logic (actually they proved that a tree language defined by such an automaton can be defined by a Σ^1_1 formula hence, can also be defined by a *Büchi* tree automaton) while the one we developed with Arnaud Carayol and Axel Haddad in [CHS14a] is based on designing acceptance games. Note that we also considered the case where one requires that there are at most finitely many (resp. countably many) rejecting branches in a run to be accepting. Also note that these properties can be expressed in MSO logic [BKR10] implying directly that the classes of languages accepted under these various restrictions are ω-regular. However this logical approach does not give a tractable transformation to standard parity or Büchi automata.

  Another option is to use a notion of *topological* "bigness" and to require for a run to be accepting that the set of accepting branches is topologically large. Topological largeness was previously considered in the literature (*e.g.* with motivations from program verification in order to define fairness properties of transition systems [VV12]) with a special focus on its tight connection with so-called Banach-Mazur games (see *e.g.* [Grä08] for a survey on that topic). Together with Arnaud Carayol and Axel Haddad we have shown in [CHS14a] that the tree languages accepted with this new condition are indeed regular. As for the case where one counts accepting/rejecting branches, the result is obtained by designing an acceptance game and later argue that it can be seen as another equivalent acceptance game for the usual semantics on runs.

  A last option is to *measure* (in the usual sense of measure theory *à la* Lebesgue) the set of accepting branches and to put a constraint on this measure (*e.g.* positive, at least one half, equal to one). With the parity condition and allowing a negligible set of non-accepting branches for a run to be accepting, this leads to a new class of tree languages that we called *qualitative tree languages* [CHS11; CHS14b]. We showed that this class enjoys many desirable properties: closure under union and intersection (but not under complement), the emptiness problem is decidable in polynomial time (contrasting with the fact that no polynomial algorithm is known for the emptiness test of standard parity tree automata). We also proved that there exists a strong connection between automata accepting qualitative tree languages and Markov

decision processes, which play here a similar role as two-player games for usual tree automata [GH82].

- The third lever has to do with the set of runs. The usual definition is existential: a tree is accepted if there exists an accepting run on that tree. A dual notion is universality: a tree is accepted if all runs of the automaton on that tree are accepting.

  A stronger way of dealing with the set of runs (and generalising both existential and universal automata) is obtained using alternation [CKS81]: here a run is built as an interaction between an existential and a universal player and a tree is accepted if and only if the existential player has a strategy so that the resulting run is accepting whatever is the strategy of the universal player. Alternation makes complementation a simple task as it simply consists in switching the two players and dualising the acceptance condition. In particular, combining alternation with the parity condition yields the automata-theoretic counterpart of the propositional μ-calculus, where the translation from one to the other can be done in linear time [Wil01; AN01; GTW02]. Hence, the model-checking and the satisfiability/validity of logical formulas amount to respectively verifying membership and non-emptiness/universality on their corresponding tree automata. The membership problem for alternating tree automata has a fairly simple algorithm: one compiles the input tree and the automaton into a polynomial size perfect information parity game and solves it. On the contrary, the usual roadmap to check the emptiness of an alternating tree automaton is more involved. First one builds an equivalent non-deterministic automaton thanks to the Simulation Theorem [MS95], and then one checks the emptiness of this latter automaton by solving an associated two-player perfect information game. This yields an exponential time algorithm, which is optimal as the emptiness problem is ExpTime-complete. Together with Nathanaël Fijalkow and Sophie Pinchinat we proposed in [FPS13] an alternative technique to solve the emptiness problem of alternating tree automata, by directly reducing it to a two-player game with *imperfect information*[1]. This builds on a long tradition initiated by Reif in [Rei79], that advocates the use of games with imperfect information to solve algorithmic problems for automata; in his seminal paper, Reif introduced the notion of *blindfold games* and used them to check the universality of non-deterministic automata over finite words. This approach has later been extended in [DR10; WDHR06] and combined with antichains representations, to check universality and inclusion of non-deterministic word automata, as well as the emptiness for alternating word automata. This was backed with solid experimental results (see *e.g.* the tool Alaska [WDMR08]), where the emptiness of alternating Büchi word automata was considered, building on the Miyano-Hayashi construction [MH84]. To the best of our knowledge, antichains approaches have not yet been extended to alternating parity tree automata. However, solving the emptiness problem for alternating parity tree automata through games of imperfect information has been considered by Puchala in his PhD [Puc13], where he has provided a reduction of the emptiness problem for alternating parity automata to solving a *three-player* game with imperfect information, but no algorithm to solve the latter. Our technique is of interest for at least two reasons: (i) it pushes the algorithmic difficulty to the game solving part, for which antichains representations have recently been developed [BCWDHR09], hence could lead to efficient algorithms, and (ii) a "Simulation Theorem"-free technique is required for classes of tree automata for which no (effective) Simulation Theorem exists.

[1] This does not lead to a gain in complexity due to intrinsic hardness, but unravels the two key ingredients: the first one is the positional determinacy of parity games, to prove the correctness of the reduction, and the second is the determinisation property of ω-word automata, to solve the obtained two-player imperfect information game. Note that, for comparison with the classical approach for alternating parity tree automata, the Simulation Theorem [MS95] also combines the above two key ingredients.

We illustrated this latter situation by considering an alternation extension of the class of qualitative tree automata and we applied our technique to check the emptiness of an alternating qualitative Büchi tree automaton. Furthermore, we observed that the emptiness problem becomes undecidable for the co-Büchi condition, implying that there is no simulation theorem for alternating qualitative tree co-Büchi automata. For our technique to go through, the key ingredient is a positionality result for stochastic Büchi games over *infinite* arenas[2].

Another natural variant is to *measure* the set of accepting runs. This led the concept of probabilistic automata: such an automaton associates with an input a measurable set of runs and acceptance is defined with respect to the measure (*e.g.* positive, at least one third, equal to one) of the set of accepting runs. On finite words probabilistic automata have been introduced by Rabin in [Rab63]. Compared with the standard setting, the non-deterministic guesses are replaced by random choices (according to some probabilistic distribution depending on the control state and the input letter). Hence, the set of transitions is replaced by a probability distribution over the set of all transitions which induces a probability measure on the set of runs of the automaton and acceptation is defined using a threshold $0 < \lambda < 1$ on the probability of a run to be accepting. In contrast to the non-deterministic setting, the emptiness problem for probabilistic automata on finite words is undecidable [Paz71]. The probabilistic model was recently extended to infinite words in [BG05] by Baier and Größer[3] and studied in more details in [BBG08; CSV09; CDH09]. In addition to the threshold criterion, two additional semantics were considered : *almost-sure* and *probable* which respectively corresponds to a probability 1 or > 0 for a run to be accepting[4]. Surprisingly the class of languages defined by Büchi automata with the probable semantics is closed under complement, which implies that it coincides with the class of languages defined by co-Büchi automata with the almost-sure semantics.The emptiness problem for Büchi automata with the almost-sure semantics as well as for co-Büchi automata with the probable semantics are decidable. However the emptiness problem for Büchi automata with the probable semantics as well as for co-Büchi automata with the almost-sure semantics are undecidable. Of course, the emptiness is undecidable when considering a threshold semantics. We refer the reader to [BGB12] for a very rich overview of this topic. In [CHS11; CHS14b], we considered probabilistic automata on infinite trees. We focused on the *almost-sure* semantics, *i.e.* a tree is accepted if almost every run over it is accepting, and on the *probable* semantics, *i.e.* a tree is accepted if the set of accepting runs on it has a (strictly) positive measure. Of course each semantics can be used in combination with the acceptance criteria on runs: the classical one (all branches are accepting), the qualitative one (almost all branches are accepting) and the positive one (there is a non negligible set of accepting branches). For all these combinations, we established that the definition makes sense (*i.e.* we proved measurability of the set of accepting runs). For the qualitative criterion on runs combined with the almost-sure semantics, as well as for the probable criterion on runs combined with the positive semantics, we proved that there exists a strong connection with *partial observation* Markov decision processes. This condition is independent of the acceptance condition on branches (Büchi, co-Büchi, parity…). In particular, for the Büchi (*resp.* co-Büchi) acceptance condition on branches, probabilistic automata on infinite trees with the qualitative criterion on runs combined with the almost-sure semantics (*resp.* with the

[2] To the best of our knowledge, very few positionality results are known in the literature that combine both stochastic aspects and infiniteness of the game arena; notable exceptions are [BKN13; Kuc11].

[3] A previous attempt by Reisz[Rei99] should be mentioned but this approach does not make real use of probabilities as in this setting an input word is accepted if after some time the behaviour in the run becomes deterministic.

[4] In the *finite* words case, the almost-sure and probable acceptance are trivial as the set of runs for a given word is finite.

positive criterion on runs combined with the probable semantics) enjoy a decidable emptiness problem. To our knowledge, this was the first positive result for a class of probabilistic automata over infinite trees. On the negative side, we derived from undecidability results on probabilistic automata on infinite words similar results for probabilistic automata on infinite trees. Note that whilst being immediate for the classical criterion on runs such a reduction is not as simple for the qualitative and positive criteria.

1.3 Imperfect Information Games

As already mentioned in Section 1.1, there is a tight connection between non-deterministic tree automata and two-player perfect information games. In particular:

(i) the acceptance problem (*i.e.* whether some given tree is accepted by some given tree automaton) can be reduced to deciding the winner in a two-player game;

(ii) the emptiness problem (*i.e.* whether the set of trees accepted by a given tree automaton is empty) can be reduced to deciding the winner in a *finite* two-player game.

In a nutshell, in the first game Éloïse builds a run on the input tree while Abélard tries to exhibit a rejecting branch in the run. A natural way to obtain the second game from the first one is to let Éloïse handle the existential quantification (*i.e.* "there exists a tree that is accepted by the automaton"): she is not only in charge of choosing the run but also of providing the input tree. Éloïse has no way of cheating by choosing the input on the fly because Abélard is not involved in choosing the run but only in selecting a branch (and as a winning strategy of Éloïse must be winning against *any* strategy of Abélard it implies that any branch must be accepting).

Imperfect information naturally arises when considering universal automata (for which an input is accepted if *all* runs on this input are accepting). A natural game consists in asking Éloïse to provide a tree and then to ask Abélard to provide both a run and a rejecting branch in that run. But as the first step requires an infinite size description it is not satisfying. A temptation would consist in letting Éloïse describe the tree while Abélard builds on the fly the run and selects a branch: but the drawback is that Éloïse, observing the run Abélard is constructing, could cheat and adapt the input accordingly. Hence, the solution is to *hide* information to Éloïse, namely the one on the run.

In this chapter we use several classes of imperfect-information (possibly stochastic) games to design the emptiness games for the variants of tree automata that we consider. For simplicity in the study of decision problems related to these game we do this in a general framework that we introduced with Vincent Gripon in [GS09]. Note that Bertrand, Genest and Gimbert independently considered in [BGG09] an equivalent[5] formalism (stochastic games with signals which are widely studied in classical game theory; see *e.g.* [AH92; OR94]). We prefer the formalism of [GS09] because it is closer to the one of previously defined weaker classes of imperfect information games considered in our field [CDHR07; BCDHR08; BD08; BCWDHR09; CDGH10].

Imperfect-information stochastic games are finite state games in which, at each round, the two players choose concurrently an action and based on these actions the successor state is chosen according to some fixed probability distribution. The resulting infinite play is won by Éloïse if it satisfies a given *objective* (*e.g.* reachability, safety, Büchi or co-Büchi). Imperfect information is modelled as follows: both players have an equivalence relation over states and, instead of observing the exact state, they only see to which equivalence class it belongs to. Therefore, if two partial plays are indis-

[5]Note that the translations from the model in [GS09] to the model in [BGG09] and *vice versa* are straightforward (and do not lead to any blow-up in the size).

tinguishable by some player (*i.e.* they give rise to the same sequence of equivalence classes for this player), he should behave the same in both of them.

1.3.1 The Randomised Strategies Setting

In [GS09; BGG09] the authors were considering general strategies where a player is allowed to use *randomisation* when choosing her/his next action. Of special interest are almost-surely (*resp.* positively) winning strategies, *i.e.* strategies that against any counter-strategy of the adversary wins with probability 1 (*resp.* positive probability): indeed, due to concurrency and imperfect information, sure winning is often unlikely to happen. It was then shown [GS09; BGG09], for Büchi objectives, that one can decide whether Éloïse has such a strategy φ that is almost-surely winning against any strategy ψ of Abélard (meaning that an infinite play obtained from φ and ψ is won by Éloïse with probability 1). It was also established in [BGG09] that one can decide for co-Büchi objectives whether Éloïse has a positively winning strategy. This is in fact a consequence of a determinacy-like result also stated in [BGG09]: in a Büchi game, either Éloïse has an almost-surely winning strategy or Abélard has a positively winning strategy. Note that the technique in [GS09; BGG09] always provide a *randomised* strategy: in particular if an optimal pure (*i.e.* not randomised) strategy exists, it is not the one that is computed.

1.3.2 The Pure Strategies Setting

Motivated by the quest for simple winning strategies (in particular ones that do not use randomisation) and by applications to the theory of automata on infinite trees we then restricted our attention to *pure* strategies, *i.e.* we forbid the players to randomise when choosing their actions. Later in the chapter we show how our decidability results can be used to establish the decidability of the emptiness problem for several classes of alternating or probabilistic tree automata. Our main positive results were obtained in the special case where Abélard is more informed than Éloïse. On the negative side, by a reduction of the value 1 problem for probabilistic word automata [Col09; GO10], we proved that even if Abélard is fully informed and Éloïse is totally blind (*i.e.* all states are indistinguishable for her), it is undecidable whether Éloïse can positively win a safety game. Under the same restrictions, positive winning in Büchi games and almost-sure winning in co-Büchi games are known to be undecidable by a reduction from the emptiness problem for classes of probabilistic ω-word automata [BGB12]. Note that the undecidability result for safety objective strongly contrasts with the decidability result in the setting of [BGG09] where one allows randomised strategies (actually, when one restricts to pure strategies, the qualitative determinacy result of [BGG09] no longer holds). On the positive side, we can decide whether Éloïse has a positively winning pure strategy in a reachability game. Then, using this result as a black box in a fixpoint computation, we proved that one can decide whether Éloïse has an almost-surely winning pure strategy in a Büchi game. Moreover, if exists, such a strategy can be constructed and requires *finite* memory. In both cases, we obtained algorithms in ExpTime if Abélard has perfect information and in 2-ExpTime if he is more informed than Éloïse.

These results, done in collaboration with Arnaud Carayol and Christof Löding, overlap with the ones *independently* obtained by Chatterjee and Doyen in [CD12]. A major difference is that the results obtained in [CD12] only concern the case where Abélard is *fully informed* while our results only assume that he is more informed than Éloïse. On the decidability side and when Abélard is fully informed, the results obtained in [CD12] are similar to the ones presented here but the proofs are quite different. On the undecidability side, the only result mentioned in [CD12] is the one that directly follows from [BGB12] (positive Büchi); in particular [CD12] does not contain the undecidability result for positive safety.

1.3.3 Related Work in Classical Game Theory

Let us stress that zero-sum finite-state stochastic games are intensively studied in classical game theory since the seminal paper of Shapley [Sha53]. The games we consider here differ from Shapley's on the following points: (i) the games are not required to be stopping; (ii) the players have imperfect information; (iii) the players are (sometimes) assumed to be using pure strategies. While the last difference (playing pure) is quite restrictive from a game-theoretic point of view, all other specificities have already been considered (and combined). This led to consider various games, among which *recursive games* [Eve57], *repeated games* with perfect information (*aka supergames*) [Rub79] or with incomplete information [AM95], *repeated games with an informed controller* [Ren12] and stochastic games with signals. See [AH92] for a reference textbook on the topic.

Also note that classical game theory deeply differs from the studies in computer science. Indeed, while in computer science the focus is on algorithmic aspects with a special interest in decidability, in mathematics the focus is on finding general classes of games for which key concepts and results still hold (*e.g.* existence of equilibria, folk Theorem, existence of uniform max-min value…). In particular, the payoff functions considered in computer science (specially when considering powerful classes of games like the one with imperfect information) are quite simple compared with the one considered in classical game theory, but this is the price to pay to preserve decidability. Hence, we believe that the present work, even if considering concepts widely studied in a neighbour community is quite orthogonal to these studies.

1.4 Structure of this Chapter

The chapter is structured as follows. We start in Section 2 by the main classical definitions on tree automata and regular tree languages. Section 3 studies tree languages accepted by automata with cardinality constraints and mainly focus on designing acceptance games for them that are later used to prove that those languages are always regular. Section 4 studies tree languages accepted by automata with topological bigness constraints. Section 5 is devoted to qualitative tree languages and to the study of their properties and decision problems, as well as to compare them with regular tree languages. In Section 6, we introduce imperfect information games and study the related decision problems both in the setting of randomised strategies and pure strategies. Section 7 revisits the emptiness problem for usual alternating automata using imperfect information games and explains how this technique can also be used for alternating qualitative tree automata. Section 8 we consider probabilistic automata on infinite trees. Finally, Section 9 present several natural problems as well as perspectives related to the topics discussed in the present chapter.

2 Basic Objects, Tree Automata and Regular Tree Languages

2.1 Trees and Branches

In all this chapter when dealing with trees, we only consider full binary node-labelled trees. An ***A-labelled tree*** t is a mapping from $\{0,1\}^*$ to A. In this context, an element $u \in \{0,1\}^*$ is called a ***node***, and the node $u0$ (*resp.* $u1$) is the ***left child*** (*resp.* ***right child***) of u. The node ε is called the ***root***. We shall refer to $|u|$ as the ***depth*** of u. The letter $t(u)$ is called the ***label*** of u in t.

A ***branch*** is an infinite word $\pi \in \{0,1\}^\omega$. We write $\mathrm{Br} = \{0,1\}^\omega$ for the set of all branches. A node u belongs to a branch π if u is a prefix of π. For an A-labelled tree t and a branch $\pi = \pi_0\pi_1\cdots$ we define the ***label*** of π as the ω-word $t(\pi) = t(\varepsilon)t(\pi_0)t(\pi_0\pi_1)t(\pi_0\pi_1\pi_2)\cdots$.

Given a tree t and a node u, the ***subtree of t rooted at u*** denoted t_u is the tree defined by

$t_u(v) = t(u \cdot v)$. A tree t is said to be ***regular*** if it contains only finitely many different subtrees, *i.e.* the set $\{t_u \mid u \in \{0,1\}^*\}$ is finite.

2.2 Non-Deterministic Tree Automata and Regular Tree Languages

A ***tree automaton*** $\mathcal{A}$ is a tuple $\langle A, Q, q_{\text{ini}}, \Delta, \text{Acc}\rangle$ where A is the ***input alphabet***, Q is the finite ***set of states***, $q_{\text{ini}} \in Q$ is the ***initial state***, $\Delta \subseteq Q \times A \times Q \times Q$ is the ***transition relation*** and $\text{Acc} \subseteq Q^\omega$ is the ***acceptance condition***. In the following, we use the notation $q \xrightarrow{a} (q_0, q_1)$ as a shorthand for $(q, a, q_0, q_1) \in \Delta$. An automaton is ***deterministic*** if $q \xrightarrow{a} (q_0, q_1)$ and $q \xrightarrow{a} (q'_0, q'_1)$ implies $q_0 = q'_0$ and $q'_1 = q'_1$. An automaton is ***complete*** if, for all $q \in Q$ and $a \in A$ there is at least one pair $(q_0, q_1) \in Q^2$ such that $q \xrightarrow{a} (q_0, q_1)$.

Given an A-labelled tree t, a ***run*** of $\mathcal{A}$ over t is a Q-labelled tree ρ such that

- the root is labelled by the initial state, *i.e.* $\rho(\varepsilon) = q_{\text{ini}}$;
- for all nodes u, $(\rho(u), t(u), \rho(u0), \rho(u1)) \in \Delta$.

A branch $\pi \in \{0,1\}^\omega$ is ***accepting*** in the run ρ if $\rho(\pi) \in \text{Acc}$. A run ρ is ***accepting*** if *all* its branches are accepting. Finally, a tree t is ***accepted*** if *there exists* an accepting run of $\mathcal{A}$ over t. The set of all trees accepted by $\mathcal{A}$ is denoted $L(\mathcal{A})$.

We consider the following classical acceptance conditions for branches:

- A ***reachability*** acceptance condition is given by a subset $F \subseteq Q$ of final states by letting $\text{Acc} = Reach(F) = Q^*FQ^\omega$, *i.e.* a branch is accepting if it contains a final state.
- A ***safety*** acceptance condition is given by a subset $F \subseteq Q$ of forbidden states by letting $\text{Acc} = Saftey(F) = (Q \backslash F)^\omega$, *i.e.* a branch is accepting if its labelling never contains a forbidden state.
- A ***Büchi*** acceptance condition is given by a subset $F \subseteq Q$ of final states by letting $\text{Acc} = Buchi(F) = (Q^*F)^\omega$, *i.e.* a branch is accepting if it contains infinitely many final states.
- A ***co-Büchi*** acceptance condition is given by a subset $F \subseteq Q$ of forbidden states by letting $\text{Acc} = coBuchi(F) = Q^*(Q \backslash F)^\omega$, *i.e.* a branch is accepting it contains finitely many forbidden states.
- A ***parity*** acceptance condition is given by a colouring mapping $\text{Col} : Q \to \mathbb{N}$ by letting $\text{Acc} = Parity = \{q_0q_1q_2 \cdots \mid \liminf(\text{Col}(q_i))_i \text{ is even}\}$, *i.e.* a branch is accepting if the smallest colour appearing infinitely often is even.

All these conditions are examples of ω-regular acceptance conditions, *i.e.* Acc is a regular set of ω-words [PP04].

***Remark* 26.** *Büchi winning conditions are those parity conditions using only colours* 0 *and* 1 *(final vertices are those coloured by* 0*) while co-Büchi winning conditions are those parity conditions using only colours* 1 *and* 2 *(forbidden vertices are those coloured by* 1*).*

***Remark* 27.** *The parity condition is expressive enough to capture the general case of an arbitrary ω-regular condition. Indeed it is well known that* Acc *is accepted by a* deterministic *parity word automaton. By taking the synchronised product of this automaton with the tree automaton, we obtain a parity tree automaton accepting the same language (see* e.g. *[PP04]).*

When it is clear from the context, we may replace, in the description of $\mathcal{A}$, Acc by F (for a reachability, safety, Büchi or co-Büchi condition) or Col (for a parity condition), and we shall refer to the automaton as a reachability (*resp.* safety, Büchi, co-Büchi, parity) tree automaton. A set L of trees is a ***regular*** language if there exists a parity tree automaton $\mathcal{A}$ such that $L = L(\mathcal{A})$.

Example 26

Let L be the set of $\{a, b\}$-labeled trees that contain an infinite branch with infinitely many nodes labeled by b. Then L is easily seen to be a regular tree language. Indeed, a non-deterministic Büchi automaton accepting L works as follows. It has three states q_f, q_a and q_b, and both q_f and q_b are final states. When being in states q_f it stays in it forever (*i.e.*, regardless on the node label it goes in both direction to q_f). In state q_x, for $x \in \{a, b\}$ on reading a node labeled by y, it goes to either to q_f on the left and q_y on the right, or to q_y on the left and q_f on the right. Hence, the automaton guesses a branch (all other branches are trivially accepting thanks to the absorbing state q_f) along which it goes to a final state only after being in an b-labeled node. Hence, the guessed branch is accepting if and only if it contains infinitely many b-labeled nodes.

The class of regular tree languages is robust, as illustrated by the following fundamental result.

Theorem 29 [Rab69]

The class of regular tree languages is a Boolean algebra.

***Remark* 28.** *Note that, contrarily to what happens for words, non-deterministic tree automata are strictly more expressive than deterministic ones (regardless of the acceptance condition). Indeed, a typical language of regular tree language that cannot be accepted by any deterministic automaton is the set of all trees containing at least one node labeled by a distinguished letter a.*

Also note that allowing more colours in the parity condition increases the expressive power of tree automata. This can easily be deduced from a similar result on infinite words (see e.g. *[PP04]).*

2.3 Decision Problems, Acceptance Game and Emptiness Game

The ***emptiness problem*** is to decide for a given automaton $\mathcal{A}$ whether one has $L(\mathcal{A}) = \varnothing$. This problem is well-known to be polynomially equivalent to the problem of deciding the winner in a two-player parity game on a finite graph (and those colours used in the automaton and in the game are identical). Indeed, the acceptance of a given tree by a given parity tree automaton can be rephrased in terms of games, and then starting from that game and forgetting about the input tree one can think of the emptiness problem as solving a (finite) game.

Fix an automaton $\mathcal{A} = \langle A, Q, q_{\text{ini}}, \Delta, \text{Acc}\rangle$ and a tree t and define an *acceptance game* $\mathbb{G}_{\mathcal{A},t}$ as follows. Intuitively, a play in $\mathbb{G}_{\mathcal{A},t}$ consists in moving a pebble along a branch of t in a top-down manner: the pebble is attached a state and in a node u with state q Éloïse picks a transition $(q, t(u), q_0, q_1) \in \Delta$, and then *Abélard* chooses to move down the pebble either to $u0$ (and update the state to q_0) or to $u1$ (and update the state to q_1).

Formally (see Figure 28 for an illustration), one lets $G_{\mathcal{A},t} = (V_E \uplus V_A, E)$ with $V_E = Q \times \{0, 1\}^*$ and $V_A = \{(q, u, q_0, q_1) \mid u \in \{0, 1\}^* \text{ and } (q, t(u), q_0, q_1) \in \Delta\} \subseteq Q \times \{0, 1\}^* \times Q \times Q$ and

$$\begin{aligned} E = \quad & \{((q, u), (q, u, q_0, q_1)) \mid (q, u, q_0, q_1) \in V_A)\} \quad \cup \\ & \{((q, u, q_0, q_1), (u \cdot x, q_x)) \mid x \in \{0, 1\} \text{ and } (q, u, q_0, q_1) \in V_A)\} \end{aligned}$$

Then let $\mathcal{G}_{\mathcal{A},t} = (G_{\mathcal{A},t}, V_E, V_A)$ and extend Col on $V_E \cup V_A$ by letting $\mathrm{Col}((q,u)) = \mathrm{Col}((q,u,q_0,q_1)) = \mathrm{Col}(q)$. Finally define $\mathbb{G}_{\mathcal{A},t}$ as the parity game $(\mathcal{G}_{\mathcal{A},t}, \mathrm{Col})$.

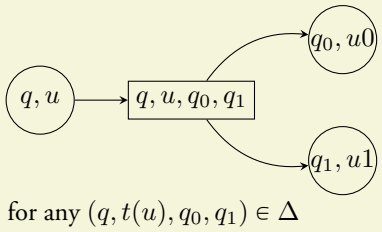


Figure 28 – Local structure of the (infinite) arena of the acceptance game $\mathbb{G}_{\mathcal{A},t}$.

The next proposition is a well-known result and its proof is obtained by noting that strategies for Éloïse in $\mathbb{G}_{\mathcal{A},t}$ are in bijection with runs of $\mathcal{A}$ on t.

Proposition 6

For any automaton $\mathcal{A}$ and any tree t one has $t \in L(\mathcal{A})$ if and only if Éloïse wins in $\mathbb{G}_{\mathcal{A},t}$ from $(q_{\text{ini}}, \varepsilon)$.

Now, in order to check the emptiness for a given parity tree automaton $\mathcal{A}$, the idea is to slightly modify the previous game to additionally require Éloïse to describe a tree t in $L(\mathcal{A})$. Therefore, instead of picking a transition she also has to provide a node label. To make the resulting game finite one "forgets" about the nodes, *i.e.* the component in $\{0,1\}^*$.

Formally (see Figure 29 for an illustration), one let $G_{\mathcal{A}} = (V'_E \uplus V'_A, E')$ with $V'_E = Q$ and $V'_A = \Delta$ and

$$\begin{aligned} E' = \quad & \{(q, (q,a,q_0,q_1)) \mid (q,a,q_0,q_1) \in \Delta)\} \quad \cup \\ & \{((q,a,q_0,q_1), q_x) \mid x \in \{0,1\} \text{ and } (q,a,q_0,q_1) \in \Delta)\} \end{aligned}$$

Then one lets $\mathcal{G}_{\mathcal{A}} = (G_{\mathcal{A}}, V'_E, V'_A)$ and extends Col on $V'_E \cup V'_A$ by letting $\mathrm{Col}((q,a,q_0,q_1)) = \mathrm{Col}(q)$. Finally, one defines $\mathbb{G}_{\mathcal{A}}$ as the parity game $(\mathcal{G}_{\mathcal{A}}, \mathrm{Col})$.

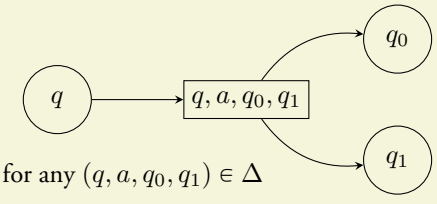


Figure 29 – Local structure of the (finite) arena of the acceptance game $\mathbb{G}_{\mathcal{A}}$.

The next proposition is a well-known result and its proof is obtained by noting that strategies for Éloïse in $\mathbb{G}_{\mathcal{A}}$ are in bijection with pairs made of a tree t and a run of $\mathcal{A}$ on t.

Proposition 7

For any automaton $\mathcal{A}$ one has $L(\mathcal{A}) \neq \varnothing$ if and only if Éloïse wins in $\mathbb{G}_{\mathcal{A}}$ from q_{ini}.

Hence, from Theorem 2 and Proposition 7, follows a decision procedure for the emptiness problem.

Corollary 10

The emptiness problem for parity tree automata is decidable and belongs to UP∩co-UP. If one bounds the number of colours (*e.g.* considering a Büchi or a co-Büchi condition) the problem can be solved in polynomial time.

3 Counting Branches in Trees: Automata with Cardinality Constraints

In the classical definition (cf Section 2.2 above), a run of a tree automaton $\mathcal{A}$ is accepting if and only if *all* its branches satisfy the acceptance condition. In collaboration with Arnaud Carayol and Axel Haddad we introduced in [CHS14a] several relaxed notions of acceptation where a run is accepting depending on how many accepting/rejecting branches it contains.

The idea of allowing a certain amount of rejecting branches in a run was first considered by Beauquier, Nivat and Niwiński in [BNN91; BN95], where it was required that the number of accepting branches in a run belongs to a specified set of cardinals Γ. In particular, they proved that if Γ consists of all cardinals greater than some γ, then one obtains a regular tree language. Their approach was based on logic (actually they proved that a tree language defined by such an automaton can be defined by a Σ^1_1 formula hence, can also be defined by a *Büchi* tree automaton) while the one we developped in [CHS14a] was based on designing acceptance games. Note that we also considered the case where one requires that there are at most finitely many (resp. countably many) rejecting branches in a run to be accepting. As the properties we considered can be expressed in MSO logic [BKR10], the classes of languages accepted under these various restrictions are ω-regular. However this logical approach does not give a tractable transformation to standard parity or Büchi automata.

For each of the various classes, we provided an acceptance game. Then we constructed a tree automaton with the classical semantics whose acceptance game is equivalent to this game. Hence, we could conclude that the languages accepted by automata with cardinality constraints are always regular tree languages. In the case where one counts rejecting branches we showed that in general (and contrarily to what happens when one counts accepting branches) the language that we obtain cannot be accepted by a *Büchi* tree automaton.

3.1 Automata with Cardinality Constraints

We now relax the criterion for a run to be accepting. For a given automaton $\mathcal{A}$, we define the following four criteria (two for the case where one counts the number of accepting branches and two for the case where one counts the number of rejecting branches) for a run to be accepting. Note that the case where one counts accepting branches was already considered in [BNN91; BN95].

- There are finitely many rejecting branches in the run. A tree $t \in L^{\text{Rej}}_{\text{Fin}}(\mathcal{A})$ if and only if there is a run of $\mathcal{A}$ on t satisfying the previous condition.
- There are at most countably many rejecting branches in the run. A tree $t \in L^{\text{Rej}}_{\leqslant\text{Count}}(\mathcal{A})$ if and only if there is a run of $\mathcal{A}$ on t satisfying the previous condition.

- There are infinitely many accepting branches in the run. A tree $t \in L^{\text{Acc}}_{\infty}(\mathcal{A})$ if and only if there is a run of $\mathcal{A}$ on t satisfying the previous condition.
- There are uncountably many accepting branches in the run. A tree $t \in L^{\text{Acc}}_{Uncount}(\mathcal{A})$ if and only if there is a run of $\mathcal{A}$ on t satisfying the previous condition.

***Remark* 29.** *Recall that the continuum hypothesis is true for any Borel sets [Ale16]. Therefore, as the set of accepted branches in a run of a tree automaton is Borel (see* e.g. *Proposition 9 below), it follows that the cardinality of a set of accepting branches in a run is either finite or equal to $\aleph_0$ or to $2^{\aleph_0}$. Hence, the four classes above are the only ones that makes sense.*

3.2 Counting Rejecting Branches

For the classes of automata where acceptance is defined by a constraint on the number of rejecting branches we show that the associated languages are regular. For this, we adopt the following roadmap: first we design an acceptance game and then we note that it can be transformed into another *equivalent* game that turns out to be the (usual) acceptance game (as presented in Section 2.3) for some tree automaton.

Fix, for this section, a parity tree automaton $\mathcal{A} = \langle A, Q, q_{ini}, \Delta, \text{Col}\rangle$ and recall that a tree t is in $L^{\text{Rej}}_{\leqslant \text{Count}}(\mathcal{A})$ (resp. in $L^{\text{Rej}}_{\text{Fin}}(\mathcal{A})$) if and only if there is a run of $\mathcal{A}$ on t in which there are at most countably (resp. finitely) many rejecting branches.

3.2.1 The Case of Languages $L^{\text{Rej}}_{\leqslant \text{Count}}(\mathcal{A})$

Fix a tree t and define an acceptance game for $L^{\text{Rej}}_{\leqslant \text{Count}}(\mathcal{A})$ as follows. In this game (we refer the reader to Figure 30 for the local structure of the arena for game $\mathbb{G}^{\text{Rej}\leqslant\text{Count}}_{\mathcal{A},t}$) the two players move a pebble along a branch of t in a top-down manner: the pebble is attached a state whose colour gives the colour of the configuration. Hence, (Éloïse's main) configurations in the game are elements of $Q \times \{0,1\}^*$. In a node u with state q Éloïse picks a transition $(q, t(u), q_0, q_1) \in \Delta$, and then Abélard has two possible options:

(i) he chooses a direction 0 or 1; or

(ii) he lets Éloïse choose a direction 0 or 1.

Once the direction $i \in \{0,1\}$ is chosen, the pebble is moved down to ui and the state is updated to q_i. A play is won by Éloïse if one of the following two situations occurs: either the parity condition is satisfied or Abélard did not let Éloïse infinitely often choose the direction. Call this game $\mathbb{G}^{\text{Rej}\leqslant\text{Count}}_{\mathcal{A},t}$.

The next theorem states that it is an acceptance game for language $L^{\text{Rej}}_{\leqslant \text{Count}}(\mathcal{A})$.

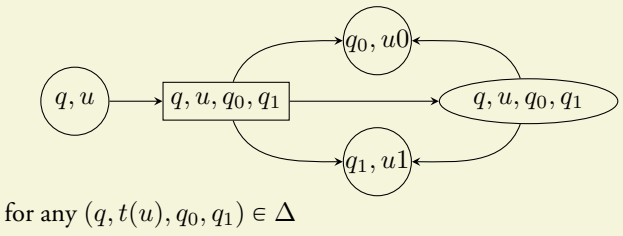


Figure 30 – Local structure of $\mathbb{G}^{\text{Rej}\leqslant\text{Count}}_{\mathcal{A},t}$.

Theorem 30 [CHS14a]

One has $t \in L^{\text{Rej}}_{\leqslant \text{Count}}(\mathcal{A})$ if and only if Éloïse wins in $\mathbb{G}^{\text{Rej}\leqslant\text{Count}}_{\mathcal{A},t}$ from $(q_{\text{ini}}, \varepsilon)$.

Consider game $\mathbb{G}^{\text{Rej}\leqslant\text{Count}}_{\mathcal{A},t}$ and modify it so that Éloïse is now announcing in advance which direction she would choose if Abélard let her do so. This new game is equivalent to the previous one (meaning that she wins in one game if and only if she does so in the other game). As this new game can easily be modified to obtain an *equivalent* acceptance game for the classical acceptance condition (as described in Section 2.3) one concludes that the languages of the form $L^{\text{Rej}}_{\leqslant \text{Count}}(\mathcal{A})$ are regular.

Theorem 31 [CHS14a]

Let $\mathcal{A} = \langle A, Q, q_{ini}, \Delta, \text{Col}\rangle$ be a parity tree automaton using d colours. Then there exists a parity tree automaton $\mathcal{A}' = \langle A, Q', q'_{ini}, \Delta', \text{Col}'\rangle$ such that $L^{\text{Rej}}_{\leqslant \text{Count}}(\mathcal{A}) = L(\mathcal{A}')$. Moreover $|Q'| = \mathcal{O}(d|Q|)$ and $\mathcal{A}'$ uses $d+1$ colours.

3.2.2 The Case of Languages $L^{\text{Rej}}_{\text{Fin}}(\mathcal{A})$

The following lemma characterises finite sets of branches by noting that for such a set there is a finite number of nodes belonging to at least two branches in the set.

Lemma 1 [CHS14a]

Let Π be a set of branches. Then Π is finite if and only if the set $W = \{u \in \{0,1\}^* \mid \exists \pi_0, \pi_1 \in \Pi \text{ s.t. } \pi_0 \neq \pi_1,\ u \sqsubseteq \pi_0 \text{ and } u \sqsubseteq \pi_1\}$ is finite. Equivalently, Π is finite if and only if there exists some $\ell \geqslant 0$ such that for all $u \in \{0,1\}^{\geqslant \ell}$ there is at most one $\pi \in \Pi$ such that $u \sqsubseteq \pi$.

Now, fix a tree t and define an acceptance game for $L^{\text{Rej}}_{\text{Fin}}(\mathcal{A})$ as follows. In this game (we refer the reader to Figure 31 for the local structure of the arena for game $\mathbb{G}^{\text{RejFin}}_{\mathcal{A},t}$) the two players move a pebble along a branch of t in a top-down manner: as in the classical case the players first select a transition and then a direction. The colour of the current state gives the colour of the configuration. There are three modes in this game: *wait* mode, *path* mode and *check* mode and the game starts in *wait* mode. Hence, (Éloïse's main) configurations in the game are elements of $Q \times \{0,1\}^* \times \{wait, path, check\}$.

Regardless of the mode, in a node u with state q Éloïse picks a transition $(q, t(u), q_0, q_1) \in \Delta$, and for each direction in $i \in \{0,1\}$ she proposes the next mode m_i in $\{wait, path, check\}$ (we describe below what are the possible options depending on the current mode). Then Abélard chooses a direction $j \in \{0,1\}$, the pebble is moved down to $u \cdot j$, the state is updated to q_j and the mode changes to m_j. The possible modes that Éloïse can propose depend on the current mode in the following manner.

- In *wait* mode she can propose any modes m_i in $\{wait, path, check\}$ but if one proposed mode m_i is *path* then the other mode m_{1-i} must be *check*.
- In *check* mode the proposed modes must be *check* (*i.e.* once the mode is *check* it no longer changes).
- In *path* mode one proposed mode must be *path* and the other must be *check*.

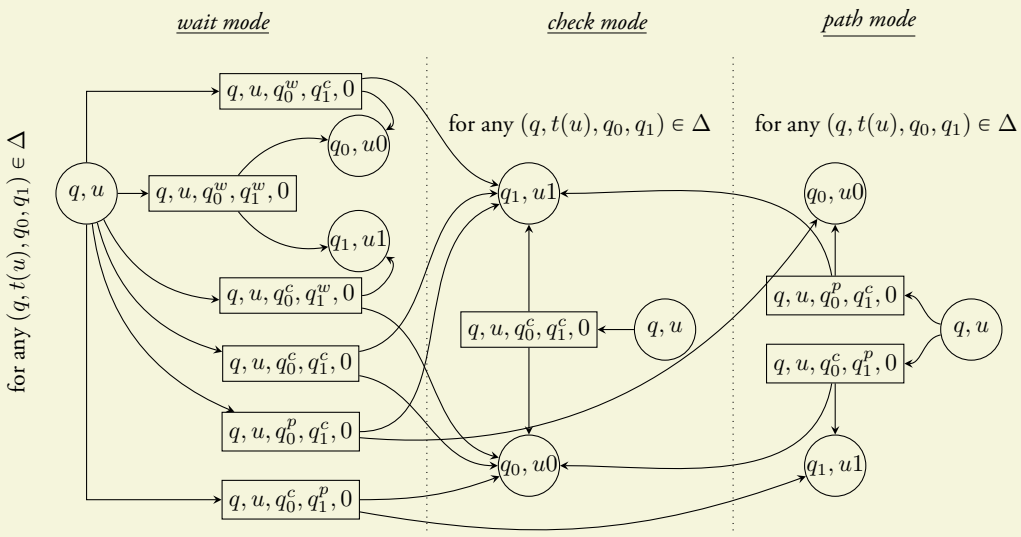


Figure 31 – Local structure of the arena of the acceptance game $\mathbb{G}_{\mathcal{A},t}^{\text{RejFin}}$. We use superscript to indicate which modes have been proposed by Éloïse.

A play is won by Éloïse if one of the two following situation occurs.

- The *wait* mode is eventually left and the parity condition is satisfied.
- The mode is eventually always equal to *path*.

In particular a play in which the mode is *wait* forever is lost by Éloïse. Note that the latter winning condition can easily be reformulated as a parity condition. Call this game $\mathbb{G}_{\mathcal{A},t}^{\text{RejFin}}$.

The next theorem states that it is an acceptance game for language $L_{\text{Fin}}^{\text{Rej}}(\mathcal{A})$.

Theorem 32 [CHS14a]

One has $t \in L_{\text{Fin}}^{\text{Rej}}(\mathcal{A})$ if and only if Éloïse wins in $\mathbb{G}_{\mathcal{A},t}^{\text{RejFin}}$ from $(q_{\text{ini}}, \varepsilon, wait)$.

From Theorem 32 and the local structure of the arena of game $\mathbb{G}_{\mathcal{A},t}^{\text{RejFin}}$ one easily concludes that any language of the form $L_{\text{Fin}}^{\text{Rej}}(\mathcal{A})$ is regular.

Theorem 33 [CHS14a]

Let $\mathcal{A} = \langle A, Q, q_{ini}, \Delta, \text{Col}\rangle$ be a parity tree automaton using d colours. Then there exists a parity tree automaton $\mathcal{A}' = \langle A, Q', q'_{ini}, \Delta', \text{Col}'\rangle$ such that $L_{\text{Fin}}^{\text{Rej}}(\mathcal{A}) = L(\mathcal{A}')$. Moreover $|Q'| = \mathcal{O}(|Q|)$ and $\mathcal{A}'$ uses d colours.

3.2.3 Languages $L_{\leqslant\text{Count}}^{\text{Rej}}(\mathcal{A})$ and $L_{\text{Fin}}^{\text{Rej}}(\mathcal{A})$ *vs* Büchi Tree Languages

One can wonder, as it will be later the case (see Section 3.3) for languages of the form $L_{\infty}^{\text{Acc}}(\mathcal{A})$ or $L_{Uncount}^{\text{Acc}}(\mathcal{A})$, whether a Büchi condition is enough to accept (with the classical semantics) a language of the form $L_{\leqslant\text{Count}}^{\text{Rej}}(\mathcal{A})$ (resp. $L_{\text{Fin}}^{\text{Rej}}(\mathcal{A})$). The next Proposition answers negatively.

Proposition 8 [CHS14a]

There is a co-Büchi deterministic tree automaton $\mathcal{A}$ such that for any *Büchi* tree automaton $\mathcal{A}'$, $L^{\text{Rej}}_{\leqslant\text{Count}}(\mathcal{A}) \neq L(\mathcal{A}')$ and $L^{\text{Rej}}_{\text{Fin}}(\mathcal{A}) \neq L(\mathcal{A}')$.

3.3 Counting Accepting Branches

We now consider the case where acceptance is defined by a constraint on the number of accepting branches and we show that the associated languages are regular. It leads to new proofs, that rely on games rather than on logic, of the results in [BN95] .

Fix, for this section, a parity tree automaton $\mathcal{A} = \langle A, Q, q_{ini}, \Delta, \text{Col}\rangle$ and recall that a tree t is in $L^{\text{Acc}}_{\infty}(\mathcal{A})$ (resp. $L^{\text{Acc}}_{Uncount}(\mathcal{A})$) if and only if there is a run of $\mathcal{A}$ on t that contains infinitely (resp. uncountably) many accepting branches.

3.3.1 The Case of Languages $L^{\text{Acc}}_{\infty}(\mathcal{A})$

The key idea behind defining an acceptance game for $L^{\text{Acc}}_{\infty}(\mathcal{A})$ for some tree t is to exhibit a *pseudo comb* in a run of $\mathcal{A}$ over t. In a nutshell, a pseudo comb consists of an infinite branch U and a collection V of accepting branches each of them sharing some prefix with U. One easily proves that a run contains infinitely many accepting branches if and only if it contains a pseudo comb.

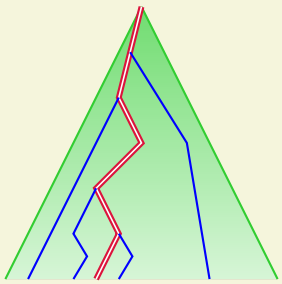


Figure 32 – A pseudo comb $(\mathbb{U}, V)$

More formally, a ***pseudo comb*** (see Figure 32 for an illustration) is a pair of subset (U, V) of nodes with $U, V \subseteq \{0,1\}^*$ such that:

- U and V are disjoint.
- U is a branch: $\varepsilon \in U$ and for all $u \in U$ one has $|\{u0, u1\} \cap U| = 1$.
- V is a set of nodes such that
  - (i) for all $v \in V$ one has $|\{v0, v1\} \cap V| = 1$;
  - (ii) for all $v \in V$, $v \in (U \cup V) \cdot \{0,1\}$.
- For infinitely many $u \in U$ there exists some $v \in V$ such that either $v = u0$ or $v = u1$.

The following folklore lemma characterises infinite sets of branches in the full binary tree.

Lemma 2

Let Π be a set of branches. Then Π is infinite if and only if the set $W = \{w \mid \exists \pi \in \Pi \text{ s.t. } w \text{ belongs to } \pi\}$ contains a pseudo comb (U, V), *i.e.* $U \cup V \subseteq W$.

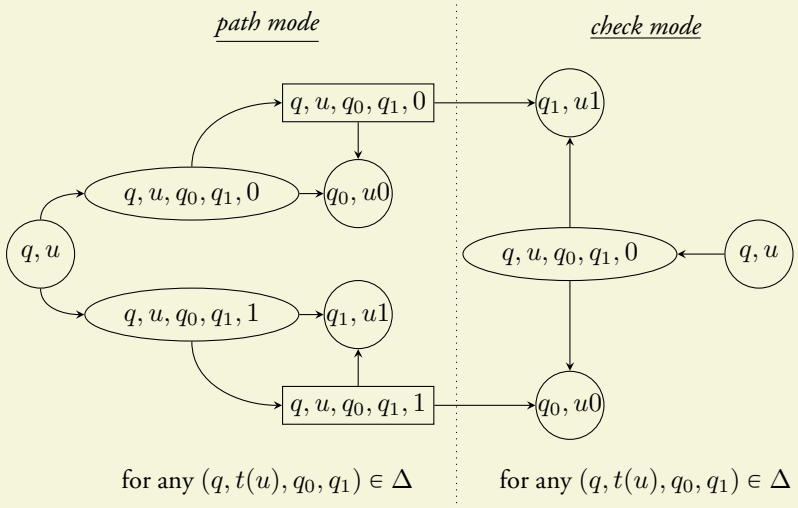


Figure 33 – Local structure of the arena of the acceptance game $\mathbb{G}^{\text{Acc}\,\infty}_{\mathcal{A},t}$.

Now, fix a tree t and define an acceptance game for $L^{\text{Acc}}_{\infty}(\mathcal{A})$. There are two modes in the game (See Figure 33 for the local structure of the arena): *path* mode and *check* mode and the game starts in *path* mode. Hence, (Éloïse's) configurations in the game are elements of $Q \times \{0,1\}^* \times \{path, check\}$. In *path* mode, in a node u with state q Éloïse picks a transition $(q, t(u), q_0, q_1) \in \Delta$, and she chooses a direction $i \in \{0,1\}$. Then Éloïse has two options. Either she moves down the pebble to ui and updates the state to be q_i. Or she proposes Abélard to change to *check* mode: if he accepts, the pebble is moved down to $u \cdot (1-i)$ and the state is updated to $q_{(1-i)}$; if he refuses, the pebble is moved down to ui and the state is updated to q_i

In *check* mode Éloïse plays alone: in a node u with state q she picks a transition $(q, t(u), q_0, q_1) \in \Delta$, and she chooses a direction $i \in \{0,1\}$; then the pebble is moved down to ui and the state is updated to q_i. Note that there is no possible switch from *check* mode to *path* mode.

A play is won by Éloïse if one of the two following situations occurs.

- Eventually the players switched to *check* mode and the parity condition is satisfied.
- Éloïse proposed infinitely often to Abélard to switch the mode but he systematically refused.

Call this game $\mathbb{G}^{\text{Acc}\,\infty}_{\mathcal{A},t}$.

The next theorem states that it is an acceptance game for language $L^{\text{Acc}}_{\infty}(\mathcal{A})$.

Theorem 34 [CHS14a]

One has $t \in L^{\text{Acc}}_{\infty}(\mathcal{A})$ if and only if Éloïse wins in $\mathbb{G}^{\text{Acc}\,\infty}_{\mathcal{A},t}$ from $(q_{\text{ini}}, \varepsilon, path)$.

One can modify $\mathbb{G}^{\text{Acc}\,\infty}_{\mathcal{A},t}$ so that to obtain an *equivalent* game that has the form of a classical acceptance game. From this follows the fact that the languages of the form $L^{\text{Acc}}_{\infty}(\mathcal{A})$ are indeed

regular. As the new game can be seen to be obtained from a Büchi automaton, this also permits to lower the acceptance condition.

Theorem 35 [BN95; CHS14a]

Let $\mathcal{A} = \langle A, Q, q_{ini}, \Delta, \text{Col}\rangle$ be a parity tree automaton using d colours. Then there exists a *Büchi* tree automaton $\mathcal{A}' = \langle A, Q', q'_{ini}, \Delta', \text{Col}'\rangle$ such that $L^{\text{Acc}}_{\infty}(\mathcal{A}) = L(\mathcal{A}')$. Moreover $|Q'| = \mathcal{O}(d|Q|)$.

3.3.2 The Case of Languages $L^{\text{Acc}}_{Uncount}(\mathcal{A})$

The key idea behind defining an acceptance game for $L^{\text{Acc}}_{Uncount}(\mathcal{A})$ for some tree t is to exhibit an *accepting pseudo binary tree* in a run of $\mathcal{A}$ over t. In a nutshell, an accepting pseudo binary tree is an infinite set U of nodes with a tree-like structure between them and such that any branch that has infinitely many prefixes in U is accepting.

Formally, let ρ be a run of $\mathcal{A}$ on some tree t. Then an ***accepting-pseudo binary tree*** in ρ (see Figure 34 for an illustration) is a subset $U \subseteq \{0,1\}^*$ of nodes such that

(i) for all $u \in U$ there are $v, w \in U$ such that $v = u0v'$ and $w = u1w'$ for some v' and $w' \in \{0,1\}^*$;

(ii) for all $v, w \in U$ the largest common prefix u of v and w belongs to U;

(iii) any branch π that goes through infinitely many nodes in U is accepting.

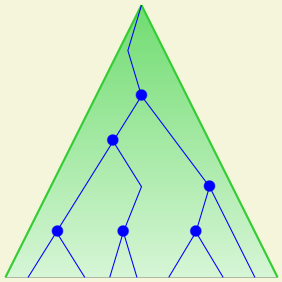


Figure 34 – An accepting-pseudo binary tree U: nodes in U are marked by symbol • and all blue branches are accepting.

The following lemma characterises runs that contains an uncountable sets of accepting branches. Its proof is a direct consequence of Beauquier and Niwiński [BN95, Lemma 2].

Lemma 3

Let ρ be a run. Then ρ contains uncountably many accepting branches if and only if it contains an accepting-pseudo binary tree.

Fix a tree t and define an acceptance game for $L^{\text{Acc}}_{Uncount}(\mathcal{A})$. In this game (see Figure 35 for the local structure of the arena) the two players move a pebble along a branch of t in a top-down manner: the pebble is attached a state and the colour of the state gives the colour of the configuration. Hence, (Éloïse's main) configurations in the game are elements of $Q \times \{0,1\}^*$. In a node u with state q

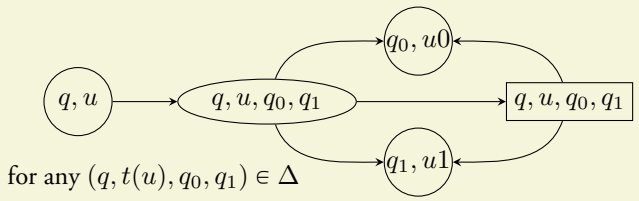


Figure 35 – Local structure of the arena of the acceptance game $\mathbb{G}^{\text{Acc Uncount}}_{\mathcal{A},t}$.

Éloïse picks a transition $(q, t(u), q_0, q_1) \in \Delta$, and then Éloïse has two possible options. Either she chooses a direction 0 or 1 or she lets Abélard choose a direction 0 or 1. Once the direction $i \in \{0, 1\}$ is chosen, the pebble is moved down to ui and the state is updated to q_i. A play is won by Éloïse if and only if

(1) the parity condition is satisfied and

(2) Éloïse lets Abélard infinitely often choose the direction during the play.

Call this game $\mathbb{G}^{\text{Acc Uncount}}_{\mathcal{A},t}$.

The next theorem states that it is an acceptance game for language $L^{\text{Acc}}_{Uncount}(\mathcal{A})$.

Theorem 36 [CHS14a]

One has $t \in L^{\text{Acc}}_{Uncount}(\mathcal{A})$ if and only if Éloïse wins in $\mathbb{G}^{\text{Acc Uncount}}_{\mathcal{A},t}$ from $(q_{\text{ini}}, \varepsilon)$.

One can modify $\mathbb{G}^{\text{Acc Uncount}}_{\mathcal{A},t}$ so that to obtain an *equivalent* game that has the form of a classical acceptance game. From this follows the fact that the languages of the form $L^{\text{Acc}}_{Uncount}(\mathcal{A})$ are indeed regular. Using a more involved game than $\mathbb{G}^{\text{Acc Uncount}}_{\mathcal{A},t}$ one can lower the acceptance condition to a Büchi condition (see [CHS14a] for a description of that game).

Theorem 37 [BN95; CHS14a]

Let $\mathcal{A} = \langle A, Q, q_{ini}, \Delta, \text{Col}\rangle$ be a parity tree automaton using d colours. Then there exists a *Büchi* tree automaton $\mathcal{A}' = \langle A, Q', q'_{ini}, \Delta', \text{Col}'\rangle$ such that $L^{\text{Acc}}_{Uncount}(\mathcal{A}) = L(\mathcal{A}')$. Moreover $|Q'| = \mathcal{O}(d|Q|)$.

4 Requiring Large Sets of Accepting Branches: Automata with Topological Bigness Constraints

Another option to quantify over sets of accepting branches in a run is to use a notion of *topological* "bigness" and to require for a run to be accepting that the set of accepting branches is topologically large. Topological largeness was previously considered in the literature (*e.g.* with motivations from program verification in order to define fairness properties of transition systems [VV12]) with a special focus on its tight connection with so-called Banach-Mazur games (see *e.g.* [Grä08] for a survey on that topic). Together with Arnaud Carayol and Axel Haddad we have shown in [CHS14a] that the tree languages accepted with this new condition are indeed regular. The roadmap to this result is the same as for the case of automata with cardinality constrains that we considered in Section 3: first, one characterises topologically large sets of accepting branches, then one uses this

characterisation to design an acceptance game and finally one argues that it can be seen as another equivalent acceptance game but now for the usual semantics on runs.

4.1 Automata with Topological Bigness Constraints

Consider a tree t and a node u. Then the ***cone*** going through a node $u \in \{0,1\}^*$ is the set $\text{Cone}(u) = u \cdot \{0,1\}^\omega$.

One can see the set of branches in a tree as a topological space by taking as basic open sets the cones. A notion of topological "bigness" is given by *large* sets. A set of branches $B \subseteq \{0,1\}^\omega$ is ***nowhere dense*** if for all node u, there exists $v \in \{0,1\}^*$ such that such that no branch of B has uv as a prefix. It is ***meagre*** if it is the countable union of nowhere dense sets. Finally it is ***large*** if it is the complement of a meagre set.

For a given automaton $\mathcal{A}$, we define the following acceptance criterion: a run is accepting if and only if the set of accepting branches in it is large. Note that this is equivalent to require that the set of rejecting branches is meagre. Finally, a tree $t \in L^{\text{Acc}}_{\text{Large}}(\mathcal{A})$ if and only if there is a run of $\mathcal{A}$ on t satisfying the previous condition.

4.2 Acceptance Game & Regularity

In [CHS14a] we proved with Arnaud Carayol and Axel Haddad that languages of the form $L^{\text{Acc}}_{\text{Large}}(\mathcal{A})$ are always regular.

Theorem 38 [CHS14a]

Let $\mathcal{A} = \langle A, Q, q_{ini}, \Delta, \text{Col}\rangle$ be a parity tree automaton using d colours. Then there exists a parity tree automaton $\mathcal{A}' = \langle A, Q', q'_{ini}, \Delta', \text{Col}'\rangle$ such that $L^{\text{Acc}}_{\text{Large}}(\mathcal{A}) = L(\mathcal{A}')$. Moreover $|Q'| = \mathcal{O}(d|Q|)$ and $\mathcal{A}'$ uses $d+2$ colours.

The roadmap to prove Theorem 38 is as follows. First one characterises large set of branches (Lemma 4), then based on this, one defines an acceptance game for $L^{\text{Acc}}_{\text{Large}}(\mathcal{A})$ and finally one transforms it so that to obtain an *equivalent* game that has the form of a classical acceptance game (as described in Section 2.3) from which one extracts $\mathcal{A}'$.

Call a set of nodes $U \subseteq \{0,1\}^*$ ***dense*** if for all node $v \in \{0,1\}^*$, there exists a node $u \in U$ such that $v \sqsubseteq u$. Given a dense set of nodes U, the set of ***branches supported by U***, $\mathcal{B}(U)$ is the set of branches that have infinitely many prefixes in U. The following lemma characterises large sets of branches.

Lemma 4 [CHS14a]

A set of branches $B \subseteq \{0,1\}^\omega$ is large if and only if there exists a dense set of nodes $U \subseteq \{0,1\}^*$ such that $\mathcal{B}(U) \subseteq B$.

Fix a tree t and define an acceptance game $\mathbb{G}^{\text{Acc}\,Large}_{\mathcal{A},t}$ for $L^{\text{Acc}}_{\text{Large}}(\mathcal{A})$. In this game, Éloïse describes a run ρ together with a dense set U of nodes while Abélard tries either to prove that U is not dense or that there is a rejecting branch in $\mathcal{B}(U)$. The way Éloïse describes a run is as usual (she proposes valid transitions); the way she describes U is by (1) indicating explicitly when a node is in U and; (2) at each node giving a direction i that should lead (by iteratively following the directions) to node

in U. Abélard chooses the direction: if it does not select i and does not go to a node in U the colour is a large even one (preventing him not to follow Éloïse forever); if he chooses i but does not go to a node in U the colour is a large odd one (forcing Éloïse to describe a dense set U); and if he chooses i and goes to a node in U the colour is the smallest one seen since the last visit to a node in U (and it is computed in the game).

The next theorem states that $\mathbb{G}_{\mathcal{A},t}^{\mathrm{Acc}\,Large}$ is an acceptance game for $L_{\mathrm{Large}}^{\mathrm{Acc}}(\mathcal{A})$. It also easily permits to prove Theorem 38 above.

Theorem 39 [CHS14a]

One has $t \in L_{\mathrm{Large}}^{\mathrm{Acc}}(\mathcal{A})$ if and only if Éloïse wins in $\mathbb{G}_{\mathcal{A},t}^{\mathrm{Acc}\,Large}$.

5 Measuring Branches in Trees: Qualitative Tree Languages

In the classical definition (cf Section 2.2 above), a run of a tree automaton $\mathcal{A}$ is accepting if all its branches satisfy the acceptance condition. In collaboration with Arnaud Carayol and Axel Haddad we introduced in [CHS11] a more relaxed notion of acceptation: a run is qualitatively accepting if almost every (in the sense of the measure μ defined below) branch in it is accepting.

5.1 Definitions

We assume that the reader is familiar with basic notions of measure theory and from probability theory and we use [Bau01; Bau96] as references for all known results related to this field. The ***cone*** going through a node $u \in \{0,1\}^*$ is the set $\mathrm{Cone}(u) = u\cdot\{0,1\}^\omega$. A ***sub-cone*** of a cone $\mathrm{Cone}(u)$ is a cone $\mathrm{Cone}(v)$ with $u \sqsubseteq v$. Let $\mathcal{F}_{\mathrm{Br}}$ be the σ-algebra generated by the set of cones (*i.e.* the smallest set of subsets of $\{0,1\}^\omega$ containing the cones and closed under countable union and complementation). Let μ be the unique probability measure on $\mathcal{F}_{\mathrm{Br}}$ such that for all $u \in \{0,1\}^*$, $\mu(\mathrm{Cone}(u)) = 2^{-|u|}$. The existence and unicity of μ are guaranteed by Carathéorody's extension theorem [Bau01]. For all $0 < p < 1$, a probability measure μ_p is similarly defined by taking $\mu_p(\mathrm{Cone}(u)) = p^{|u|_0}(1-p)^{|u|_1}$ where $|u|_0$ and $|u|_1$ respectively designate the number of occurrences of 0 and 1 in u. In particular, the measure μ corresponds to $\mu_{1/2}$.

Now, consider a tree automaton $\mathcal{A}$ with an ω-regular acceptance condition Acc. A run ρ of $\mathcal{A}$ is ***qualitatively accepting*** if the set $\mathrm{AccBr}(\rho) = \{\pi \in \{0,1\}^\omega \mid \rho(\pi) \in \mathrm{Acc}\}$ has measure 1, *i.e.* $\mu(\mathrm{AccBr}(\rho)) = 1$. Note that, thanks to Proposition 9 below, the set $\mathrm{AccBr}(\rho)$ is indeed measurable hence, the previous definition makes sense.

Proposition 9 [CHS11]

Let $\mathcal{A}$ be a tree automaton equipped with an ω-regular acceptance condition, let t be an input tree and let ρ be a run of $\mathcal{A}$ over t. The set $\mathrm{AccBr}(\rho)$ is measurable.

A tree t is ***qualitatively accepted*** by $\mathcal{A}$ if there exists a qualitatively accepting run of $\mathcal{A}$ over t and the set of all trees qualitatively accepted by $\mathcal{A}$ is denoted $L_{\mathrm{Qual}}(\mathcal{A})$. Finally, a ***qualitative tree language*** is a set L of trees such that there exists a parity automaton $\mathcal{A}$ such that $L_{\mathrm{Qual}}(\mathcal{A}) = L$.

***Remark* 30.** *One can wonder how the notion of qualitatively accepting run compares with the existence of a run with a topologically large set of accepting branches (in the sense of Section 4). Indeed, the two*

notions are incomparable: in particular there exists a run (on the dummy tree) whose set of accepting branches is topologically large while having measure 0 *(see* e.g. *[VV12]).*

Example 27

Let $\mathcal{L}_a$ be the language of $\{a,b\}$-labelled trees whose set of branches containing at least one a has measure 1. This language is recognised by the following reachability deterministic automaton $\mathcal{A} = \langle \{a,b\}, \{q_{\text{ini}}, q_f\}, q_{\text{ini}}, \Delta, \{q_f\}\rangle$ where: $\Delta = \{q_{\text{ini}} \xrightarrow{b} (q_{\text{ini}}, q_{\text{ini}}), q_{\text{ini}} \xrightarrow{a} (q_f, q_f), q_f \xrightarrow{a} (q_f, q_f), q_f \xrightarrow{b} (q_{\text{ini}}, q_{\text{ini}})\}$.

If one considers $\mathcal{A}$ as a *Büchi* automaton, the accepted language consists of those trees whose set of branches containing infinitely many a has measure 1.

Example 28

Let $\mathcal{L}_1$ be the language of trees t such that in almost every branch, there is a node u labelled by a such that the subtree $t[u]$ has only a on its leftmost branch. This language is recognised by the non-deterministic reachability automaton $\mathcal{A} = \langle A, Q, q_w, \Delta, \{q_{acc}\}\rangle$ with $A = \{a,b\}$, $Q = \{q_w, q_l, q_{acc}, q_{rej}\}$, and Δ contains the following transitions: $q_w \xrightarrow{*} (q_w, q_w)$, $q_w \xrightarrow{a} (q_l, q_{acc})$, $q_l \xrightarrow{a} (q_l, q_{acc})$, $q_l \xrightarrow{b} (q_{rej}, q_{rej})$, $q_{acc} \xrightarrow{*} (q_{acc}, q_{acc})$, $q_{rej} \xrightarrow{*} (q_{rej}, q_{rej})$ (here $*$ is a shorthand for an arbitrary letter). Intuitively, the automaton can wait in state q_w as long as it wants. Using the second transition, the automaton can guess that the node u (labeled by a) has a leftmost branch containing only a. This assumption is checked by sending on the leftmost branch the state q_l and the accepting state q_{acc} on all other branches. As long as the nodes are labelled by a state q_l is propagated to the left child. If all nodes on the leftmost branch starting at u are labelled by a, this branch will be rejecting, but this does not affect the measure as there are only countably many such branches). If a node v labelled by b is encountered in state q_l the non-accepting state q_{rej} is propagated on all branches. This last scenario cannot occur in an accepting run as these cones of rejecting branches have a strictly positive measure. Hence the automaton is penalised for wrong guesses.

For the same reasons as for regular tree languages (cf. Remark 27), the parity condition is expressive enough to capture any ω-regular condition: for any automaton $\mathcal{A}$ with an ω-regular acceptance condition, there exists a parity automaton $\mathcal{B}$ such that $L_{\text{Qual}}(\mathcal{A}) = L_{\text{Qual}}(\mathcal{B})$.

Thanks to the following proposition, we can only focus on complete automata.

Proposition 10 [CHS11]

For any tree automaton $\mathcal{A}$ with an ω-regular acceptance condition, there exists a *complete* tree automaton $\mathcal{B}$ with the same acceptance condition and such that $L_{\text{Qual}}(\mathcal{A}) = L_{\text{Qual}}(\mathcal{B})$.

Unsurprisingly determinism is a restriction (consider *e.g.* the language $\mathcal{L}_a$ from Example 27).

Proposition 11 [CHS11]

There is a qualitative tree language that cannot be qualitatively accepted by any *deterministic* automaton.

The choice of the measure μ though natural is arbitrary. Considering the measure μ_p for some $0 < p < \frac{1}{2}$ would not affect the results obtained in this chapter (provided that definitions of the games are modified accordingly). However note that changing the measure does change the accepted language for a given automaton.

Proposition 12 [CHS11]

Let $0 < p < q < 1$ be two reals. Let $\mathcal{A}$ be the (deterministic and complete) automaton of Example 27. Then, there is a tree t such that $\mu_p(\mathrm{Acc}(\mathcal{A}, t)) = 0$ and $\mu_q(\mathrm{Acc}(\mathcal{A}, t)) = 1$, where $\mathrm{Acc}(\mathcal{A}, t)$ denotes the set of accepting branches for the unique run of $\mathcal{A}$ over t.

5.2 Pumping Lemma and Closure Properties

Let t be a tree and $u \in \{0,1\}^*$ be a node. A pair $\Delta = (t, u)$ is called a ***pointed tree***. With a pointed tree $\Delta_1 = (t_1, u_1)$ and a tree t_2, we associate a new tree, $\Delta_1 \cdot t_2$, by plugging t_2 in t_1 instead of the subtree rooted at u_1. Formally, $\Delta_1 \cdot t_2(u) = t_1(u)$ if u_1 is not a prefix of u and $\Delta_1 \cdot t_2(u) = t_2(u')$ if $u = u_1 u'$ for some $u' \in \{0,1\}^*$. We can also define the product of two pointed trees $\Delta_1 = (t_1, u_1)$ and $\Delta_2 = (t_2, u_2)$ by letting $\Delta_1 \cdot \Delta_2 = (\Delta_1 \cdot t_2, u_1 \cdot u_2)$. Finally, with a pointed tree $\Delta = (t, u)$, we associate a tree Δ^ω by taking an ω-iteration of the product: $\Delta^\omega(v) = t(v')$ where v' is the shortest word such that $v = u^k v'$ for some $k \geqslant 0$.

Qualitative tree languages enjoy a pumping lemma (see Figure 5 for an illustration), which contrasts with regular tree languages.

Lemma 5 [CHS11]

Let $\mathcal{A}$ be an n-state parity automaton, t be a tree in $L_{\mathrm{Qual}}(\mathcal{A})$ and u be a node of depth greater that n. Then there exists three pointed trees Δ_1, Δ_2 and Δ_3 such that $t = \Delta_1 \cdot \Delta_2 \cdot \Delta_3 \cdot t_u$ and $\Delta_1 \cdot \Delta_2^\omega \in L_{\mathrm{Qual}}(\mathcal{A})$.

The following proposition summarises the closure properties of qualitative tree languages under Boolean operations (positive results are obtained using the usual product of automata while the negative result follows from the pumping lemma applied to the complement of language $\mathcal{L}_a$ from Example 27).

Proposition 13 [CHS11]

Qualitative tree languages are closed under union and intersection but not under complement.

5.3 Regular Tree Languages and Qualitative Tree Languages are Incomparable

The next proposition shows that regular tree languages and qualitative tree languages are incomparable. For one direction, it suffices *e.g.* to consider the regular language of those trees that contains at least one node labeled a, reason by contradiction and use the pumping lemma; conversely, one considers *e.g.* the language $\mathcal{L}_a$ from Example 27, reasons by contradiction and mainly relying on closure properties of regular tree languages constructs another regular tree languages without regular tree inside while being not empty, hence leading to the contradiction.

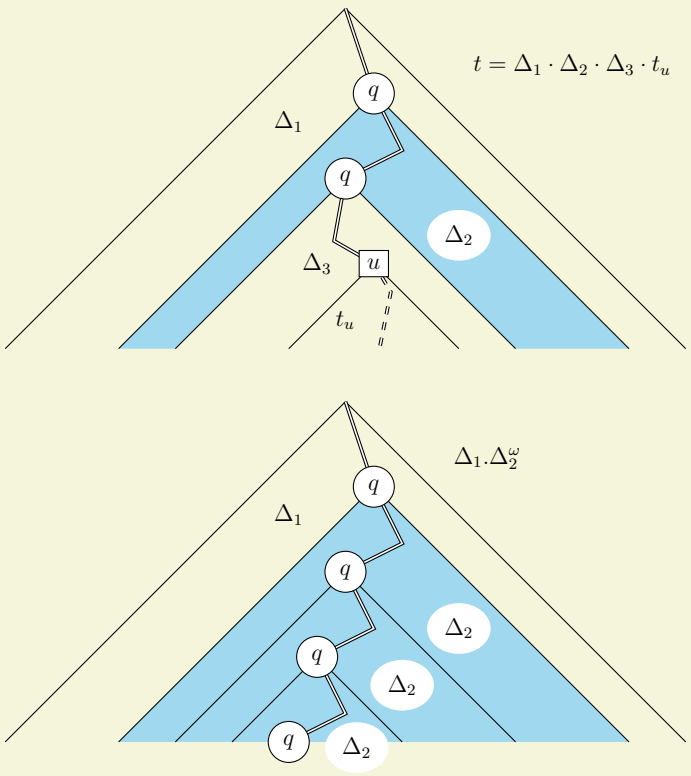


Figure 36 – Pumping Lemma

Proposition 14 [CHS11]

Regular Tree Languages and Qualitative Tree Languages are Incomparable.

- There is a regular tree language that is not qualitative.
- There is a qualitative tree language that is not regular.

5.4 Markov Decision Processes

We now introduce Markov Decision Processes that will later be used in Section 5.5 as a crucial tool to check the emptiness of qualitative tree languages.

A ***Markov arena*** (or simply an *arena*) is a tuple $\mathcal{G} = \langle S, s_{\text{ini}}, \Sigma, \delta \rangle$ where S is a countable set of states, s_{ini} is an initial state, Σ is a finite set of ***actions*** and $\delta : S \times \Sigma \to \mathcal{D}(S)$ is the transition (total) function.

A play in a Markov arena proceeds as follows. It starts in state s_{ini} and Éloïse picks an action σ, and a successor state is chosen according to the probability distribution $\delta(s_{\text{ini}}, \sigma)$. Then Éloïse chooses a new action and the state is updated and so on forever. Hence a ***play*** is an infinite sequence $s_0 s_1 s_2 \cdots \in S^\omega$ such that $s_0 = s_{\text{ini}}$ and for every $i \geqslant 0$, there exists a $\sigma \in \Sigma$ with $\delta(s_i, \sigma)(s_{i+1}) > 0$. In the sequel we refer to a prefix of a play as a ***partial play*** and we denote by Plays the set of all plays.

A ***(pure) strategy***[6] for Éloïse is a function $\varphi : S^* \to \Sigma$ assigning to every partial play an action.

[6] We do not consider here randomised strategies as in this setting they are useless. Note that for finite MDP, optimal strategies – when exists – can always be chosen to be pure. More general strategies in a more general setting are discussed

Of special interest are those strategies that do not require memory: a strategy φ is ***positional*** if $\varphi(\lambda \cdot s) = \varphi(\lambda' \cdot s)$ for all partial play λ, λ' and all states s (*i.e.* φ only depends on the current state). A play $\lambda = s_0 s_1 s_2 \cdots$ is ***consistent*** with a strategy φ if $\delta(s_i, \varphi(s_0 \cdots s_i))(s_{i+1}) > 0$, for all $i \geqslant 0$.

Now, for any partial play λ, the ***cylinder*** for λ is the set $\mathrm{Cyl}(\lambda) = \lambda S^\omega \cap \mathrm{Plays}$. Let $\mathcal{F}_P$ be the σ-algebra generated by the set of cylinders. Then, $(\mathrm{Plays}, \mathcal{F}_P)$ is a measurable space.

A strategy φ induces a probability space over $(\mathrm{Plays}, \mathcal{F}_P)$ as follows: one defines a measure μ_φ on cylinders and then uniquely extends it to a probability measure on $\mathcal{F}_P$ using the Carathéodory's unique extension theorem [Bau01]. For this, we first define inductively μ_φ on cylinders:

- as all plays start from s_{ini}, we let $\mu_\varphi(\mathrm{Cyl}(s_{\text{ini}})) = 1$;
- for any partial play λ ending in some state s, we let $\mu_\varphi(\mathrm{Cyl}(\lambda \cdot s')) = \mu_\varphi(\mathrm{Cyl}(\lambda)) \cdot \delta(s, \varphi(\lambda))(s')$.

We also denote by μ_φ the unique extension of μ_φ to a probability measure on $\mathcal{F}$. Then $(\mathrm{Plays}, \mathcal{F}_P, \mu_\varphi)$ is a probability space.

An ***objective*** is a measurable set $\mathcal{O} \subseteq \mathrm{Plays}$: a play is winning if it belongs to $\mathcal{O}$. A ***Markov decision process (MDP, aka* one-and-half-player game*)*** is a pair $\mathbb{G} = (\mathcal{G}, \mathcal{O})$ where $\mathcal{G}$ is a Markov arena and $\mathcal{O}$ is an objective. In the sequel we should focus on ω-regular objectives (which are easily seen to be measurable), whose definitions are the same as for the acceptance condition on tree automata (the only differences is that we may have an infinite set of states and that we restrict ourselves to the set Plays).

A strategy φ is ***almost-surely winning*** (*resp.* ***positively winning***) if $\mu_\varphi(\mathcal{O}) = 1$ (*resp.* $\mu_\varphi(\mathcal{O}) > 0$). If such a strategy exists, we say that Éloïse ***almost-surely wins*** (*resp.* ***positively wins***) $\mathbb{G}$. The ***value*** of $\mathbb{G}$ is defined as $\mathrm{Val}(\mathbb{G}) = \sup_\varphi \mu_\varphi(\mathcal{O})$, and a strategy φ is ***optimal*** if $\mathrm{Val}(\mathbb{G}) = \mu_\varphi(\mathcal{O})$.

When the set of actions Σ is reduced to one element, the MDP $(\mathcal{G}, \mathcal{O})$ is called a ***Markov chain*** and we omit the unique action from all the definitions. The set Plays is called the set of ***traces*** of the Markov chain and is denoted Traces. We write $\mu_\mathcal{G}$ the probability measure associated with the unique strategy. We say that the Markov chain ***almost-surely*** fulfils its objective if $\mu_\mathcal{G}(\mathcal{O}) = 1$.

MDPs over finite arenas enjoys many good properties in particular the one described in the following theorem.

Theorem 40 [CY90; CJH04]

Let $\mathbb{G}$ be an MDP over a finite arena with a parity objective. Then, one can decide in polynomial time whether Éloïse almost-surely (*resp.* positively) wins. Moreover, Éloïse always has an optimal memoryless strategy.

5.5 The Emptiness Problem

As recalled in Section 2.3, it is well known that tree automata (as acceptors of regular languages) and two-player (perfect information) games are closely related. In particular, the emptiness problem for regular tree languages and the problem of deciding the winner in a parity game on a finite graph are polynomially equivalent. From the proof of this result also follows that a regular tree language is non-empty if and only if it contains a regular tree.

We show that a similar connection exists between tree automata as acceptors of qualitative tree languages and MDP. For this, fix a parity tree automaton $\mathcal{A} = \langle A, Q, q_{\text{ini}}, \Delta, \mathrm{Col} \rangle$ and a tree

in Section 6.

t. Consider the Markov arena, depicted in Figure 37, $\mathcal{G}_{\mathcal{A},t} = \langle S, s_{\text{ini}}, \Sigma, \delta \rangle$ where $S = Q \times \{0,1\}^* \cup \{\bot\}$, $s_{\text{ini}} = (q_{\text{ini}}, \varepsilon)$, $\Sigma = \Delta$ and δ is defined as follows. First we let $d_\bot$ be the distribution defined by $d_\bot(s) = 1$ if $s = \bot$ and $d_\bot(s) = 0$ otherwise, and, for all $q_0, q_1 \in Q$ and $u \in \{0,1\}^*$, we let $d_{q_0,q_1,u}$ be the distribution such that $d_{q_0,q_1,u}(q_0, u0) = d_{q_0,q_1,u}(q_1, u1) = 1/2$ and $d_{q_0,q_1,u}(s) = 0$ for all other $s \in S$. Then we let $\delta((q,u),(q',a,q_0,q_1)) = d_\bot$ if $q \neq q'$ or $a \neq t(u)$, $\delta((q,u),(q,t(u),q_0,q_1)) = d_{q_0,q_1,u}$ and $\delta(\bot,\tau) = d_\bot$ for all $\tau \in \Delta$. Finally, we define a colouring function ρ by letting $\rho((q,u)) = \text{Col}(q)$ and $\rho(\bot) = 1$, and we call $\mathbb{G}_{\mathcal{A},t} = (\mathcal{G}_{\mathcal{A},t}, \mathcal{O}_\rho)$ the MDP equipped with the parity objective $\mathcal{O}_\rho$ defined by ρ, *i.e.* $\mathcal{O}_\rho = \{s_0 s_1 s_2 \cdots \mid \liminf(\rho(s_i))_{i \geqslant 0} \text{ is even}\}$.

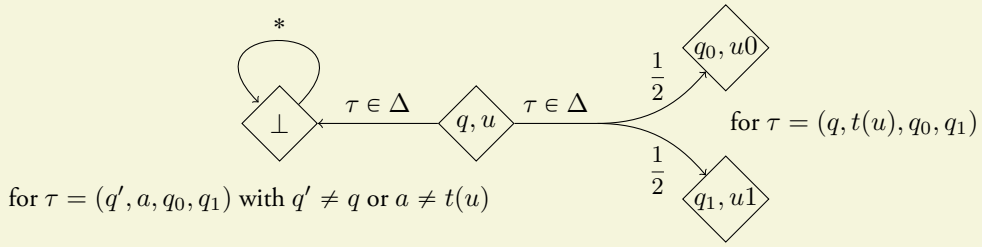


Figure 37 – Local structure of the Markov arena $\mathcal{G}_{\mathcal{A},t}$ of the acceptance game.

Then, the following holds.

Theorem 41 [CHS11]

The tree t belongs to $L_{\text{Qual}}(\mathcal{A})$ if and only if Éloïse almost-surely wins in $\mathbb{G}_{\mathcal{A},t}$.

Now, in order to check the emptiness the idea is to slightly modify the previous acceptance game by additionally requiring Éloïse to describe a tree t in $L_{\text{Qual}}(\mathcal{A})$. Therefore, instead of picking a transition she also has to provide a node label. To make the resulting game finite one "forgets" about the nodes, *i.e.* the component in $\{0,1\}^*$.

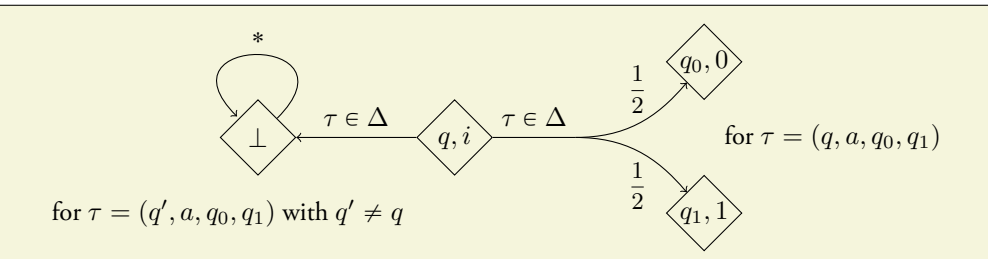


Figure 38 – Local structure of the Markov arena $\mathcal{G}_{\mathcal{A}}$ of the emptiness game.

Formally one considers the (finite) Markov arena $\mathcal{G}_{\mathcal{A}} = \langle S, s_{\text{ini}}, \Sigma, \delta \rangle$, depicted in Figure 38, where $S = Q \times \{0,1\} \cup \{q_{\text{ini}}, \bot\}$, $s_{\text{ini}} = q_{\text{ini}}$, $\Sigma = \Delta$ and δ is defined as follows. First we let $d_\bot$ be the distribution defined by $d_\bot(s) = 1$ if $s = \bot$ and $d_\bot(s) = 0$ otherwise, and, for all $q_0, q_1 \in Q$, we let d_{q_0,q_1} be the distribution such that $d_{q_0,q_1}((q_0,0)) = d_{q_0,q_1}((q_1,1)) = 1/2$ and $d_{q_0,q_1}(s) = 0$ for all other $s \in S$. Then we let $\delta((q,i),(q,a,q_0,q_1)) = d_{q_0,q_1}$, $\delta((q,i),(q',a,q_0,q_1)) = d_\bot$ if $q \neq q'$, $\delta(q_{\text{ini}},(q_{\text{ini}},a,q_0,q_1)) = d_{q_0,q_1}$, $\delta(q_{\text{ini}},(q,a,q_0,q_1)) = d_\bot$ if $q \neq q_{\text{ini}}$, and $\delta(\bot,\tau) = d_\bot$ for all $\tau \in \Delta$. Finally, we define a colouring function ρ by letting $\rho((q,i)) = \text{Col}(q)$ and $\rho(\bot) = 1$,

and we call $\mathbb{G}_{\mathcal{A}} = (\mathcal{G}_{\mathcal{A}}, \mathcal{O}_\rho)$ the MDP equipped with the parity objective $\mathcal{O}_\rho$ defined by ρ. Then, the following hold.

Theorem 42 [CHS11]

The language $L_{\text{Qual}}(\mathcal{A})$ is non empty if and only if Éloïse almost-surely wins in $\mathbb{G}_{\mathcal{A}}$ from q_{ini}.

In particular, thanks to Theorem 40, it permits to decide the emptiness of a qualitative tree language in *polynomial* time, which contrast with the case of regular tree languages (where so far no polynomial time emptiness algorithm is known).

Corollary 11 [CHS11]

Let $\mathcal{A}$ be a parity tree automaton. Then one can decide whether $L_{\text{Qual}}(\mathcal{A}) = \varnothing$ in polynomial time. Moreover, if $L_{\text{Qual}}(\mathcal{A}) \neq \varnothing$, it contains a regular tree, and such a tree can also be constructed in polynomial time.

***Remark* 31.** *Motivated by a decision problem for qualitative tree language we designed a polynomial reduction of the emptiness problem to the problem of deciding almost-surely winning in a finite MDP.*

A similar connection exists between tree automata (as acceptors of regular tree languages) and two-player (perfect information) games (see e.g. *[GH82; GTW02]). Indeed, the emptiness problem for regular tree languages and the problem of deciding the winner in a parity game on a finite graph are polynomially equivalent.*

Hence, one may ask whether conversely the problem of deciding almost-surely winning in a finite MDP can be polynomially reduced to the emptiness problem for qualitative tree languages. It is indeed possible and the proof is very similar to the one from a two-player game to a regular tree language. We briefly sketch the proof below.

First note that one can, up to coding, restrict its attention to finite MDP with only two actions and such that from any state and any action there are always two possible successors that can both be reached with same probability $1/2$*. Then one designs a* deterministic *tree automaton whose states are identified with the ones of the MDP, whose input alphabet is identified with the set of actions of the MDP, and whose transition function mimics the one of the MDP. Then one concludes by noting that there is a bijection between trees and strategies and that this bijection is such that the measure of the accepting branches in the (unique) run of the automaton on a tree equals the value of the corresponding strategy in the MDP.*

5.6 Variants: Value of a Tree, Positive Tree Languages

We defined qualitative acceptance of a tree by the existence of a run whose set of accepting branches has measure 1. We can refine this notion by defining the ***value*** of a tree as follows. For a tree automaton $\mathcal{A}$, and and tree t we let

$$\text{Val}_{\mathcal{A}}(t) = \sup_{\rho_t \text{ run of } \mathcal{A} \text{ over } t} \mu(\text{AccBr}(\rho_t))$$

In particular $L_{\text{Qual}}(\mathcal{A})$ is the set of trees t whose value is 1 and is reached for some run (*i.e.* the sup is a max). The following result proves that the value may not be reached by some run.

Theorem 43 [CHS11]

There is a reachability automaton $\mathcal{A}$ and a tree t such that $\mathrm{Val}_{\mathcal{A}}(t) = 1$ but $t \notin L_{\mathrm{Qual}}(\mathcal{A})$.

If one is interested in computing the value of a tree, the proof of Theorem 41 directly leads to the following (hence, showing that this value can be computed).

Corollary 12 [CHS11]

Let $\mathcal{A}$ be a parity tree automaton and let t be a tree. Then $\mathrm{Val}_{\mathcal{A}}(t) = \mathrm{Val}(\mathbb{G}_{\mathcal{A},t})$.

So far, we favoured the almost-sure acceptance condition (*i.e.* requiring the measure to be equal to 1) over the positive one (*i.e.* requiring the measure to be strictly positive). However, the decidability results on MDP stated in Theorem 40 still hold if we replace the almost-sure acceptance by the positive acceptance [CY90; CJH04]. We now discuss the impact when considering positive acceptance instead of almost-sure acceptance, and motivate our choice to focus on almost-sure acceptance.

We say that a run ρ of a tree automaton $\mathcal{A}$ is ***positively accepting*** if the measure of its set of accepting branches is (strictly) positive, *i.e.* $\mu(\mathrm{AccBr}(\rho)) > 0$. A tree t is ***positively accepted*** if there exists a positively accepting run of $\mathcal{A}$ over t, and we denote by $L_{\mathrm{Pos}}(\mathcal{A})$ the set of all trees positively accepted by $\mathcal{A}$. Finally, a ***positive tree language*** is a language L of trees such that there exists a parity automaton $\mathcal{A}$ with $L_{\mathrm{Pos}}(\mathcal{A}) = L$.

Note that, contrarily to the qualitative semantics, we can no longer assume that our automata are complete. In particular, the naive idea of adding a sink state does not work, as it would lead to have new runs that may be accepting (going to the sink state leads to have a rejecting cone, but this may not affect the positivity of the measure of the set of accepting branches).

Example 29

Consider the language $\mathcal{L}_a^{>0}$ of $\{a, b\}$-labeled trees that have a non-negligible set of branches containing infinitely many a's. This language is positively accepted by a deterministic Büchi tree automaton (that goes in a final state whenever an a is read and in a non-final state otherwise) and hence is a positive tree language.

***Remark* 32.** *The complement of a language qualitatively accepted by a* deterministic *tree automaton $\mathcal{A}$ is positively accepted by a deterministic tree automaton $\mathcal{B}$. Indeed, it suffices to define $\mathcal{B}$ starting from $\mathcal{A}$ and dualise its acceptance condition (namely increment the value of the colouring function by 1).*

Let us now give two examples of languages that are not positive tree languages.

Proposition 15 [CHS11]

The language $\mathcal{L}_a$ of $\{a, b\}$-labelled trees whose set of branches containing an a has measure 1 and the language $\mathcal{L}_{a \wedge a}$ of $\{a, b\}$-labelled trees containing an a in both their left and right subtrees are not positive tree languages.

More generally, we have the following incomparability results.

Proposition 16 [CHS11]

One has the following incomparability results.

- The classes of positive and qualitative tree languages are incomparable.
- The classes of positive and regular tree languages are incomparable.

The most notable difference between positive and qualitative tree languages is that positive tree languages are not closed under intersection.

Proposition 17 [CHS11]

The class of positive tree languages is closed under union but neither under intersection nor complementation.

However, positive tree languages enjoy all decidability properties of qualitative languages presented in Section 5.5.

Theorem 44 [CHS11]

The following holds for any automaton $\mathcal{A}$ and any tree t.

- The tree t belongs to $L_{\text{Pos}}(\mathcal{A})$ if and only if Éloïse positively wins in $\mathbb{G}_{\mathcal{A},t}$.
- The language $L_{\text{Pos}}(\mathcal{A})$ is non empty if and only if Éloïse positively wins in $\mathbb{G}_{\mathcal{A}}$ from q_{ini}.
- One can decide whether $L_{\text{Pos}}(\mathcal{A}) = \varnothing$ in polynomial time. Moreover, if $L_{\text{Pos}}(\mathcal{A}) \neq \varnothing$, it contains a regular tree, and such a tree can be constructed in polynomial time.

6 Imperfect Information Stochastic Games

In order to check the emptiness of several classe of automata considered in the rest of this chapter we will use imperfect information games. In order to treat them in a unified setting we now consider a quite general class of games: *imperfect-information stochastic games*. In a nutshell those are finite state games in which, at each round, the two players choose concurrently an action and based on these actions the successor state is chosen according to some fixed probability distribution. The resulting infinite play is won by Éloïse if it satisfies a given *objective* (*e.g.* reachability, safety, Büchi or co-Büchi). Imperfect information is modelled as follows: both players have an equivalence relation over states and, instead of observing the exact state, they only see to which equivalence class it belongs to. Therefore, if two partial plays are indistinguishable by some player (*i.e.* they give rise to the same sequence of equivalence classes for this player), he should behave the same in both of them.

After giving the definitions together with some examples we study decidability of various problems in two different settings: one where the players are allowed to use randomised strategy and another one where they are forced to use deterministic strategies.

6.1 Definitions

6.1.1 Concurrent Arenas

A ***concurrent arena with imperfect information*** is a tuple $\mathcal{G} = \langle S, \Sigma_E, \Sigma_A, \delta, \sim_E, \sim_A \rangle$ where

- S is a finite set of control states;
- Σ_E (*resp.* Σ_A) is the (finite) set of actions for Éloïse (*resp.* Abélard);
- $\delta : S \times \Sigma_E \times \Sigma_A \to \mathcal{D}(S)$ is the transition (total) function;
- $\sim_E$ and $\sim_A$ are two equivalence relations over states.

A play in a such an arena proceeds as follows. First it starts in some initial state s. Then the first player, Éloïse, picks an action $\sigma_E \in \Sigma_E$ and, *simultaneously* and *independently*, the second player, Abélard, chooses an action $\sigma_A \in \Sigma_A$. Then a successor state is chosen accordingly to the probability distribution $\delta(s, \sigma_E, \sigma_A)$ and process restarts: the players choose a new pair of actions that induces, together with the current state, a new state and so on forever. Hence a ***play*** is an infinite sequence $s_0(\sigma_{E,0}, \sigma_{A,0})s_1(\sigma_{E,1}, \sigma_{A,1})s_2(\sigma_{E,2}, \sigma_{A,2}) \cdots$ in $(S \cdot (\Sigma_E \times \Sigma_A))^\omega$ such that for every $i \geqslant 0$, $\delta(s_i, \sigma_{E,i}, \sigma_{A,i})(s_{i+1}) > 0$. In the sequel we refer to a prefix of a play (ending in a state) as a ***partial play*** and we denote by $Plays(\mathcal{G})$ the set of all plays in arena $\mathcal{G}$.

The intuitive meaning of $\sim_E$ (*resp.* $\sim_A$) is that two states s_1 and s_2 such that $s_1 \sim_E s_2$ (*resp.* $s_1 \sim_A s_2$) cannot be distinguished by Éloïse (*resp.* by Abélard). We easily extend the relation $\sim_E$ to partial plays (here Eloise observes her actions, but does not observe Adam's actions): let $\lambda = s_0(\sigma_{E,0}, \sigma_{A,0})s_1(\sigma_{E,1}, \sigma_{A,1}) \cdots s_n$ and $\lambda' = s'_0(\sigma'_{E,0}, \sigma'_{A,0})s'_1(\sigma'_{E,1}, \sigma'_{A,1}) \cdots s'_n$ be two partial plays, then $\lambda \sim_E \lambda'$ if and only if $s_i \sim_E s'_i$ and $\sigma_{E,i} = \sigma'_{E,i}$ for all $i = 0, \cdots, n$.

Perfect information concurrent arenas (in the sense of [AHK07; AH00]) correspond to the special case where $\sim_E$ and $\sim_A$ are the equality relation over S. More generally, we say that Éloïse (resp. Abélard) has ***perfect information*** when $\sim_E$ (resp. $\sim_A$) is the equality relation, and we say that Abélard is ***more informed*** than Éloïse if $\sim_A$ refines $\sim_E$ (*i.e.* $s \sim_E t$ implies $s \sim_A t$).

Example 30

As a first example, consider the arena depicted in Figure 39. In this example we have $S = \{s_0, s_1, s_2, s_3, s_4, f\}$ (state f is double-circled meaning that it is final) and $\Sigma_E = \Sigma_A = \{a, b\}$. From s_0 if Abélard plays the action a then any action played by Éloïse leads with probability $\frac{1}{2}$ either to s_1 or s_2 (we use symbol $*$ in figures to indicate any possible action for some player). More formally, we have for any $x \in \{a, b\}$ that $\delta(s_0, x, a) = d$ where $d(s_1) = d(s_2) = \frac{1}{2}$. Similarly if Abélard plays b then any action played by Éloïse leads with probability $\frac{1}{2}$ either to s_3 or s_4. The equivalence relation $\sim_E$ has 3 equivalence classes : $\{s_0\}$, $\{s_1, s_2, s_3, s_4\}$ and $\{f\}$, and the equivalence relation $\sim_A$ is the equality relation (*i.e.* Abélard has perfect information). In the states s_1, s_2, s_3 and s_4, which are indistinguishable by Éloïse, the action of Abélard has no impact. In the figure, we represent the fact that $s_1 \sim_E s_2 \sim_E s_3 \sim_E s_4$ by putting them into a dashed box. If Éloïse plays a from s_1 or s_4 or b from s_2 or s_3 the play goes to the final state f which is a sink state. Any other action by Éloïse from one of those states leaves the current state unchanged.

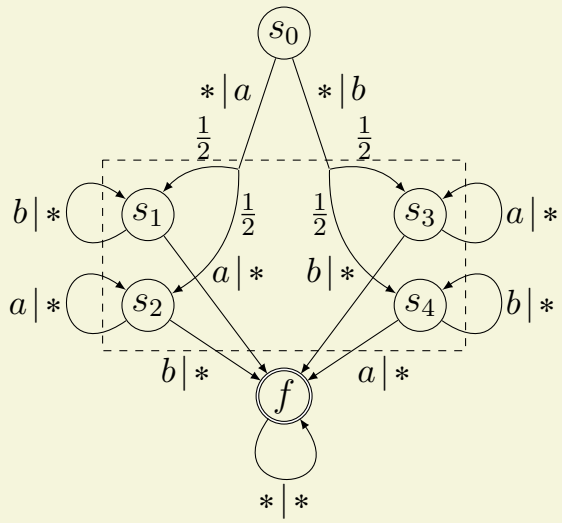


Figure 39 – Reachability game of Example 30

6.1.2 Strategies

In order to choose their moves the players follow strategies, and, for this, they may use all the information they have about what was played so far. However, if two partial plays are equivalent for $\sim_E$, then Éloïse cannot distinguish them, and should therefore behave the same. This leads to the following notion.

An ***(observation-based) randomised strategy*** for Éloïse is a function $\varphi_E : (S/_{\sim_E}) \cdot (\Sigma_E \cdot S/_{\sim_E})^* \to \mathcal{D}(\Sigma_E)$, *i.e.*, choosing her next action based on the sequence of observations/actions she has seen/played so far. In particular, a strategy φ_E is such that $\varphi_E(\lambda) = \varphi_E(\lambda')$ whenever $\lambda \sim_E \lambda'$. Observation-based strategies for Abélard are defined similarly.

An ***(observation-based) pure strategy*** is a randomised strategy φ such that whenever defined $\varphi(\lambda)$ is a probability distribution whose support is a singleton. In other word, one can think of a pure strategy for Éloïse (resp. Abélard) as a function taking its value in Σ_E (resp. Σ_A).

A ***finite-memory strategy*** for Éloïse with memory M (M being a finite set) is some triple $\varphi = (Move, Up, m_0)$ where $m_0 \in M$ is the initial memory, $Move : M \to \mathcal{D}(\Sigma_E)$ associates a distribution of actions with any element in the memory M and $Up : M \times S/_{\sim_E} \times \Sigma_E \to M$ is a mapping updating the memory with respect to some observation (and the last action played by Éloïse). One defines $\varphi(s_0) = Move(m_0)$ and, for any $n \geqslant 1$,

$$\varphi(s_0(\sigma_{E,0}, \sigma_{A,0}) \cdots s_n) = Move(Up(\cdots Up(Up(m_0, [s_1]/_{\sim_E}, \sigma_{E,0}), [s_2]/_{\sim_E}, \sigma_{E,1}), \cdots, [s_n]/_{\sim_E}, \sigma_{E,n-1}) \cdots)$$

Hence, a finite-memory strategy is an observation-based strategy that can be implemented by a finite transducer whose set of control states is M.

6.1.3 Probability Space and Outcomes of Strategies

Let $\mathcal{G} = \langle S, \Sigma_E, \Sigma_A, \delta, \sim_E, \sim_A \rangle$ be a concurrent arena with imperfect information, let $s_0 \in S$ be an initial state, φ_E be a strategy for Éloïse and φ_A be a strategy for Abélard. In the sequel we are interested in defining the probability of a (measurable) set of plays knowing that Éloïse (*resp.* Abélard) plays accordingly φ_E (*resp.* φ_A). This is done in the classical way: first one defines the probability measure for basic sets of plays (called here *cones* and corresponding to plays having some initial common prefix) and then extends it in a unique way to all measurable sets.

First define $Outcomes(s_0, \varphi_E, \varphi_A)$ to be the set of all possible plays when the game starts on s_0 and when Éloïse and Abélard plays respectively accordingly to φ_E and φ_A. More formally, an infinite play $\lambda = s_0(\sigma_{E,0}, \sigma_{A,0})s_1(\sigma_{E,1}, \sigma_{A,1})s_2 \cdots$ belongs to $Outcomes(s_0, \varphi_E, \varphi_A)$ if and only if, for every $i \geqslant 0$, $\varphi_E(s_0\sigma_{E,0}s_1\sigma_{E,1} \cdots s_i)(\sigma_{E,i}) > 0$ and $\varphi_A(s_0\sigma_{A,0}s_1\sigma_{A,1} \cdots s_i)(\sigma_{A,i}) > 0$ (*i.e.* $\sigma_{X,i}$ is possible accordingly to φ_X, for $X = E, A$).

Now, for any partial play λ, the ***cone*** for λ is the set $\text{Cone}(\lambda) = \lambda \cdot ((\Sigma_E \times \Sigma_A) \cdot S)^\omega$ of all infinite plays with prefix λ. Denote by $Cones$ the set of all possible cones and let $\mathcal{F}$ be the Borel σ-field generated by $Cones$ considered as a set of basic open sets (*i.e.* $\mathcal{F}$ is the smallest set containing $Cones$ and closed under complementation and countable union). Then $(Plays(\mathcal{G}), \mathcal{F})$ is a σ-algebra.

A pair of strategies (φ_E, φ_A) induces a probability space over $(Plays(\mathcal{G}), \mathcal{F})$. Indeed one can define a measure $\mu_{s_0}^{\varphi_E,\varphi_A} : Cones \to [0, 1]$ on cones (this task is easy as a cone is uniquely defined by a finite partial play) and then uniquely extend it to a probability measure on $\mathcal{F}$ using the Carathéodory's unique extension theorem. For this, one defines $\mu_{s_0}^{\varphi_E,\varphi_A}$ inductively on cones:

- $\mu_{s_0}^{\varphi_E,\varphi_A}(\text{Cone}(s)) = 1$ if $s = s_0$ and $\mu_{s_0}^{\varphi_E,\varphi_A}(s) = 0$ otherwise.
- For every partial play λ ending in some vertex s,

$$\mu_{s_0}^{\varphi_E,\varphi_A}(\text{Cone}(\lambda \cdot (\sigma_E, \sigma_A) \cdot s')) = \mu_{s_0}^{\varphi_E,\varphi_A}(\text{Cone}(\lambda)).\varphi_E(\lambda)(\sigma_E).\varphi_A(\lambda)(\sigma_A).\delta(s, \sigma_E, \sigma_A)(s')$$

Denote by $\Pr_{s_0}^{\varphi_E,\varphi_A}$ the unique extension of $\mu_{s_0}^{\varphi_E,\varphi_A}$ to a probability measure on $\mathcal{F}$. Then $(Plays(\mathcal{G}), \mathcal{F}, \Pr_{s_0}^{\varphi_E,\varphi_A})$ is a probability space.

6.1.4 Objectives, Value of a Game

Fix a concurrent arena with imperfect information $\mathcal{G}$. An objective for Éloïse is a measurable[7] set $\mathcal{O} \subseteq Plays(\mathcal{G})$: a play is won by her if it belongs to $\mathcal{O}$; otherwise it is won by Abélard. A ***concurrent game with imperfect information*** is a triple $(\mathcal{G}, s_0, \mathcal{O})$ where $\mathcal{G}$ is a concurrent arena with imperfect information, s_0 is an initial state and $\mathcal{O}$ is an objective. In the sequel we focus on the following special classes of objectives (note that all of them are Borel sets hence measurable for all measures $\Pr_{s_0}^{\varphi_E,\varphi_A}$) that we define as means of a subset $F \subseteq S$ of ***final states***.

- ***Reachability objective***: a play is winning if it eventually goes through some final state.
- ***Safety objective***: a play is winning if it never goes through a final state.
- ***Büchi objective***: a play is winning if it goes infinitely often through final states.
- ***Co-Büchi objective***: a play is winning if it goes finitely often through final states.

***Remark* 33.** *We do not consider here the parity objective as the problem we will tackle are already undecidable for simpler objectives.*

A reachability (*resp.* safety, Büchi, co-Büchi) game is a game equipped with a reachability (*resp.* safety, Büchi, co-Büchi) objective. In the sequel we may replace $\mathcal{O}$ by F when it is clear from the context which winning condition we consider.

Fix a concurrent game with imperfect information $\mathbb{G} = (\mathcal{G}, s_0, \mathcal{O})$. A strategy φ_E for Éloïse is ***almost-surely winning*** if, for any counter-strategy φ_A for Abélard, $\Pr_{s_0}^{\varphi_E,\varphi_A}(\mathcal{O}) = 1$. If such

[7] Actually, as the mesure is defined with respect to a pair of strategies (φ_E, φ_A) the objective has to be mesurable for all measures $\Pr_{s_0}^{\varphi_E,\varphi_A}$ when φ_E (resp. φ_A) is ranging over Éloïse's (resp. Abélard's) strategies.

a strategy exists, we say that Éloïse ***almost-surely wins*** $\mathbb{G}$. A strategy φ_E for Éloïse is ***positively winning*** if, for any counter-strategy φ_A for Abélard, $\Pr_{s_0}^{\varphi_E,\varphi_A}(\mathcal{O}) > 0$. If such a strategy exists, we say that Éloïse ***positively wins*** $\mathbb{G}$.

6.1.5 Examples

Let us start with a famous example of a concurrent perfect-information safety game known as *Hide-or-Run* (originally introduced by Everett [Eve57] and later adapted in [KS81] and [AHK07]).

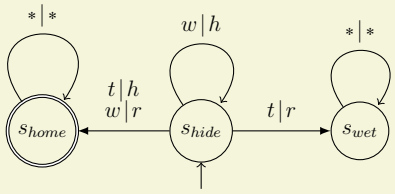


Figure 40 – The game *Hide-or-Run*: $\mathbb{G}_{HR}$

Example 31 (Hide-or-Run)

In the Hide-or-Run game (see Figure 40) Abélard wants to reach a target state s_{home} and the interesting part of the game happens at state s_{hide}. At this state, Abélard is hiding behind a small hill, while Éloïse is trying to hit him with a snowball. Abélard can choose between hiding (h) and running (r), and Éloïse can choose between waiting (w) and throwing (t) her only snowball. If Abélard hides and Éloïse waits, the game stays in state s_{hide}. If Abélard runs and Éloïse throws the snowball, then Abélard is hit, and the game proceeds to sink state s_{wet}. In all other cases, Abélard gets home (either he runs without being hit or he can safely run after Éloïse has thrown her snowball) and the game proceeds to sink state s_{home}. This is modelled easily as a safety game where Éloïse wants to prevent visiting s_{home}.

Obviously, Éloïse has no *pure* positively winning strategy. Indeed, fix a strategy for Éloïse. Either it plays w or t in the first move: a counter strategy for Abélard consists in the first case to play r, in the second case to play h.

If one considers now randomised strategies, a strategy for Éloïse is a sequence $(p_i)_{i\geqslant 1}$ of probabilities ($0 \leqslant p_i \leqslant 1$ for all $i \geqslant 1$): p_i is the probability that she waits at round i if the game is still in s_{hide} (in all other cases her move is not important). It is then easy to check that such a strategy is positively winning for Éloïse if and only if $0 < p_i < 1$ for all i and $\prod_i p_i > 0$ (hence there are p_i's arbitrary close to 1). Intuitively, there should be a positive chance that she never throws the ball, which forces Abélard to eventually run with positive probability at some stage i; but at stage i the probability that Éloïse still has her snowball and throws it is positive. Note that all this remains true even if Éloïse is blind in this game (*i.e.* $s_{hide} \sim s_{wet} \sim s_{wait}$).

Example 32 (Pure strategies *vs* determinacy)

Consider the (perfect information) concurrent reachability game depicted in Figure 41 with q_f as unique final state. In state q_w, if both players choose the same action then they stay in state q_w and otherwise they move to state q_f. In state q_f, all choices of actions stay in state q_f.

Éloïse does not have any almost-surely pure winning strategy. Indeed, given any *pure* strategy φ_E for Éloïse, the counter-strategy φ_A for Abélard mirroring the strategy of Éloïse (*i.e.* $\varphi_A = \varphi_E$) only allows for the play q_w^ω and hence, $\Pr_{s_0}^{\varphi_E,\varphi_A}(\mathcal{O}) = 0$. Similarly Abélard does not have an almost-surely pure winning strategy. For any fixed pure strategy φ_A of Abélard, any counter-strategy φ_E for Éloïse that satisfies $\varphi_E(q_w) \neq \varphi_A(q_w)$ is such that $\Pr_{s_0}^{\varphi_E,\varphi_A}(\mathcal{O}) = 1$.

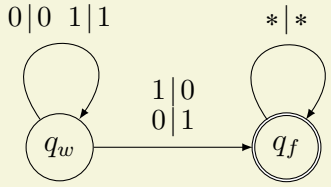


Figure 41 – Reachability game of Example 32

***Remark* 34.** *The situation in Example 32 contrasts with the case of perfect information non-concurrent ω-regular games which are* determined*: from any state one of the players has a* surely *winning strategy (see* e.g. *[Zie98; GTW02]). In the perfect information setting (even with concurrency and stochastic transition function) there is also a determinacy result, when allowing randomised strategies, using the notion of values (see* e.g. *[AH00] for ω-regular objectives). In the imperfect information setting, if one allows randomisation in strategies, one has, for Büchi conditions, a determinacy result (called* qualitative determinacy *in [BGG09]): either Éloïse has an almost-surely winning strategy or Abélard has a positively winning strategy (in Example 32, the randomised strategy for Éloïse consisting in playing* 0 *and* 1 *with equal probability in any state is almost-surely winning). Other qualitative determinacy results were previously obtained in the perfect information setting [AHK07; AH00; Hor08].*

6.2 Decision Problems in the Randomised Strategies Setting

In collaboration with Vincent Gripon, we obtained the following result. A similar result was independently obtained by Bertrand *et al.* [BGG09].

Theorem 45 [GS09; BGG09]

Deciding whether Éloïse almost-surely wins (possibly using a randomised strategy) a concurrent *Büchi* game with imperfect information is a 2-ExpTime-complete problem.

The roadmap to prove Theorem 45's upper bound is the following. One first proves that if Éloïse has an almost surely winning strategy then this latter strategy requires at most an exponential size memory. Then one exhaustively tries all exponential size strategies hence, checks whether Abélard has a positively winning strategy in the $1\frac{1}{2}$-player co-Büchi game obtained by fixing the Éloïse's strategy (she is no longer "playing" and we see this game as one where Abélard plays alone a co-Büchi game). As one can prove that positively winning in such a game requires at most an exponential-size memory, we can check whether the original strategy of Éloïse is almost surely winning in doubly exponential time (indeed, note that the $1\frac{1}{2}$-player co-Büchi game we built is of exponential size). As there are doubly-exponentially many possible strategies of Éloïse to check, this leads to the desired upper bound.

The proof of the lower bound is a tricky generalisation of a similar result given in [CDHR07] showing ExpTime-hardness of concurrent games where *only* Éloïse is imperfectly informed. Note that the lower bound already holds if the game is equipped with a *reachability* objective. The idea is to simulate an alternating exponential space Turing machine (without input). We design a game where the players describe the run of such a machine: transitions from existential (resp. universal) states are chosen by Éloïse (resp. Abélard) and Abélard is also in charge of describing the successive configurations of the machine. To prevent him from cheating, Éloïse can secretly mark a cell of the tape, and later check whether it was correctly updated (if not, she wins). As she cannot store the exact index of the cell (it is of exponential size), she could cheat in the previous phase: hence Abélard secretly marks some bit and one recalls the value of the corresponding bit of the index of the marked cell: this bit is checked when Éloïse claims that Abélard cheated (if it is wrong then she is loosing). Éloïse also wins if the described run is accepting. Éloïse can also restart the computation whenever she wants (this is useful when she cannot prove that Abélard cheated): hence if the machine accepts the only option for Abélard is to cheat, and Éloïse will eventually catch him with probability one. Now if the machine does not accept, the only option for Éloïse is to cheat, but it will be detected with positive probability.

Concerning positive winning, the following results were proved in [BGG09].

Theorem 46 [BGG09]

Deciding whether Éloïse positively wins (possibly using a randomised strategy) a concurrent *Reachabilty* game with imperfect information is an ExpTime-complete problem.

Deciding whether Éloïse positively wins (possibly using a randomised strategy) a concurrent *co-Büchi* game with imperfect information is a 2-ExpTime-complete problem.

On the negative side note the following result that is a direct consequence of a combination of a similar undecidability result on probabilistic automata on infinite words [BGB12] with the fact that randomisation is useless in $1\frac{1}{2}$-player games with imperfect information [CDGH10].

Theorem 47 [BGB12; CDGH10]

Deciding whether Éloïse almost-surely wins (possibly using a randomised strategy) a concurrent *co-Büchi* game with imperfect information is undecidable.

Deciding whether Éloïse positively wins (possibly using a randomised strategy) a concurrent *Büchi* game with imperfect information is undecidable.

We terminate this Section with Table 1 that summarises the (un)decidability results on games with imperfect information.

6.3 Decision Problems in the Pure Strategies Setting

In collaboration with Arnaud Carayol and Christof Löding we considered the case where we assume that Éloïse is only allowed to use *pure* strategies. In addition to the natural intrinsic interest of this problem we give an application in Section 7.3 by showing how to reduce the emptiness problem for *alternating* qualitative tree automata to this question.

***Remark* 35.** *These results overlap with the ones* independently *obtained by Chatterjee and Doyen in [CD12]. A major difference is that the results obtained in [CD12] only concern the case where Abélard*

	Does Éloïse positively win?	Does Éloïse almost surely win?
Reachability	ExpTime-complete (Theorem 46)	2-ExpTime-complete (Theorem 45)
Safety	2-ExpTime-complete (Theorem 46)	ExpTime-complete (Theorem 46)
Büchi	Undecidable (Theorem 47)	2-ExpTime-complete (Theorem 46)
Co-Büchi	2-ExpTime-complete (Theorem 46)	Undecidable (Theorem 47)

Table 1 – Complexity of decision problems in the randomised strategy setting when both players are uniformed. Note that when Abélard is more informed than Éloïse complexity goes down from 2-ExpTime to ExpTime [BGG09; CDHR07].

is fully informed *while our results only assume that he is more informed than Éloïse. On the decidability side and when Abélard is fully informed, the results obtained in [CD12] are similar to the ones presented here but the proofs are quite different. On the undecidability side, the only result mentioned in [CD12] is the one that directly follows from [BGB12] (positive Büchi); in particular [CD12] does not contain the undecidability result for positive safety.*

Also note that even if we obtained the results in [CLS12] since 2011 they have not yet been published. However a research report can be obtained under request.

On the positive side, if we assume that Abélard has perfect information, one obtains the following decidability result.

Theorem 48 [CLS12; CD12]

Deciding whether Éloïse almost-surely wins using a *pure* strategy a concurrent *Büchi* game where Abélard is perfectly informed while Éloïse has imperfect information is an ExpTime-complete problem.

The proof of Theorem 48 we gave in [CLS12] can be refined to obtain a stronger decidability result where we only need to assume that Abélard is more informed than Éloïse

Theorem 49 [CLS12]

Deciding whether Éloïse almost-surely wins using a *pure* strategy a concurrent *Büchi* game with imperfect information where Abélard is *more informed* than Éloïse is a 2-ExpTime problem.

The proof of both Theorem 48 and Theorem 49 actually rely on the following result that deals with positive winning.

Theorem 50 [CLS12]

Deciding whether Éloïse positively wins using a *pure* strategy a concurrent *reachability* game with imperfect information where Abélard is *perfectly informed* is an ExpTime-complete problem.

Deciding whether Éloïse positively wins using a *pure* strategy a concurrent *reachability* game with imperfect information where Abélard is *more informed* than Éloïse is a 2-ExpTime problem.

On the negative side, it trivially follows from the work of Baier *et al.* on probabilistic ω-word automata [BGB12] that, even when Abélard is perfectly informed, positive (resp. almost-sure) winning in a Büchi (resp. co-Büchi) game is undecidable.

Theorem 51 [BGB12]

The following problems are undecidable:

- Does Éloïse positively win using a *pure* strategy a concurrent *Büchi* game with imperfect information where Abélard is perfectly informed while Éloïse has imperfect information?
- Does Éloïse almost-surely win using a *pure* strategy a concurrent *co-Büchi* game with imperfect information where Abélard is perfectly informed while Éloïse has imperfect information?

One can actually prove a stronger result by a reduction to the value-1 problem for ω-word automata which is known to be undecidable [GO10; Col09].

Theorem 52 [CLS12]

The following problem is undecidable:

- Does Éloïse positively win using a *pure* strategy a concurrent *safety* game with imperfect information where Abélard is perfectly informed while Éloïse has imperfect information?

We terminate this Section with Table 2 that summarises the (un)decidability results on games with imperfect information.

7 Emptiness Of Alternating Tree Automata Using Games With Imperfect Information

We now present results obtained together with Nathanaël Fijalkow and Sophie Pinchinat [FPS13]. It consists of an alternative technique to solve the emptiness problem of alternating tree automata, by directly reducing it to a two-player game with *imperfect information*. This does not lead to a gain in complexity due to intrinsic hardness, but unravels the two key ingredients: the first one is the positional determinacy of parity games, to prove the correctness of the reduction, and the second is the determinisation property of ω-word automata, to solve the obtained two-player imperfect information game.

	Does Éloïse positively win?	Does Éloïse almost surely win?
Reachability	ExpTime-complete (Theorem 50) 2-ExpTime (Theorem 50)	ExpTime-complete (Theorem 48) 2-ExpTime (Theorem 49)
Safety	Undecidable (Theorem 52)	ExpTime-complete (Theorem 48) 2-ExpTime (Theorem 49)
Büchi	Undecidable (Theorem 51)	ExpTime-complete (Theorem 48) 2-ExpTime (Theorem 49)
Co-Büchi	Undecidable (Theorem 52)	Undecidable (Theorem 51)

Table 2 – Complexity of decision problems in the pure strategy setting when Abélard is perfectly informed (first line) / more informed than Éloïse (second line).

Our technique is of interest for at least two reasons: (i) it pushes the algorithmic difficulty to the game solving part, for which antichains representations have recently been developed [BCWDHR09], hence could lead to efficient algorithms, and (ii) a "Simulation Theorem"-free technique is required for classes of tree automata for which no (effective) Simulation Theorem exists.

We illustrated this latter situation by considering an alternation extension of the class of qualitative tree automata and we applied our technique to check the emptiness of an alternating qualitative Büchi tree automaton. Furthermore, we observed that the emptiness problem becomes undecidable for the co-Büchi condition, implying that there is no simulation theorem for alternating qualitative tree co-Büchi automata.

For our technique to go through, the key ingredient is a positionality result for stochastic Büchi games over *infinite* arenas.

7.1 Definitions

We start with some definitions. We first give in Section 7.1.1 a stochastic version of turn-based games on graph that permits in Section 7.1.2 to define a qualitative version of *alternating* parity tree automata; finally in Section 7.1.3 we introduce a special class of imperfect information games that is later used to design an emptiness game for alternating parity tree automata.

7.1.1 Stochastic Turn-Based Games

We now introduce a class of games that (i) generalises the class of two-player perfect information turn-based games on graphs (see Chapter 1 Section 4) and (ii) is easily seen to be captured by the class of perfect-information concurrent games as defined in Section 6. The reason why we introduce this class is because it will "coincide" with the class of alternating parity qualitative tree automata that will be defined later.

A (turn-based) ***stochastic arena*** is a tuple $\mathcal{G} = (G, V_E, V_A, V_R, \delta)$ where $G = (V, E)$ is a graph, $V = V_E \uplus V_A \uplus V_R$ is a partition of the vertices among two players, Éloïse and Abélard, and an extra player Random, and $\delta : V_R \to \mathcal{D}(V)$ is a map such that $\mathrm{Supp}(\delta(v)) = E(v)$ for all $v \in V_R$. In a vertex $v \in V_E$ (*resp.* $v \in V_A$) Éloïse (*resp.* Abélard) chooses a successor vertex from $E(v)$ and in a random vertex $v \in V_R$, a successor vertex is chosen according to the probability distribution $\delta(v)$.

A (pure) ***strategy***[8] for Éloïse is a function $\varphi_E : V^* \cdot V_E \to V$ such that for every $\lambda \cdot v \in V^* \cdot V_E$ one has $\varphi_E(\lambda \cdot v) \in E(v)$. Strategies of Abélard are defined likewise, and usually denoted φ_A.

It should be fairly clear how, using similar definitions as we did for the more general setting of concurrent games in Section 6, how to define, given an initial vertex $v_0 \in V$, a strategy φ_E for Éloïse and a strategy φ_A for Abélard, a probability measure $\Pr_{v_0}^{\varphi_E,\varphi_A}$ on the σ-algebra of plays starting from v_0 where Éloïse (resp. Abélard) follows φ_E (resp. φ_A).

An ***Objective*** $\Omega \subseteq V^\omega$ is again defined as a (measurable) subset of plays and a (two-player perfect-information) ***stochastic game*** is a pair $\mathbb{G} = (\mathcal{G}, \Omega)$. A game is ***deterministic*** whenever $V_R = \varnothing$ (and in this case we omit both V_R and δ in the notations hence, retrieving the setting of Chapter 1 Section 4).

A strategy φ_E for Éloïse is ***surely winning*** from a given vertex v_0 if the set of plays starting from v_0 where Éloïse respects φ_E is included in Ω; it is ***almost-surely winning*** from v_0 if $\Pr_{v_0}^{\varphi_E,\varphi_A}(\Omega) = 1$ for every strategy φ_A of Abélard. Similar notions for Abélard are defined dually.

Again, a positional strategy φ is one that does not require memory, *i.e.* such that for any two partial plays of the form $\lambda \cdot v$ and $\lambda' \cdot v$, one has $\varphi(\lambda \cdot v) = \varphi(\lambda' \cdot v)$, equivalently φ only depends on the current vertex. It is well-known that positional strategies suffice to surely win in *deterministic* parity games (see Theorem 1) while for stochastic games the following result is well-known (see *e.g.* [GZ07] for a slightly more general result).

Theorem 53 (Positional Determinacy)

Let $\mathbb{G}$ be a stochastic parity game played on a *finite* arena and let v_0 be some initial vertex. If Éloïse almost-surely wins from v_0 then she can do so using a positional strategy.

To the best of our knowledge, no extension of this result is known when dropping the assumption that the arena is finite. We give such an extension (for Büchi games on so-called chronological arenas of finite out-degree) in Theorem 55.

7.1.2 Alternating Parity Tree Automata

An ***alternating parity tree automaton*** is a tuple $\mathcal{A} = \langle A, Q_\exists, Q_\forall, q_{\text{ini}}, \Delta, \mathrm{Col}\rangle$, where A is a the input alphabet, $Q_\exists$ is a set of existential states and $Q_\forall$ is a set of universal states such that $Q_\exists$ and $Q_\forall$ are disjoint (we let $Q = Q_\exists \uplus Q_\forall$), $q_{\text{ini}} \in Q$ is an initial state, $\Delta \subseteq Q \times A \times Q \times Q$ is a (finite) transition relation and $\mathrm{Col} : Q \to \mathbb{N}$ is a colouring function. We additionally assume without loss of generality that for all $(q, a) \in Q \times A$ there is at least one $(q, a, q_0, q_1) \in \Delta$.

Note that non-deterministic parity tree automata as defined in Section 2.2 coincide with alternating parity tree automata in which all states are existential.

We use tree automata as acceptors for tree languages, and now acceptance is defined by means of a perfect-information (possibly stochastic) turn-based parity game. We will define two semantics for acceptance – *classical* and *qualitative* – and when considering non-deterministic versions of alternating tree automata we will get back to the notions of regular (resp. qualitative) tree languages as defined in Section 2.2 (resp. Section 5).

[8] We do not consider randomised strategies as pure strategies are the right model here.

Fix an alternating parity tree automaton $\mathcal{A} = \langle A, Q_\exists, Q_\forall, q_{\text{ini}}, \Delta, \text{Col}\rangle$ and an A-labelled tree t. From $\mathcal{A}$ and t, we define two games: $\mathbb{G}_{\mathcal{A},t}$ and $\mathbb{G}^{=1}_{\mathcal{A},t}$.

Intuitively, a play in those two games consists in moving a pebble along a branch of t in a top-down manner: the pebble is attached to a state and in a node u with state q Éloïse (if $q \in Q_\exists$) or Abélard (if $q \in Q_\forall$) picks a transition $(q, t(u), q_0, q_1) \in \Delta$, and then *Abélard* (in $\mathbb{G}_{\mathcal{A},t}$) or *Random* (in $\mathbb{G}^{=1}_{\mathcal{A},t}$) chooses to move down the pebble either to $u0$ (and update the state to q_0) or to $u1$ (and update the state to q_1).

Formally, one let $G = (V_\exists \uplus V_\forall \uplus V_\Delta, E)$ with $V_\exists = \{0,1\}^* \times Q_\exists$, $V_\forall = \{0,1\}^* \times Q_\forall$ and $V_\Delta = \{(u, q, q_0, q_1) \mid u \in \{0,1\}^* \text{ and } (q, t(u), q_0, q_1) \in \Delta\}$ and

$$\begin{aligned} E \quad = \quad & \{((u,q),(u,q,q_0,q_1)) \mid (u,q,q_0,q_1) \in V_\Delta)\} \quad \cup \\ & \{((u,q,q_0,q_1),(u\cdot x, q_x)) \mid x \in \{0,1\} \text{ and } (u,q,q_0,q_1) \in V_\Delta)\} \end{aligned}$$

Then let $\mathcal{G}_{\mathcal{A},t} = (G, V_E, V_A)$ be the deterministic arena defined by letting $V_E = V_\exists$ and $V_A = V_\forall \cup V_\Delta$ and let $\mathcal{G}^{=1}_{\mathcal{A},t} = (G, V_E, V_A, V_R, \delta)$ be the stochastic arena defined by letting $V_E = V_\exists$, $V_A = V_\forall$, $V_R = V_\Delta$ and $\delta((u,q,q_0,q_1))$ be the distribution $(u\cdot 0, q_0) \mapsto \frac{1}{2}$ and $(u\cdot 1, q_1) \mapsto \frac{1}{2}$.

Extend Col on V by letting $\text{Col}((u,q)) = \text{Col}((u,q,q_0,q_1)) = \text{Col}(q)$. Finally one let $\mathbb{G}_{\mathcal{A},t} = (\mathcal{G}_{\mathcal{A},t}, \text{Col})$ and $\mathbb{G}^{=1}_{\mathcal{A},t} = (\mathcal{G}^{=1}_{\mathcal{A},t}, \text{Col})$.

A tree t is ***accepted*** (*resp.* ***qualitatively accepted***) by $\mathcal{A}$ if Éloïse has a surely (*resp.* almost-surely) winning strategy from $(\varepsilon, q_{\text{ini}})$ in the game $\mathbb{G}_{\mathcal{A},t}$ (*resp.* $\mathbb{G}^{=1}_{\mathcal{A},t}$). Finally, we define the set $L(\mathcal{A})$ as the set of trees accepted by $\mathcal{A}$ and the set $L^{=1}(\mathcal{A})$ as the set of trees qualitatively accepted by $\mathcal{A}$.

***Remark* 36.** *In the case of non-deterministic automata (*i.e. *alternating automata in which all states are existential) the (acceptance) game $\mathbb{G}_{\mathcal{A},t}$ (resp. $\mathbb{G}^{=1}_{\mathcal{A},t}$) coincides with the acceptance game defined in Section 2.3 (resp. Section 5.5).*

***Remark* 37.** *Those tree languages accepted by alternating tree automata coincide with regular tree languages (*i.e. *alternation does not give extra expressive power to tree automata). See Theorem 54 below.*

Example 33

An example of a language $L^{=1}(\mathcal{A}_\mathcal{B})$ with $\mathcal{A}_\mathcal{B}$ being alternating is the set of trees such that all subtrees belongs to some $L^{=1}(\mathcal{B})$ where $\mathcal{B}$ is non-deterministic.

***Remark* 38.** *There are several definitions of alternating tree automata, and another popular one is by not distinguishing between existential and universal states but replacing the transition relation by a map $\delta : Q \times A \to \mathcal{B}^+(Q \times \{0,1\})$ where $\mathcal{B}^+(X)$ denotes the positive Boolean formulas over X (see* e.g. *[KVW00]). Our model is easily seen to be equi-expressive with that one.*

***Remark* 39.** *Any* positional *strategy for Éloïse in $\mathbb{G}_{\mathcal{A},t}$ or $\mathbb{G}^{=1}_{\mathcal{A},t}$ can be described as a function $\varphi : \{0,1\}^* \times Q_\exists \to Q \times Q$ that satisfies the following property: $\forall n \in \{0,1\}^*$, if $\varphi(n,q) = (q_0,q_1)$ then $(q, t(n), q_0, q_1) \in \Delta$. Equivalently, in a curryfied form, φ is a map $\{0,1\}^* \to (Q_\exists \to Q \times Q)$. Hence, if one lets $\mathcal{T}$ be the set of functions from $Q_\exists$ into $Q \times Q$, Éloïse's positional strategies are in bijection with $\mathcal{T}$-labelled binary trees.*

7.1.3 Half-Perfect-Information Turn-Based Stochastic Parity Games

In the following we introduce a quite restrictive class of games with imperfect-information which is essentially a stochastic version of the model in [CDHR07]. In particular it differs (up to coding) from the general definition of Section 6 by the following two aspects: (i) Abélard is perfectly informed and (ii) the game is turned-based (Abélard knows Éloïse's action before choosing his action).

A ***turn-based arena of half-perfect-information*** is a tuple $\mathcal{G} = \langle S, \Sigma, T, \sim \rangle$ where S is a finite set of states, Σ is the (finite) alphabet of Éloïse's actions, $T \subseteq S \times \Sigma \times \mathcal{D}(S)$ is a (finite) stochastic transition relation and $\sim$ is an equivalence relation over S. We additionally require that for all $(s, \sigma) \in S \times \Sigma$ there is at least one $\delta \in \mathcal{D}(S)$ such that $(s, \sigma, \delta) \in T$. If for all probability distributions in T the support is a singleton, we say that $\mathcal{G}$ is an imperfect-information ***deterministic*** arena and we see T as a subset of $S \times \Sigma \times S$.

An ***imperfect-information stochastic parity game*** is a pair $\mathbb{G} = \langle \mathcal{G}, \mathrm{Col} \rangle$ where $\mathcal{G}$ is a turn-based arena of half-perfect-information with set of states S and $\mathrm{Col} : S \to \mathbb{N}$ is a colouring function defining a parity condition $\Omega_{\mathrm{Col}} \subseteq S^\omega$. The game is ***deterministic*** if its underlying arena is deterministic.

A ***play*** from some initial state s_0 proceeds as follows: Éloïse plays an action $\sigma_0 \in \Sigma$, then Abélard resolves the non-determinism by choosing a distribution δ_0 such that $(s_0, \sigma_0, \delta_0) \in T$ and finally a new state is randomly chosen according to δ_0. Then Éloïse plays a new action, Abélard resolves the non-determinism and a new state is randomly chosen and so on forever. Hence a play is an infinite word $s_0\sigma_0\delta_0 s_1\sigma_1\delta_1 s_2 \cdots \in (S \times \Sigma \times \mathcal{D}(S))^\omega$ and is won by Éloïse if and only if $s_0 s_1 s_2 \cdots \in \Omega_{\mathrm{Col}}$.

Imperfect-information is modeled thanks to the equivalence relation $\sim$ with the meaning that Éloïse cannot distinguish two states that are $\sim$-equivalent. This is important when defining strategies for Éloïse. Indeed, she should not play differently in two indistinguishable plays, where the indistinguishability of Éloïse is based on *perfect recall* [FHMV95], that is: Éloïse cannot distinguish two plays $s_0\sigma_0\delta_0 s_1\sigma_1\delta_1 \cdots s_\ell$ and $s'_0\sigma'_0\delta'_0 s'_1\sigma'_1\delta'_1 \cdots s'_\ell$ with $s_i \sim s'_i$ for all $i \leqslant \ell$ and $\sigma_i = \sigma'_i$ for all $i < \ell$. Hence, a strategy for Éloïse is a function $\varphi : (S_{/\sim} \times \Sigma)^* \cdot (S_{/\sim}) \to \Sigma$ assigning an action to every set of indistinguishable plays. Éloïse ***respects a strategy*** φ during a play $\lambda = s_0 a_0 \delta_0 s_1 a_1 \delta_1 \cdots$ if $a_{i+1} = \varphi([s_0]_\sim a_0 [s_1]_\sim \cdots [s_i]_\sim)$, for all $i \geqslant 0$. As always, a strategy φ for Éloïse is ***surely winning*** if Éloïse wins all plays consistent with φ and it is ***almost-surely winning*** if Éloïse wins almost all plays consistent with φ.

***Remark* 40.** *It is important to note that Éloïse may not observe the colour of the current state, as in general we do not require that* $\sigma \sim \sigma' \Rightarrow \mathrm{Col}(\sigma) = \mathrm{Col}(\sigma')$. *In particular, this has to be taken into account when eventually solving the game.*

7.2 Revisiting The Emptiness Using Imperfect-Information Game

In this section, we introduce our technique by revisiting the emptiness checking of regular tree languages when described by an alternating parity tree automaton. Whereas the standard technique [MS95] first removes alternation and then reduces the emptiness to decide the winner in a finite perfect-information game, our technique goes directly to decide the winner in an imperfect-information game. Our construction is reminiscent of the notion of *blindfold games* introduced by Reif in [Rei79], and later thoroughly extended [WDHR06; WDMR08; DR10].

Fix an alternating parity tree automaton $\mathcal{A} = \langle A, Q_\exists, Q_\forall, q_{\mathrm{ini}}, \Delta, \mathrm{Col} \rangle$. Our goal is to check whether $L(\mathcal{A}) = \varnothing$ and for this we design an imperfect-information *deterministic* parity game. We first present the game, make use of it to decide the emptiness, and then compare the approach with the standard one.

7.2.1 A Half-Perfect-Information Emptiness Game

We define a half-perfect-information deterministic parity game $\mathbb{G}_{\mathcal{A}}$ (as defined in Section 7.1.3) that intuitively works as follows. Éloïse describes both a tree t and a positional strategy φ_t for her in the game $\mathbb{G}_{\mathcal{A},t}$; the strategy φ_t is described as a $\mathcal{T}$-labeled tree (where $\mathcal{T}$ is the set of functions from $Q_\exists$ into $Q \times Q$, see Remark 39 above). As the plays are of ω-length, she actually does not fully describe t and φ_t but only a branch: this branch is chosen by Abélard, who also takes care of computing the sequence of states along it (either by updating an existential state accordingly to φ_t or, when the state is universal, by choosing an arbitrary valid transition of the automaton). In this game, Éloïse observes the directions, but not the actual control state of the automaton (indeed, otherwise she could easily "cheat").

Formally, we let $\mathcal{G}_{\mathcal{A}} = \langle S, \Sigma, T, \sim \rangle$ where $S = (Q \times \{0,1\}) \cup \{(q_{\text{ini}}, \varepsilon)\}$, $\Sigma \subseteq A \times \mathcal{T}$ is the set of pairs (a, τ) such that for all $q \in Q_\exists$ we have that $(q, a, q_0, q_1) \in \Delta$ where $\tau(q) = (q_0, q_1)$, $(q, i) \sim (q', i)$ for all $q, q' \in Q$ and $i \in \{0,1\}$, and

$$T = \{((q,i),(a,\tau),(q_0,0)),((q,i),(a,\tau),(q_1,1)) \mid q \in Q_\exists \text{ and } \tau(q) = (q_0,q_1)\}$$
$$\cup\ \{((q,i),(a,\tau),(q_0,0)),((q,i),(a,\tau),(q_1,1)) \mid q \in Q_\forall \text{ and } (q,a,q_0,q_1) \in \Delta\}$$

Finally we let $\mathbb{G}_{\mathcal{A}} = \langle \mathcal{G}_{\mathcal{A}}, \text{Col}_{\mathcal{A}} \rangle$ be the imperfect-information *deterministic* parity game obtained by letting $\text{Col}_{\mathcal{A}}(q,i) = \text{Col}(q)$ for any $(q,i) \in S$. The following result states that $\mathbb{G}_{\mathcal{A}}$ is an emptiness game for $L(\mathcal{A})$.

Lemma 6 [FPS13]

Éloïse has a surely winning strategy in $\mathbb{G}_{\mathcal{A}}$ from $(q_{\text{ini}}, \varepsilon)$ if and only if $L(\mathcal{A}) \neq \varnothing$.

***Remark* 41.** *From the proof of Lemma 6, one can also conclude that if $L(\mathcal{A}) \neq \varnothing$ then $L(\mathcal{A})$ contains a regular tree (*i.e. *the unfolding of a finite graph). Indeed, this is a direct consequence of the fact that if Éloïse has a surely winning strategy in $\mathbb{G}_{\mathcal{A}}$, then she has one that uses finite memory [CDHR07].*

Lemma 6 provides a reduction of the emptiness problem to the existence of a surely winning strategy in an imperfect-information deterministic game. One also have a converse result.

Lemma 7 [FPS13]

For any imperfect-information deterministic parity game $\mathbb{G}$ one can construct an alternating parity tree automaton $\mathcal{A}_{\mathbb{G}}$ such that Éloïse surely wins in $\mathbb{G}$ if and only if $L(\mathcal{A}_{\mathbb{G}}) \neq \varnothing$. Moreover in $\mathcal{A}_{\mathbb{G}}$ all states are universal.

7.2.2 Comparison with the Standard Approach

The usual roadmap to check the emptiness of an alternating tree automaton is as follows. First one builds an equivalent non-deterministic automaton thanks to Simulation Theorem (see below) and then one checks the emptiness of this latter automaton by solving an associated *perfect-information* game. It is a well-known result that alternating and non-deterministic automata are equi-expressive [MS95].

Theorem 54 (Simulation Theorem) [MS95]

Let $\mathcal{A}$ be an alternating parity tree automaton with n states and using k colours. Then one can effectively construct a non-deterministic parity tree automaton $\mathcal{B}$ such that $L(\mathcal{A}) = L(\mathcal{B})$. The automaton $\mathcal{B}$ has $2^{\mathcal{O}(nk\log(nk))}$ and it uses $\mathcal{O}(nk)$ colours.

Proof. We do not give a complete proof of this classical result [MS95] but we rather exhibit the crucial arguments here to later revisit the emptiness problem for alternating parity tree automata.

Fix an alternating parity tree automaton $\mathcal{A} = \langle A, Q_\exists, Q_\forall, q_{\text{ini}}, \Delta, \text{Col}\rangle$. For any letter $a \in A$, call an ***a-tile*** any subset $\tau \subseteq Q \times Q \times Q$ such that

- for all $q, q_0, q_1 \in Q$, if $(q, q_0, q_1) \in \tau$ then $(q, a, q_0, q_1) \in \Delta$;
- for all $q \in Q_\exists$ there exists a *unique* $(q_0, q_1) \in Q^2$ such that $(q, q_0, q_1) \in \tau$;
- for all $q \in Q_\forall$ and for all $(q, a, q_0, q_1) \in \Delta$, $(q, q_0, q_1) \in \tau$.

Hence, one should think of an a-tile as a description of the local value of a *positional* strategy for Éloïse in a node labeled by a from a tree t in the game $\mathbb{G}_{\mathcal{A},t}$ (the case of $q \in Q_\forall$ is here to leave all options of Abélard). In the following it is convenient to think of a tile τ as a pair (τ_0, τ_1) of ***semi-tiles*** where $\tau_0, \tau_1 \subseteq Q \times Q$ and $(q, q_0) \in \tau_0$ (*resp* $(q, q_1) \in \tau_1$) if and only if there exists $p \in Q$ such that $(q, q_0, p) \in \tau$ (*resp.* $(q, p, q_1) \in \tau$). See Figure 42a for an example.

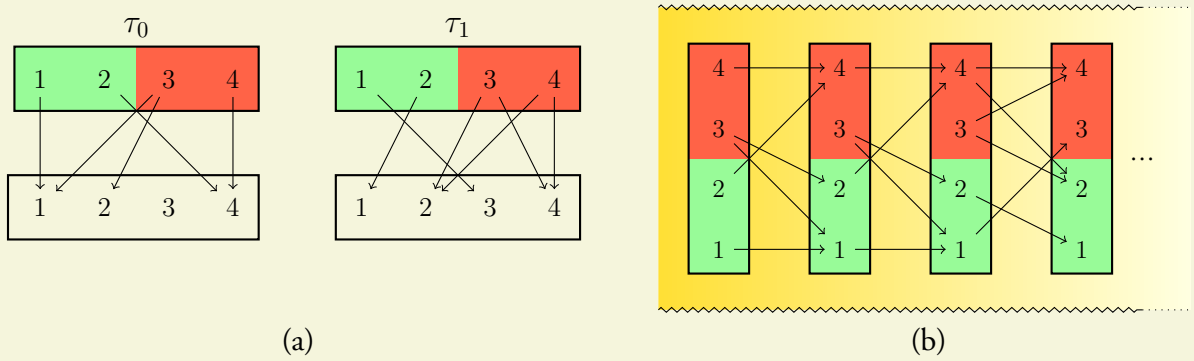


Figure 42 – **(a)** Semi-tiles τ_0 and τ_1 such that (τ_0, τ_1) represents the tile $\{(1,1,3),(2,4,1),(3,1,2),(3,2,4),(4,4,2),(4,4,4)\}$ with $Q_\exists = \{1,2\}$ and $Q_\forall = \{3,4\}$. **(b)** Representation of the sequence of semi-tiles $\tau_0\tau_0\tau_1\cdots$

Now an equivalent non-deterministic automaton $\mathcal{A}'$ is obtained by taking as control states the set S of all possible semi-tiles augmented with an extra dummy initial state s_{ini}. The transition relation Δ' consists of all those elements of the form (s, a, τ_0, τ_1) where s is any state and (τ_0, τ_1) is an a-tile. Acceptance for $\mathcal{A}'$ is then defined by means of a game $\mathbb{G}_{\mathcal{A}',t}$ as previously except that the winning condition is more involved than a parity condition. A play is an element $\lambda = v_0 v_1 \cdots \in (\{0,1\}^* \times S) \cdot (\{0,1\}^* \times S \times S \times S))^\omega$ to which we can associate a sequence of semi-tiles $\pi(\lambda) = s_1 s_2 \cdots$ where $v_{2i} = (n_i, s_i)$ for all integer $i \geqslant 1$ (we ignore the dummy initial state). The sequence $\pi(\lambda)$ can be seen as a set of infinite paths in an infinite ribbon obtained by gluing together the semi-tiles $s_1, s_2, \ldots$ (see Figure 42b). An infinite sequence $q_1 q_2 \cdots$ is an infinite path in $\pi(\lambda)$ if and only if for all $i \geqslant 1$ one has $(q_i, q_{i+1}) \in s_i$; it is ***good*** if $\liminf(\text{Col}(q_i))_{i\geqslant 1}$ is even; and $\pi(\lambda)$ is ***good*** if all plays in it are good. Finally we define those winning plays for Éloïse as those plays λ such that $\pi(\lambda)$ is good.

Then one can easily remark that the set of all λ such that $\pi(\lambda)$ is good is an ω-regular language over S, hence is accepted by a deterministic parity ω-word automaton $\mathcal{C}$. Considering a "synchronised" product of $\mathcal{C}$ together with $\mathcal{A}'$ leads to $\mathcal{B}$. The desired complexity is achieved by carefully constructing $\mathcal{C}$.

■

Consider now a *non-deterministic* parity tree automaton $\mathcal{K} = \langle A, Q, q_{\text{ini}}, \Delta, \text{Col}\rangle$. Recall (see Section 2.3) that a perfect-information deterministic emptiness game for $\mathcal{K}$ is built as follows. We let $G_{\mathcal{K}} = (V_E \uplus V_A, E)$ where $V_E = Q$, $V_A = \Delta$ and

$$E = \{(q, (q, \sigma, q_0, q_1)), ((q, \sigma, q_0, q_1), q_0), ((q, \sigma, q_0, q_1), q_1) \mid (q, \sigma, q_0, q_1) \in \Delta\}$$

We define $\mathbb{G}_{\mathcal{K}} = (\mathcal{G}_{\mathcal{K}}, \text{Col})$ with $\mathcal{G}_{\mathcal{K}} = (G_{\mathcal{K}}, V_E, V_A)$ and where we extend Col by letting $\text{Col}((q, \sigma, q_0, q_1)) = \text{Col}(q)$. Then one has that $L(\mathcal{K}) \neq \varnothing$ if and only if Éloïse surely wins in $\mathbb{G}_{\mathcal{K}}$ from q_{ini}: Indeed, strategies for Éloïse in $\mathbb{G}_{\mathcal{K}}$ are in bijection with pairs made of a tree t and a strategy for Éloïse in $\mathbb{G}_{\mathcal{K},t}$, and this bijection preserves the fact that a strategy is surely winning.

Now think of adapting this construction to the automaton $\mathcal{A}'$ from the proof of the Simulation Theorem and recall that acceptance for $\mathcal{A}'$ was defined thanks to a game as the classical one except that the winning condition was more involved. Then, the same construction provides a game $\mathbb{G}_{\mathcal{A}'}$ where Éloïse's vertices are semi-tiles and Abélard's vertices are tuple made of a semi-tile, a letter in Σ and a tile, and whose winning condition for a play $\lambda = v_0 v_1 \cdots \in ((\Sigma \times S) \cdot (\Sigma \times S \times S \times S))^\omega$ is that $\pi(\lambda) = s_1 s_2 \cdots$ is good (in the previous sense) where $v_{2i} = (\sigma_i, s_i)$ for all integer $i \geqslant 1$.

Now think back to our reduction from $\mathcal{A}$ to $\mathbb{G}_{\mathcal{A}}$: it makes crucial use of positional determinacy while determinisation of automata on infinite word is implicitly needed when deciding whether Éloïse surely wins in $\mathbb{G}_{\mathcal{A}}$. Indeed, one first applies the subset construction of [CDHR07], getting an intermediate perfect-information game isomorphic to $\mathbb{G}_{\mathcal{A}'}$ and then, since Éloïse does not observe the colour, one has to embed in the previous subset/tile construction a deterministic parity automaton over infinite words that checks that all plays consistent with the observations fit the parity condition. As this latter automaton is essentially the automaton $\mathcal{C}$ one gets a perfect-information parity game isomorphic to $\mathbb{G}_{\mathcal{B}}$. Figure 43 summarises the previous discussion.

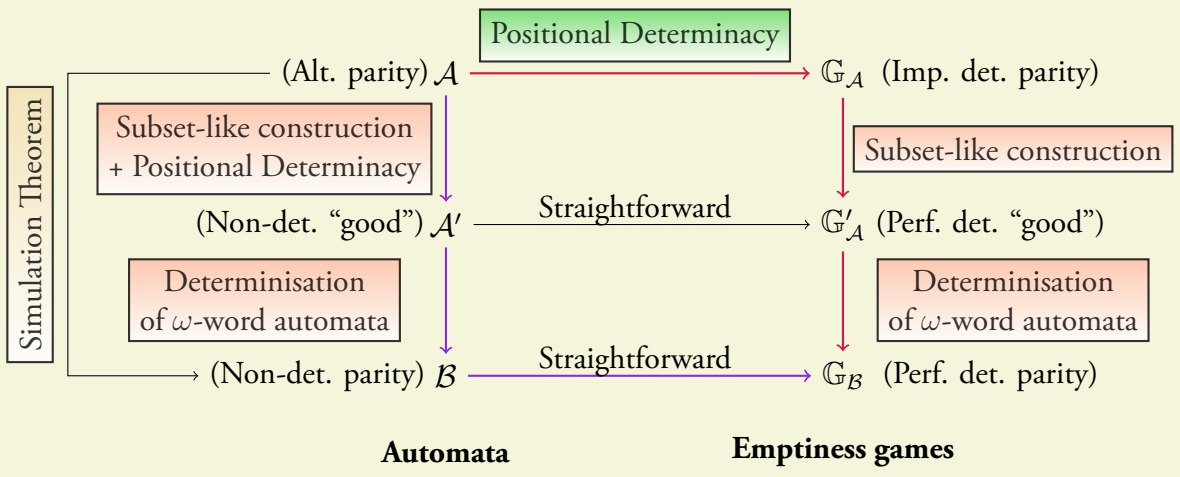


Figure 43 – Roadmaps to emptiness checking: the classical one (purple) *vs* ours (in red).

7.3 Checking Emptiness Using an Imperfect-Information Game: The Case of $L^{=1}(\mathcal{A})$ for Büchi Condition

We are now coming to the central contributions of this Section (Theorem 56). We design an emptiness game for alternating *qualitative* Büchi tree automata adapting the approach developed in Section 7.2. Recall that it relies on two key arguments: a positionality result and a decidability result for games. Hence, we start by a positionality result (Theorem 55), and then explain how to obtain a (decidable) emptiness game (Section 7.4).

7.3.1 A Positionality Result for Chronological Games

We give a positionality result for a subclass of infinite arenas, satisfying the following two conditions:

- the underlying graph has finite out-degree, *i.e* for every $v \in V$ there are finitely many outgoing edges from v, and
- there exists a function $\mathrm{rk} : V \to \mathbb{N}$ such that $\mathrm{rk}^{-1}(0) = \{v_0\}$ and for $(v, v') \in E$, $\mathrm{rk}(v') = \mathrm{rk}(v) + 1$. An arena with the later property is said to be ***chronological***.

Note that both assumptions hold for $\mathbb{G}_{\mathcal{A}}^{=1}$. Also, observe that for any k, the set $\mathrm{rk}^{-1}(k)$ is finite, and the set V of vertices is countable.

Theorem 55 [FPS13]

Let $\mathbb{G}$ be a stochastic turn-based Büchi game whose arena has finite out-degree and is chronological. Then if Éloïse wins almost-surely, then she has a positional winning strategy.

The main idea to prove Theorem 55 is to note that, if Éloïse can ensure to reach F with probability 1 from some initial vertex, then there exists a bound k such that she can ensure to reach F with probability at least $\frac{1}{2}$ within k steps against *any* strategy of Abélard. This allows to "slice" the arena into infinitely many disjoint finite arenas: in each slice Éloïse plays to reach F with probability at least $\frac{1}{2}$. Since each slice forms a finite subarena, optimal positional strategies exist by a famous result due to Condon [Con92] (also see [Kuc11] for a nice survey). The resulting strategy consists in playing in turns the above positional strategies; since each slice gives a probability to reach F of at least $\frac{1}{2}$ before proceeding to the next, the probability to reach F infinitely often is 1.

7.4 The Reduction

Fix an alternating Büchi tree automaton $\mathcal{A} = \langle A, Q_\exists, Q_\forall, q_{\text{ini}}, \Delta, \mathrm{Col}\rangle$. In order to check whether $L^{=1}(\mathcal{A}) = \varnothing$, we design a half-perfect-information *stochastic* Büchi game $\mathbb{G}_{\mathcal{A}}^{=1}$, in a way similar to the one to decide whether $L(\mathcal{A}) = \varnothing$ taking advantage of the positionality result established in Theorem 55.

In the game, Éloïse describes both a tree t and a positional strategy φ_t for her in the game $\mathbb{G}_{\mathcal{A},t}^{=1}$; the strategy φ_t is described as a $\mathcal{T}$-labeled tree (where $\mathcal{T}$ is the set of functions from $Q_\exists$ into $Q \times Q$, see Remark 39). As the plays are of ω-length, she actually does not fully describe t and φ_t but only a branch: this branch is chosen by Random, and Abélard takes care of computing the sequence of states along it (either by updating an existential state accordingly to φ_t or, when the state is universal, by choosing an arbitrary valid transition of the automaton). In this game, Éloïse observes the directions, but not the actual control state of the automaton.

Formally, we let $\mathcal{G}_{\mathcal{A}}^{=1} = \langle S, \Sigma, T, \sim \rangle$ where $S = (Q \times \{0,1\}) \cup \{(q_{\text{ini}}, \varepsilon)\}$, $\Sigma \subseteq A \times \mathcal{T}$ is the set of pairs (a, τ) such that for all $q \in Q_\exists$, $(q, a, q_0, q_1) \in \Delta$ where $\tau(q) = (q_0, q_1)$ and $T = \{((q,i),(a,\tau),d_{q_0,q_1}) \mid q \in Q_\exists \text{ and } \tau(q) = (q_0,q_1)\} \cup \{((q,i),(a,\tau),d_{q_0,q_1}) \mid q \in Q_\forall \text{ and } (q,a,q_0,q_1) \in \Delta\}$ where d_{q_0,q_1} is the probability distribution $(q_0,0) \mapsto 1/2$ and $(q_1,1) \mapsto 1/2$, and $(q,i) \sim (q',i)$ for all $q, q' \in Q$ and $i \in \{0,1\}$. Define $\mathbb{G}_{\mathcal{A}}^{=1} = \langle \mathcal{G}_{\mathcal{A}}, \text{Col}_{\mathcal{A}} \rangle$ with $\text{Col}_{\mathcal{A}}(q,i) = \text{Col}(q)$ for any $(q,i) \in S$.

We have the following result which is the "qualitative" counterpart of Lemma 6.

Lemma 8

Éloïse almost-surely wins in $\mathbb{G}_{\mathcal{A}}^{=1}$ from and $(q_{\text{ini}}, \varepsilon)$ if and only if $L^{=1}(\mathcal{A}) \neq \varnothing$.

From Theorem 48 one can decide almost-sure winning in imperfect-information Büchi games in ExpTime, hence the same holds for checking the emptiness of languages of the form $L^{=1}(\mathcal{A})$. One can also reduce the emptiness problem for probabilistic ω-words automaton with the almost-sure semantics (in the sense of [BGB12]) to check the emptiness of languages of the form $L^{=1}(\mathcal{A})$ hence, it implies lower bounds as well as undecidability results.

Theorem 56

The following properties hold.

- Deciding whether $L^{=1}(\mathcal{A}) = \varnothing$ for a given alternating Büchi tree automaton $\mathcal{A}$ is an ExpTime-complete problem.
- Deciding whether $L^{=1}(\mathcal{A}) = \varnothing$ for a given alternating co-Büchi tree automaton $\mathcal{A}$ is an undecidable problem.

***Remark* 42.** *As one can decide whether $L^{=1}(\mathcal{A}) = \varnothing$ for non-deterministic tree automata [CHS11], Theorem 56 implies that there is no* effective *simulation theorem for co-Büchi alternating qualitative tree automata.*

8 Beyond Non-Deterministic Automata: the Probabilistic Setting

Following the pioneer work of Rabin [Rab63] for finite words and the more recent one by Baier *et al.* [BG05; BBG08; BGB12] for infinite words we investigate probabilistic automata on *infinite trees*. That is the set of transitions of an automaton is replaced by a probability distribution over the set of all transitions which induces a probability measure on the set of runs of the automaton. Now, a tree is accepted if *almost every* run over the input tree is accepting. For the run, we may use either the classical or the qualitative acceptance criterion.

8.1 Definitions

8.1.1 Probabilistic Tree Automata

A ***probabilistic tree automaton*** $\mathcal{A}$ is a tuple $\langle A, Q, q_{ini}, \delta, \text{Acc} \rangle$ where A is the ***input alphabet***, Q is a finite ***set of states***, $q_{\text{ini}} \in Q$ is the ***initial state***, $\text{Acc} \subseteq Q^\omega$ is the ***acceptance condition*** and δ is a mapping from $Q \times A \times Q \times Q$ to $[0,1]$ such that for all $q \in Q$ and $a \in A$, $\sum_{q_1,q_2 \in Q} \delta(q,a,q_1,q_2) = 1$.

Intuitively, the value $\delta(q, a, q_1, q_2)$ is the probability for a transition $q \xrightarrow{a} (q_1, q_2)$ to be used by the automaton when it is in state q and reads the symbol a.

This probability distribution on the transitions induces a probability measure on the set of runs of $\mathcal{A}$. In this setting, a ***run*** of $\mathcal{A}$ is simply a Q-labeled tree whose root is labeled by the initial state q_{ini}. We denote by $\text{Runs}(\mathcal{A})$ (or simply Runs if $\mathcal{A}$ is clear from the context) the set of all runs of $\mathcal{A}$. We denote by $\text{AccRuns}(\mathcal{A})$ the set of accepting runs of $\mathcal{A}$ and by $\text{QualAccRuns}(\mathcal{A})$ the set of qualitatively accepting runs of $\mathcal{A}$.

In the sequel, we will show that, for a given tree t the sets $\text{AccRuns}(\mathcal{A})$ and $\text{QualAccRuns}(\mathcal{A})$ are measurable (Proposition 18), and this will allow us to define almost-sure acceptance of a tree by a probabilistic automaton.

8.1.2 Measurability of $\text{AccRuns}(\mathcal{A})$ and $\text{QualAccRuns}(\mathcal{A})$

We first define a σ-algebra for $\text{Runs}(\mathcal{A})$. A ***partial run*** is a partial function $\lambda : \{0,1\}^* \to Q$ with $\lambda(\varepsilon) = q_{\text{ini}}$ and such that $\text{Dom}(\lambda)$ is finite, non-empty, prefix-closed and such that for all $w \in \{0,1\}^*$, $w0 \in \text{Dom}(\lambda)$ if and only if $w1 \in \text{Dom}(\lambda)$. If λ is a partial run, we denote by $\text{Inner}(\lambda)$ the set $\{w \in \text{Dom}(\lambda) \mid w0 \in \text{Dom}(\lambda) \text{ and } w1 \in \text{Dom}(\lambda)\}$ of non-leaf nodes. The cylinder (see Figure 44 for an illustration) associated with λ, denoted $\text{Cyl}(\lambda)$, is the set of runs consistent with λ, *i.e.*

$$\text{Cyl}(\lambda) = \{\rho \in \text{Runs} \mid \forall w \in \text{Dom}(\lambda), \rho(w) = \lambda(w)\}$$

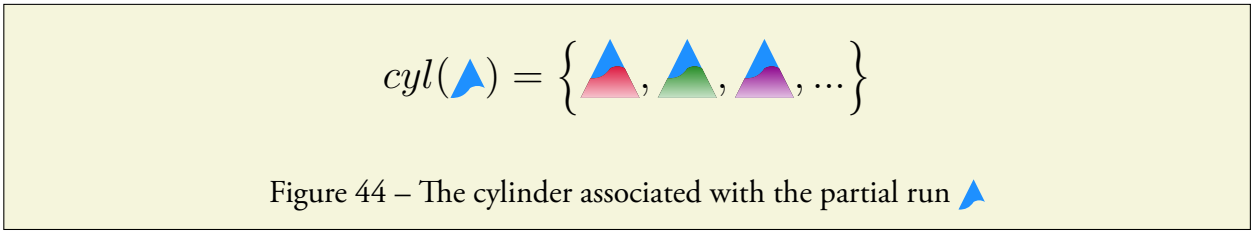


Figure 44 – The cylinder associated with the partial run

Let $\mathcal{F}_R$ be the σ-algebra generated by the cylinders. By Carathéorody's extension theorem, there exists a unique probability measure μ_t on the measurable space $(\text{Runs}, \mathcal{F}_R)$ such for all partial run $\lambda : \{0,1\}^* \to Q$,

$$\mu_t(\text{Cyl}_{\mathcal{A}}(\lambda)) = \prod_{w \in \text{Inner}(\lambda)} \delta(\lambda(w), t(w), \lambda(w0), \lambda(w1))$$

Note that both μ_t and $(\text{Runs}, \mathcal{F}_R)$ depend on t.

Let $\mathcal{A}$ be an automaton and define the mapping $f_{\mathcal{A}} : \text{Runs} \times \text{Br} \to [0,1]$ associating to a pair $(\rho, \pi) \in \text{Runs} \times \text{Br}$ the value 1 if $\rho(\pi)$ belongs to Acc and 0 otherwise. The following lemma states that $f_{\mathcal{A}}$ is integrable.

Lemma 9 [CHS11]

Let $\mathcal{A}$ be a probabilistic tree automaton with an ω-regular acceptance condition and let t be a tree. The mapping $f_{\mathcal{A}}$ is integrable in the product space $(\text{Runs}(\mathcal{A}), \mathcal{F}_R, \mu_t) \otimes (\text{Br}, \mathcal{F}_{\text{Br}}, \mu)$.

Lemma 9 permits to establish the measurability of the sets $\text{AccRuns}(\mathcal{A})$ and $\text{QualAccRuns}(\mathcal{A})$.

Proposition 18 [CHS11]

For all probabilistic tree automata $\mathcal{A}$ with an ω-regular acceptance condition, the set QualAccRuns($\mathcal{A}$) is measurable.

For all probabilistic Büchi tree automata $\mathcal{A}$, the set AccRuns($\mathcal{A}$) is measurable.

***Remark* 43.** *Recently it was pointed to us by Thomas Weidner [Wei14] that the set* AccRuns($\mathcal{A}$) *may not be measurable for co-Büchi automata. His argument is the following. In [NW03] it was proven that there are languages of infinite trees (*e.g. *all trees over an alphabet {a,b} with finitely many b's on every branch) that are not Borel while recognised by a co-Büchi automaton (in the classical sense or regular tree languages), thus also the set of runs satisfying a co-Büchi condition is not Borel and hence not measurable.*

8.1.3 Almost-Surely Accepted Trees

Thanks to Proposition 18 the following definitions are well-founded.

A tree t is ***(almost-surely) accepted*** by a Büchi automaton $\mathcal{A}$ with the *classical* semantics if almost all runs of $\mathcal{A}$ on t are accepting (*i.e.* $\mu_t(\text{AccRuns}(\mathcal{A})) = 1$). We denote by $\boldsymbol{L^{=1}(\mathcal{A})}$ the set of trees accepted by $\mathcal{A}$ with the classical semantics for runs.

A tree t is ***(almost-surely) accepted*** by a parity automaton $\mathcal{A}$ with the *qualitative* semantics if almost all runs of $\mathcal{A}$ on t are accepting (*i.e.* $\mu_t(\text{QualAccRuns}(\mathcal{A})) = 1$). We denote $\boldsymbol{L^{=1}_{\text{Qual}}(\mathcal{A})}$ the set of trees accepted by $\mathcal{A}$ with the qualitative semantics for runs.

***Remark* 44.** *Our motivation for considering almost-sure acceptation and not positive acceptation will be discussed later in Section 8.5.*

***Remark* 45.** *The definition of [BG05] (for ω-words) allows for incomplete automata (*i.e. *for all $q \in Q$, the sum $\sum_{q' \in Q} \delta(q, a, q') = 1$ or 0). However it is easy to verify that every automaton can be rendered complete without changing the acceptance condition and the measure of acceptation. Remark that removing all states q such that $\sum_{q_1, q_2 \in Q} \delta(q, a, q_1, q_2) = 0$ does not change the measure of the set of accepting runs (either in the classical or qualitative sense). By iteratively applying this process, we obtain an equivalent complete automata.*

Lemma 9 and classical techniques from measure theory (see [Bau01] for a textbook on that topic) permit to show that the almost-sure acceptation of a tree t by an automaton $\mathcal{A}$ for the qualitative semantics can be defined by integrating the mapping $f_{\mathcal{A}}$.

Proposition 19 [CHS11]

Let $\mathcal{A}$ be a probabilistic tree automaton with an ω-regular acceptance condition and let t be a tree. Then we have:

$$t \in L^{=1}_{\text{Qual}}(\mathcal{A}) \Leftrightarrow \int f_{\mathcal{A}} d\mu_t \otimes \mu = 1$$

The intuition behind Proposition 19 is that a tree t belongs to $L^{=1}_{\text{Qual}}(\mathcal{A})$ if and only if when picking "randomly" and simultaneously a branch and a (piece of) run (along it), this branch in the run fits the parity condition. This intuition is made formal in Proposition 21.

8.1.4 Examples

We now give some examples of languages accepted by probabilistic tree automata.

For an ω-word language $L \subseteq \{a,b\}^\omega$, we denote by $\mathrm{Path}^{=1}(L)$ the set of trees labeled by $\{a,b\}$ with almost all their branch labels in L (*i.e.* $\mu(\{\pi \in \mathrm{Br} \mid t(\pi) \in L\}) = 1$). It is easy to see that, for any ω-regular language L, the tree language $\mathrm{Path}^{=1}(L)$ is a qualitative tree language.

More interesting, if L is almost-surely accepted by a probabilistic ω-word automaton[9] with an ω-regular acceptance condition, we can show that $\mathrm{Path}^{=1}(L)$ is accepted by a probabilistic tree automaton (with the qualitative semantics).

Proposition 20 [CHS11]

Given a probabilistic ω-word automaton $\mathcal{B}$ with an ω-regular acceptance condition, there exists a probabilistic tree automaton $\mathcal{A}$ with the same acceptance condition such that $L^{=1}_{\mathrm{Qual}}(\mathcal{A})$ is equal to $\mathrm{Path}^{=1}(L^{=1}(\mathcal{B}))$.

8.2 An Acceptance Game for Qualitative Probabilistic Tree Automata

Fix a probabilistic tree automaton $\mathcal{A} = \langle A, Q, q_{\mathrm{ini}}, \delta, \mathrm{Acc}\rangle$ and a tree t. We define an *infinite* Markov chain $\mathcal{M}_{\mathcal{A},t} = (\mathcal{G}_{\mathcal{A},t}, \mathcal{O}_{\mathrm{Acc}})$ such that $\mathcal{M}_{\mathcal{A},t}$ almost-surely fulfils its objective if and only if t belongs to $L^{=1}_{\mathrm{Qual}}(\mathcal{A})$. Compared with the acceptance game for qualitative tree automata, the transition is no longer chosen by Éloïse: it is now randomly chosen with the probability distribution given by $\mathcal{A}$. Hence, this is why we simply obtain a Markov chain instead of an MDP.

The Markov arena $\mathcal{G}_{\mathcal{A},t}$ is equal to $\langle S, s_{\mathrm{ini}}, \delta\rangle$ where $S = Q \times \{0,1\}^* \cup \Delta \times \{0,1\}^*$ with $\Delta = Q \times Q \times Q$, $s_{\mathrm{ini}} = (q_{\mathrm{ini}}, \varepsilon)$ and $\delta : S \mapsto D(S)$ is defined as follows. For all $w \in \{0,1\}^*$ and all $q \in Q$, $\delta((q,w))((q,q_0,q_1)) = \delta(q, t(w), q_0, q_1)$ for all $q_0, q_1 \in Q$ and is equal to 0 otherwise. For all $w \in \{0,1\}^*$ and $q_0, q_1 \in Q$, $\delta(((q,q_0,q_1),w),(q_0,w0) = \delta(((q,q_0,q_1),w),(q_1,w1)) = \frac{1}{2}$ and 0 otherwise. Recall that $\mu_{\mathcal{M}_{\mathcal{A},t}}$ denotes the probability measure associated to $\mathcal{M}_{\mathcal{A},t}$.

Note that a trace in the Markov chain $\mathcal{M}_{\mathcal{A},t}$ can be uniquely represented by an infinite sequence $((p_0,q_0^0,q_0^1),a_0)((p_1,q_1^0,q_1^0),a_1)\ldots$ labeled by $\Delta \times \{0,1\}$ such that $p_0 = q_{\mathrm{ini}}$ and for all $i \geqslant 0$, $p_{i+1} = q_i^{a_i}$. The objective $\mathcal{O}_{\mathrm{Acc}}$ is the set of traces $((p_0,q_0^0,q_0^1),a_0)((p_1,q_1^0,q_1^0),a_1)\ldots$ such that $p_0p_1\ldots$ belongs to Acc.

The next proposition follows from Proposition 19 and classical results from measure theory.

Proposition 21 [CHS11]

Let $\mathcal{A}$ be a probabilistic tree automaton with an ω-regular acceptance condition and let t be a tree. We have $t \in L^{=1}_{\mathrm{Qual}}(\mathcal{A})$ if and only if $\mathcal{M}_{\mathcal{A},t}$ almost-surely fulfils its objective.

8.3 Decidability and Undecidability Results

In this section, we show that the emptiness problem for probabilistic *Büchi* tree automata is decidable for the qualitative semantics for runs. The naive idea is to start from Proposition 21 and to design a Markov Decision Process as we did in Theorem 42 for qualitative non-deterministic tree automata: we let Éloïse pick the input tree and synchronously simulate the Markov chain from

[9] In our context probabilistic ω-word automata are simply probabilistic tree automata running over unary trees. For such an automaton $\mathcal{B}$, we denote by $L^{=1}(\mathcal{B})$ the language almost-surely accepted by $\mathcal{B}$.

Proposition 21. Obviously Éloïse could cheat, even in the case of ω-words. Indeed, just think of a probabilistic automaton that can either check with probability $1/2$ that the input words contains infinitely many a's either check with probability $1/2$ that the input words contains finitely many a's, and assume that this choice is made from the very first transition (hence there is only one random transition and it occurs at the very beginning and then the automaton behaves deterministically): any word is accepted with probability $1/2$ but if one considers the MDP where Éloïse chooses the input word at the same time as the run is (randomly) chosen, she has a surely winning strategy that consists in waiting for the first transition, observe what the automaton will check for and depending on that either produce a word with infinitely many a's or with finitely many a's. Hence, it is important here that Éloïse does not have a perfect information which explains why we need to consider so-called partial observation Markov decision processes.

8.3.1 Partial Observation Markov Decision Processes

Consider a Markov arena $\mathcal{G} = \langle S, s_{\text{ini}}, \Sigma, \delta \rangle$. To reflect the fact that Éloïse has imperfect information about the current state we consider an equivalence relation $\sim$ over S.

The intuitive meaning of $\sim$ is that two states $s_1 \sim s_2$ cannot be distinguished by Éloïse. We easily extend $\sim$ to partial plays: $s_0 s_1 \cdots s_n \sim s'_0 s'_1 \cdots s'_n$ if and only if $s_i \sim s'_i$ for all $i = 0, \cdots, n$. As two equivalent plays $\lambda_1 \sim \lambda_2$ cannot be distinguished by Éloïse she should therefore behave the same in both of them.

Hence, we should only consider so-called observation-based strategies. An ***observation-based (pure) strategy*** is a function $\varphi : (S/_\sim)^* \to \Sigma$, *i.e.*, to choose her next action, Éloïse considers the sequence of observations she got so far[10]. In particular, an observation-based strategy φ is such that $\varphi(\lambda) = \varphi(\lambda')$ whenever $\lambda \sim \lambda'$. In this context, a ***positional strategy*** is a function from $S/_\sim \to \mathcal{D}(\Sigma)$, *i.e.* it only depend on the current equivalence class.

A ***partial observation Markov decision process (POMDP)*** is a triple $(\mathcal{G}, \sim, \mathcal{O})$ where $\mathcal{G}$ is an arena, $\sim$ is an equivalence relation over states and $\mathcal{O}$ is an objective. We say that Éloïse ***almost-surely wins*** $\mathbb{G}$ if she has an almost-surely winning *observation-based* strategy.

The following decidability results are known for POMDP:

Theorem 57 [BBG08]

In a POMDP with a Büchi objective, deciding whether Éloïse almost-surely wins is ExpTime-complete. Moreover if Éloïse has an almost-surely winning strategy, she has an almost-surely winning strategy with finite memory.

In a POMDP with a co-Büchi objective, it is undecidable whether Éloïse almost-surely wins.

8.3.2 Decidability Status of the Emptiness Problem for Probabilistic Tree Automata

We reduce the emptiness problem for probabilistic tree automata to deciding almost-surely winning in a POMDP, and the reduction works for *any* ω-regular acceptance condition. However, as the corresponding decision problem on POMDPs is only decidable for the Büchi condition we only obtain decidability is the Büchi case.

Let $\mathcal{A} = \langle A, Q, q_{\text{ini}}, \delta, \text{Acc} \rangle$ be a probabilistic automaton with an ω-regular acceptance condition and let $\Delta = Q \times Q \times Q$.

Define a POMDP $\mathbb{G}_\mathcal{A} = (\mathcal{G}, \sim, \mathcal{O})$ as follows. The arena $\mathcal{G}$ is equal to $\langle S, s_{\text{ini}}, A, \delta \rangle$ where $S = Q \times \{0, 1, \bot\} \times (\Delta \cup \{\bot\})$, $s_{\text{ini}} = (q_0, \bot, \bot)$ and δ is defined as follows. For all $a \in A$

[10] By abuse of notation, we shall write $\varphi(s_0 \cdots s_n)$ to mean $\varphi([s_0]_\sim \cdots [s_n]_\sim)$

and $(p, x, t) \in S$, $\delta((p, x, t), a)$ is the distribution that assigns $\frac{1}{2}\delta(p, a, q_0, q_1)$ to $(q_y, y, (p, q_0, q_1))$ where $y = 0, 1$ and 0 to all other states. The objective $\mathcal{O}$ is the set of plays for which the sequence of states obtained when projecting on the first component belongs to Acc. The equivalence $\sim$ is defined by $(q, x, t) \sim (q', x', t')$ if and only if $x = x'$.Intuitively in $\mathbb{G}_{\mathcal{A}}$, Éloïse describes a branch along a tree and Random builds a piece of run along this branch. As Éloïse does not observe the state in the run constructed by Random, it does not influence her choice for the branch.

Theorem 58 [CHS11]

Let $\mathcal{A}$ be a probabilistic tree automaton with an ω-regular acceptance condition. The language $L^{=1}_{\text{Qual}}(\mathcal{A})$ is non-empty if and only if Éloïse almost-surely wins in $\mathbb{G}_{\mathcal{A}}$.

For the Büchi acceptance condition, this leads to a decidability result for the emptiness problem.

Corollary 13 [CHS11]

Let $\mathcal{A}$ be a probabilistic Büchi tree automaton. Deciding the emptiness of $L^{=1}_{\text{Qual}}(\mathcal{A})$ is an ExpTime-complete problem. Moreover, if $L^{=1}_{\text{Qual}}(\mathcal{A}) \neq \varnothing$, it contains a regular tree.

On the negative side, we show that the emptiness problem for probabilistic co-Büchi tree automata is undecidable for the qualitative semantics for runs. These results are obtained by reduction to the undecidability of the emptiness problem for co-Büchi ω-word automata with the almost-sure acceptation [BBG08].

Theorem 59 [CHS11]

The following problems are undecidable :

- Given a probabilistic co-Büchi tree automaton $\mathcal{A}$, decide if $L^{=1}_{\text{Qual}}(\mathcal{A}) = \varnothing$.

8.4 Expressiveness: Comparison with Regular Tree Languages & Qualitative Tree Languages

We now discuss expressiveness. First, we exhibit a family of tree languages that are accepted by a co-Büchi probabilistic automaton but that are neither regular tree languages nor qualitative tree languages. On the other hand, we give an example of a qualitative tree language that no probabilistic automaton (regardless of its semantics) can accept and another example of a regular tree language that no probabilistic automaton (regardless of its semantics) can accept. Hence, this proves incomparability of probabilistic tree languages with both qualitative and regular tree languages.

For this we consider, for all $0 < \lambda < 1$, the ω-word language L_λ over $\{a, b\}$ defined by

$$L_\lambda = \{a^{k_1} b a^{k_2} b \dots \mid k_1, k_2, \dots > 0 \text{ s.t. } \prod_{i=1}^{\infty} (1 - \lambda^{k_i}) > 0\}$$

In [BBG08], L_λ is shown to be almost-surely accepted by a co-Büchi probabilistic automaton[11]. Therefore, by Proposition 20, $\text{Path}^{=1}(L_\lambda)$ is a co-Büchi probabilistic qualitative tree language.

Proposition 22 [CHS11]

For all $0 < \lambda < 1$, $\text{Path}^{=1}(L_\lambda)$ is not a qualitative tree language.

***Remark* 46.** *Using the correspondence with POMDP introduced in Theorem 58, any co-Büchi automaton accepting* $\text{Path}^{=1}(L_\lambda)$ *gives rise to an example of a co-Büchi POMDP* $\mathbb{G}_\mathcal{A}$ *in which Éloïse needs infinite memory to almost-surely win.*

We now give an example of a qualitative tree language that no probabilistic tree automaton can accept. For this we consider the language $\mathcal{L}_a^{0\vee 1}$ of $\{a,b\}$-labelled trees such that either the left subtree or the right subtree of the root belongs to the language $\mathcal{L}_a$ of example 27 (recall that $\mathcal{L}_a$ is the language of $\{a,b\}$-labelled trees whose set of branches containing at least one a has measure 1). Formally $\mathcal{L}_a^{0\vee 1} = \{t \mid t_0 \in \mathcal{L}_a \text{ or } t_1 \in \mathcal{L}_a\}$. One easily verifies that $\mathcal{L}_a^{0\vee 1}$ is a qualitative tree language. However, $\mathcal{L}_a^{0\vee 1}$ cannot be recognised by a probabilistic tree automaton.

Proposition 23

The language $\mathcal{L}_a^{0\vee 1}$ cannot be recognised by a probabilistic tree automaton (regardless of its semantics).

We now give an example of a regular tree language that no probabilistic tree automaton can accept. For this, let t_a (*resp.* t_b) be the tree whose nodes are all labeled by a (*resp.* b), *i.e.* $t_a(u) = a$ (*resp.* $t_b(u) = b$) for all node $u \in \{0,1\}^*$. Let t be the tree defined by $t(\varepsilon) = b$, $t_0 = t_a$ and $t_1 = t_b$; and t' be the tree defined by $t'(\varepsilon) = b$, ${t'}_0 = t_b$ and ${t'}_1 = t_a$. Finally let $L_{0\vee 1} = \{t, t'\}$: it is a regular tree language as it consists of two regular trees. However, $\mathcal{L}_{0\vee 1}$ cannot be recognised by a probabilistic tree automaton.

Proposition 24

The language $\mathcal{L}_{0\vee 1}$ cannot be recognised by a probabilistic tree automaton (regardless of its semantics).

8.5 Variants

A natural variant is to replace the almost-sure acceptance condition on the set of runs by the *probable* one. That is a tree t is ***probably accepted*** by $\mathcal{A}$ if $\mu_t(\text{AccRuns}(\mathcal{A})) > 0$.

[11] Actually they showed that L_λ is positively accepted by a Büchi automaton. But as the class of languages defined by Büchi automata with the probable semantics is closed under complement it implies that it coincides with the class of languages defined by co-Büchi automata with the almost-sure semantics. Indeed, let L be accepted by a Büchi automaton with the probable acceptance. As one can complement, there is a Büchi automaton with the probable semantics $\mathcal{A}$ such that $\overline{L}$ is the language accepted by $\mathcal{A}$. If one sees $\mathcal{A}$ as a co-Buchi automaton $\mathcal{B}$ (final states becoming the forbidden ones) with an almost-sure semantics, $\mathcal{B}$ accepts a word if and only if $\mathcal{A}$ does not. Hence, $\mathcal{B}$ recognises the complement of $\overline{L}$, namely L. Hence, L_λ is accepted by a co-Büchi probabilistic automaton.

Combining the conditions on the set of runs – almost-sure $(=1)$ and probable (>0) – with the one on the set of accepting branches – qualitative (Qual) and positive (Pos) – we obtain four semantics for probabilistic tree automata denoted by $(>0, \text{Qual})$, $(>0, \text{Pos})$, $(=1, \text{Qual})$ and $(=1, \text{Pos})$ where the first component corresponds to the requirement on the set of accepting runs and the second to the requirement on the set of accepting branches of a run.

In Section 8, we mainly dealt with $(=1, \text{Qual})$-probabilistic automata which have a tight link with POMDP for the almost-sure winning condition (cf. Theorem 58). It can be shown that $(>0, \text{Pos})$-probabilistic automata share the same connection with POMDP with the *positive* winning condition. It implies that the emptiness problem for the $(>0, \text{Pos})$-probabilistic automata with a *co-Büchi* acceptance condition is ExpTime-complete.

When the two conditions are not of the same nature (as for the $(>0, \text{Qual})$ and $(=1, \text{Pos})$ semantics), we were unable to define a proper acceptance game.

We now briefly discuss properties of $(>0, \text{Pos})$-probabilistic automata. If $\mathcal{A}$ is a probabilistic tree automaton with an ω-regular acceptance condition, we denote by $\text{QualAccRuns}^{>0}(\mathcal{A})$ the set of positively accepting runs of $\mathcal{A}$ and by $L^{>0}_{\text{Pos}}(\mathcal{A})$ the set of trees accepted by $\mathcal{A}$ with $(>0, \text{Pos})$-semantics. The proposition below (similar to Proposition 18) justifies the of definition of $L^{>0}_{\text{Pos}}(\mathcal{A})$.

Proposition 25 [CHS11]

For all probabilistic tree automata $\mathcal{A}$ with an ω-regular acceptance condition the set $\text{QualAccRuns}^{>0}(\mathcal{A})$ is measurable.

The following proposition is an adaptation of Proposition 19 to the setting of $(>0, \text{Pos})$-probabilistic automata.

Proposition 26 [CHS11]

Let $\mathcal{A}$ be a probabilistic tree automaton with an ω-regular acceptance condition and let t be a tree. We have

$$t \in L^{>0}_{\text{Pos}}(\mathcal{A}) \Leftrightarrow \int f_{\mathcal{A}} d\mu_t \otimes \mu > 0$$

We can also transfer the results on the acceptance game (Proposition 21 and Theorem 58).

Proposition 27 [CHS11]

Let $\mathcal{A}$ be a probabilistic tree automaton with an ω-regular acceptance condition and let t be a tree. We have $t \in L^{>0}_{\text{Pos}}(\mathcal{A})$ if and only if $\mathcal{M}_{\mathcal{A},t}$ positively fulfils its objective.

Theorem 60 [CHS11]

Let $\mathcal{A}$ be a probabilistic tree automaton with an ω-regular acceptance condition. The language $L^{>0}_{\text{Pos}}(\mathcal{A})$ is non-empty if and only if Éloïse positively wins in $\mathbb{G}_{\mathcal{A}}$.

	$L^{=1}(\mathcal{A})$	$L^{=1}_{\text{Qual}}(\mathcal{A})$	$L^{>0}_{\text{Pos}}(\mathcal{A})$	$L^{=1}_{\text{Pos}}(\mathcal{A})$ / $L^{>0}_{\text{Qual}}(\mathcal{A})$
Büchi	Open	ExpTime-complete (Corollary 13)	Undecidable (Theorem 61)	Open
Co-Büchi	Undefined	Undecidable (Theorem 59)	ExpTime-complete (Corollary 14)	Open

Table 3 – Decidability status of the emptiness problem for the different types of probabilistic semantics.

If we consider a co-Büchi acceptance condition, this leads to a decidability result for the emptiness problem (which is a dual version of Corollary 13).

Corollary 14 [CHS11]

Let $\mathcal{A}$ be a probabilistic co-Büchi tree automaton. Deciding the emptiness of $L^{>0}_{\text{Pos}}(\mathcal{A})$ is an ExpTime-complete problem. Moreover, if $L^{>0}_{\text{Pos}}(\mathcal{A}) \neq \varnothing$, it contains a regular tree.

Finally, as a dual version of Theorem 59, we show that the emptiness problem for a Büchi acceptance condition is undecidable.

Theorem 61 [CHS11]

The following problem is undecidable : given a probabilistic Büchi tree automaton $\mathcal{A}$, decide if $L^{>0}_{\text{Pos}}(\mathcal{A}) = \varnothing$.

8.6 Summary of the Decidability Results on Probabilistic Tree Automata

We terminate this Section with Table 3 that summarises the (un)decidability results and open questions on the emptiness problem for the different types of probabilistic semantics that we considered.

9 Open Problems and Perspectives

We now list several natural problems as well as perspectives related to the topics we have discussed in the present chapter.

9.1 Qualitative Tree Languages *vs* Regular Tree Languages

We have seen (Section 5.3) that qualitative tree languages and regular tree languages are incomparable. Hence, there are two series of natural questions.

- Given a qualitative tree language can one decide if it is regular?

- Given a regular tree language can one decide if it is qualitative?
- Find sufficient conditions on qualitative tree languages to be regular. A candidate is the following. Let L be a tree language. Assume that L is both a qualitative tree language and a positively tree language. Is it the case that L is always a regular tree language?
- Find sufficient conditions on regular tree languages to be qualitative. A first natural hint for that is imposed by the pumping Lemma on qualitative tree languages (which is notably false for regular tree languages).

9.2 The Class of Qualitative Tree Languages *vs* Distribution $\mu_{\mathbf{p}}$

A more general definition for qualitative tree languages is to associate with any letter a in the alphabet a pair $(p_a^0, p_a^1) \in [0,1]^2$ with $p_a^0 + p_a^1 = 1$ and then to define the measure of a cone in a tree t by letting $\mu(\mathrm{Cone}(u_1 \cdots u_n)) = p_{t(\varepsilon)}^{u_1} p_{t(u_1)}^{u_2} \cdots p_{t(u_1 \cdots u_{n-1})}^{u_n}$. Intuitively, the node label determines the respective weights of the left and right sons in the definition of the measure. In particular the measure μ_p is the one obtained by letting $(p_x^0, p_x^1) = (p, 1-p)$ for all letters x in the alphabet.

One can easily show that, with this more general definition of measure, the results presented in this document (provided that definitions of the games are modified accordingly) remain correct.

Following Proposition 12 there are two natural questions.

- Is the class of qualitative tree languages the same for each distribution μ_p with $0 < p < 1$?
- Is the class of qualitative tree languages the same for the above variant with probability values p_a^0, p_a^1 for all letters in the alphabet.

From the statement of Proposition 12[12] we conjecture that the answer to these questions is negative but we leave it open.

9.3 Qualitative Tree Languages & Applications

Unsurprisingly, there remain several open questions on qualitative tree languages. Some of them are purely theoretical (see above) but the most pressing one concerns potential applications of this concept.

A quick answer to this latter challenge is to rely on the tight connection between qualitative tree languages and Markov decision processes as exposed in Section 5.5. As these two objects are essentially the same one but considered from two different perspectives (the qualitative tree languages being a sort of unfolding of a finite MDP) one can for instance rely on the modelling work made using MDP (see *e.g.* Chapter 10 of [BK08] for many valuable examples) to argue that qualitative tree languages are equally useful for such a purpose. However, the setting of MDP seems simpler for modelling purpose as it is closer to real systems.

Another way to answer this question would be to design a logic whose expressive power (on node-labelled binary trees) is captured by qualitative tree languages (*i.e.* with any formula φ one can construct a qualitative tree language L_φ such that $L_\varphi = \{t \mid t \models \varphi\}$). Of course, as qualitative tree languages are not closed under complement, in order to define a reasonable logic (possibly with negation), it could make sense to either consider a fragment or to extend the class of qualitative tree languages for that purpose (while preserving decidability results).

[12]Note that Proposition 12 is simpler to obtain than the question we discuss now, because it deals with a fixed automaton while the present problem allows to use another automaton (it is a question about the whole class of qualitative tree languages rather than a question on a specific automaton).

On a related perspective, one should also mention that our original hope when defining qualitative tree languages was to use them to deal with decidability of standard problems (model-checking, satisfiability) for Alternating Temporal Logic (ATL) [AHK02] and its variants. In a nutshell, Alternating Temporal Logic is a logic designed to express properties of multi-agent concurrent systems. It is quite similar to CTL but one replaces the unconstrained path quantifiers of CTL with constrained path quantifiers: for instance, while the CTL formula **AG** φ asserts that the path property φ is inevitable (*i.e.* it must hold on all path), the ATL formula $\langle\langle A\rangle\rangle\varphi$ asserts that φ can be enforced by the subset A of agents, *i.e.* the coalition of agents A has a winning strategy to ensure that property φ is satisfied whatever do the other agents. Following the original work on ATL there were many other logics introduced *e.g.* ATL* [AHK02], Game Logic [AHK02], Strategy logic [CHP10; MMV10], QD_μ [Pin07] or ATL with strategy context [LLM10], and for most of these logics the model-checking problem can be reduced to check the emptiness of a tree automaton (in a nutshell the key idea is to encode strategies of the players as a tree and to use an automaton to check that under these strategies the formula is true). One can argue that a lack of these logic is that their semantics is a "sure" one while an "almost sure" semantics would be more natural as the systems that one model-checks are concurrent (and surely ensuring a property in such a system is quite often much more unrealistic than almost surely ensuring a property). A challenging problem is to design a tree automaton formalism that permits to reduce the model-checking of an ATL-like logic with an almost-sure semantics to the emptiness problem for an automaton in this class.

Concerning applications for the probabilistic setting, it is not clear yet what are the potential applications. Of course due to their incomparability with both regular and qualitative tree languages, we cannot simply extend existing applications, but this is also encouraging as it suggests potentially new applications. The hard part being that it mixes (for the qualitative semantics) two orthogonal notions of measure: the one on the run and the one the branches; if the one on branches has a simple interpretation (one looks at all possible executions of a system) the one on runs is more tricky to interpret (in a sense it speaks of all output of a machine processing the unfolding of a system).

9.4 Mostowski-Like Hierarchies

When dealing with automata models equipped with a parity condition one can define the so-called Mostowski hierarchies [Mos85] by looking at those tree languages accepted when one uses only colours in the interval $[i, j]$ (such a language is said to be i, j-*feasible*; one easily sees that we can always assume that $i = 0$ or $i = 1$; Büchi condition corresponds to the case where $i = 0$ and $j = 1$ while co-Büchi condition is the case where $i = 1$ and $j = 2$). The Mostowski hierarchy is strict for all standard models of tree automata: deterministic [Wag77] (the hierarchy is strict already for infinite words when one consider deterministic automata), non-deterministic [Niw86] and alternating [Arn99] (combined with [Bra98; Len96]). Hence, allowing more colours gives more power to the automata regardless on their nature (deterministic, non-deterministic or alternating). However, deciding for a given regular tree language whether it is i, j-feasible is an open problem in most cases: the only known cases are the ones where the regular tree language is recognised by a deterministic tree automaton [NW05], the one of $0, 1$-feasibility (*i.e.* co-Büchi) [Col13] and the one asking whether a language and its complement are both $1, 2$-feasible (*i.e.* Büchi) [CKLV13].

For the setting of qualitative tree languages one can first ask whether the corresponding Mostowski hierarchy is strict. And one can also consider the i, j-feasibility problem. Of special interest here, is for instance whether one can adapt the notions developed by Colcombet and Löding in [CL08] where they relate (in the setting of regular tree languages) the i, j-feasibility problem with boundedness questions on so-called regular cost functions: a first task is of course to adapt the main concepts; another one is for instance to see how the notion of *guidable* automata [CL08] behaves in the setting

of qualitative tree languages.

In a same flavour one can ask similar questions for the languages defined with cardinality constraints. In the particular case of languages of the form $L^{\text{Rej}}_{\leqslant \text{Count}}(\mathcal{A})$ and $L^{\text{Rej}}_{\text{Fin}}(\mathcal{A})$ one can also wonder whether all levels of the Mostowski hierarchy (for regular tree languages) are reached (so far we only proved in Section 3.2.3 that one goes beyond Büchi languages; a more general result seems technically more challenging). Also note that, in the case of languages of the form $L^{\text{Acc}}_{\infty}(\mathcal{A})$ or $L^{\text{Acc}}_{Uncount}(\mathcal{A})$ one could also have a strict Mostowski hierarchy while collapsing to Büchi regular tree languages.

Concerning tree languages defined by automata with topological bigness constraints, the same kind of problems can be tackled: Is the Mostowski hierarchy infinite? Are all levels of the Mostowski hierarchy (for regular tree languages) reached? Can one develop tools to decide i, j-feasibility?

9.5 Toward a Simulation Theorem for Büchi Alternating Qualitative Tree Automata

In Remark 42 we noted that from the undecidability result of deciding whether $L^{=1}(\mathcal{A})$ for an alternating co-Büchi tree automaton $\mathcal{A}$ (Theorem 56) follows the impossibility of having an *effective* simulation theorem for co-Büchi alternating qualitative tree automata. For the case of Büchi alternating qualitative tree automaton the picture might be completely different and we actually conjecture that, relying on the positionality result for stochastic turned based Büchi game on chronological arenas (Theorem 55), one can obtain such a simulation theorem and conclude that languages accepted by Büchi alternating qualitative tree automata are indeed qualitative tree languages (*i.e.* are accepted by parity nondeterministic qualitative tree automata).

9.6 Probabilistic Tree Automata: Emptiness of Languages of the Form $L^{=1}(\mathcal{A})$

In Theorem 59 we have seen that checking whether $L^{=1}(\mathcal{A}) = \varnothing$ is an undecidable problem when $\mathcal{A}$ is equipped with a co-Büchi acceptance condition. But the decidability status of this problem when $\mathcal{A}$ is a Büchi (or even weaker) tree automaton is still open and we have some hope that the answer could be positive. Here of course the main difficulty is that one has to deal with two quantifiers of inherently different natures: one is universal (all branches in a run must be accepting) while the other is probabilistic (almost all runs must be accepting) and it is unclear how to handle them simultaneously (which was the main idea when dealing with languages of the form $L^{=1}_{\text{Qual}}(\mathcal{A})$ thanks to Proposition 19).

In the setting of probabilistic tree automata with the usual semantic for runs (*i.e.* all branche must be accepting) a naive definition for an acceptance game would to no longer choose the successor at random but to give this choice to a second player Abélard. However this game does not faithfully reflect the acceptation of the automaton.

Indeed, consider the reachability probabilistic tree automaton $\mathcal{A} = \langle \{a\}, \{q_0, q_f\}, q_0, \delta, \{q_f\} \rangle$ with $\delta(q_0, a, q_0, q_0) = \frac{3}{4}$, $\delta(q_0, a, q_f, q_f) = \frac{1}{4}$, $\delta(q_f, a, q_f, q_f) = 1$ and 0 otherwise. Consider the tree t_a where all nodes are labeled by a. It can be shown that the measure of the set of accepting runs of $\mathcal{A}$ over t_a has measure $\frac{1}{3}$. Hence the tree t_a does not belong to $L^{=1}(\mathcal{A})$.

Now consider the naive acceptance game for $\mathcal{A}$ on t_a. Intuitively in this game, player random chooses a transition (p, q_0, q_1) and Abelard chooses to proceed either to q_0 or q_1. The set of states is $\{0,1\}^* \times \{\theta_0, \theta_1, \theta_f\}$ with $\theta_0 = (q_0, q_0, q_0)$, $\theta_1 = (q_0, q_f, q_f)$ and $\theta_f = (q_f, q_f, q_f)$, the initial state is (ε, θ_0) and the actions of Abélard are in $\{0,1\}$. For $x \in \{0,1\}$, the transition function is such that for all $w \in \{0,1\}^*$ we have: $\delta((w, \theta_0), x)$ is the probability distribution assigning $\frac{3}{4}$ to (wx, θ_0) and $\frac{1}{4}$ to (wx, θ_1), $\delta((w, \theta_1), x) = \delta((w, \theta_f), x)$ is the probability distribution assigning 1 to (wx, θ_f). The objective $\mathcal{O}$ is the set of plays containing θ_f.

It is easy to check that the strategy of Abélard has no influence on the value of the game. If fact for any fixed strategy, the game is equivalent to the Markov chain depicted in Figure 45 which fulfils its objective with probability 1.

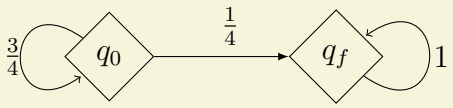


Figure 45 – A Markov chain equivalent to the naive acceptance game of $\mathcal{A}$ on t_a.

So this small example proves that the naive acceptance game does not work for languages of the form $L^{=1}(\mathcal{A})$. So there are two natural connected questions.

- Can one design an acceptance game for languages of the form $L^{=1}(\mathcal{A})$? If so, can one design an emptiness game for languages of the form $L^{=1}(\mathcal{A})$?
- Is the following problem decidable when $\mathcal{A}$ is equipped with a Büchi condition (resp. reachability, safety): does $L^{=1}(\mathcal{A}) = \varnothing$.

9.7 Decidability Frontier for Imperfect Information Games

The decidability frontier is fairly clear in the case of randomised strategies[13].

The situation is not as clear in the case of pure strategies (which is actually closely related to the one where one allows randomised strategies that *does not* observe the actions; see [CD12]). Indeed, the picture is clear when Abélard is perfectly informed. If he is more informed than Éloïse we also drew a complete picture but matching lower bounds are missing (the upper bound is 2-ExpTime while the lower bound in ExpTime-hard). In the general case (Éloïse and Abélard both imperfectly informed without any extra assumption) we can easily prove the following semi-decidability results (that comes from the fact that positive winning in reachability game can always be achieved using *finite* memory strategies).

Theorem 62 [CLS12]

The following problems are semi-decidable.

- Does Éloïse positively win using a pure strategy in a given concurrent *reachability* game with imperfect information?
- Does Éloïse almost surely win using a pure strategy in a given concurrent *Büchi* game with imperfect information?

Hence, a natural question is whether the above mentioned problems are decidable. The results in [CD12] on the memory needed to win in such games (non-elementary size) seem to indicate that these problems might not be decidable. Of course some particular cases are of special interest, *e.g.* the one where Éloïse is more informed than Abélard, or, even more restrictive, the case where she

[13] One should stress here that the notion of finite memory randomised strategies used in this document differs from the one used in [BGG09] where one allows to update the memory *randomly*.

has perfect information while Abélard is blind (*i.e.* all states are indistinguishable for him). While we conjecture that the general case lead to undecidability we expect that some special instances (in particular the above mentioned ones) may lead to some decidability results.

List of Symbols & Notations

$(\cdot)^{\perp}$	Map that takes an applicative term and replaces each non-terminal, together with its arguments, by $\perp$, 27
$L(\mathcal{A})$	Language accepted by automaton $\mathcal{A}$, 105
$L(\mathcal{A})$	Language accepted by automaton $\mathcal{A}$, 57
$M[t/x]$	Term obtained by replacing variable $x : \tau$ by a term $t : \tau$ in the term M, 26
$Post^*_{\mathcal{A}}(Init)$	Set of configurations reachable by $\mathcal{A}$ from $Init$, 86
$Pre^*_{\mathcal{A}}(Bad)$	Set of all configurations of $\mathcal{A}$ from which there exists a path to an element in Bad, 82
$\mathcal{F}_{\text{Br}}$	σ-algebra generated by the set of cones in a tree, 117
ASub(t)	Argument subterms of t, 25
AccBr(ρ)	Accepting branches in the run ρ, 117
n-CPDALTS	Set of Lts generated by order-n CPDA, 48
CPDALTS	Set of Lts generated by CPDA, 48
n-CPDATrees	Set of trees generated by order-n CPDA, 48
CPDATrees	Set of trees generated by CPDA, 48
n-CaucalLTS	Set of trees in the n-th level of Caucal Lts hierarchy, 49
n-CaucalTrees	Set of trees in the n-th level of Caucal tree hierarchy, 49
μ_{φ}	Probability measure on $\mathcal{F}_P$ induced by strategy φ, 121
n-HOPDALTS	Set of Lts generated by order-n HOPDA, 48
HOPDALTS	Set of Lts generated by HOPDA, 48
n-HOPDATrees	Set of trees generated by order-n HOPDA, 48
HOPDATrees	Set of trees generated by HOPDA, 48
$L^{\text{Acc}}_{\infty}(\mathcal{A})$	Language accepted by $\mathcal{A}$ with the constraint that there are infinitely many accepting branches in the run, 109

$L^{\text{Acc}}_{Uncount}(\mathcal{A})$ Language accepted by $\mathcal{A}$ with the constraint that there are uncountably many accepting branches in the run, 109

$L^{\text{Rej}}_{\leqslant\text{Count}}(\mathcal{A})$ Language accepted by $\mathcal{A}$ with the constraint that there are at most countably many rejecting branches in the run, 108

$L^{\text{Rej}}_{\text{Fin}}(\mathcal{A})$ Language accepted by $\mathcal{A}$ with the constraint that there are finitely many rejecting branches in the run, 108

$L_{\text{Qual}}(\mathcal{A})$ Language qualitatively accepted by automaton $\mathcal{A}$, 117

$L_{\text{Pos}}(\mathcal{A})$ Language positively accepted by automaton $\mathcal{A}$, 124

$\overset{w}{\Longrightarrow}$ Relation $\xrightarrow{L_w}$ where $L_w \overset{def}{=} \lambda^* a_1 \lambda^* \cdots \lambda^* a_n \lambda^*$, 23

$\text{Path}^{=1}(L)$ Set of trees with almost all their branch labels in L, 145

Plays Set of all plays, 120

$\text{Pref}(L)$ Tree in $\text{Trees}^{\infty}()$ containing all prefixes of words in L, 31

$\text{Sub}(t)$ Subterms of t, 25

$\text{Val}_{\mathcal{A}}(t)$ Value of tree t wrt automaton $\mathcal{A}$, 123

$\text{Terms}_{\tau}(X)$ Set of applicative terms of type τ generated from the ranked alphabet X, 25

collapse Perform the *collapse* operation, 39

$\text{Cone}(u)$ Cone going through node u, 116

$\text{Cyl}(\lambda)$ Cylinder associated with partial run λ, 143

$\text{Cyl}(\lambda)$ Cylinder associated with partial play λ, 121

$\overrightarrow{A}$ Directions built from alphabet A, 22

$\widetilde{s}$ Well-bracketed word of depth n built from order-n stack with links s by forgetting about links, 71

$\mathbb{G}_{\mathcal{A},t}$ Acceptance game for automaton $\mathcal{A}$ and tree t, 106

id Identity operation on stacks, 39

μ Lebesgue mesure on trees, 117

$\mathcal{L}_{\mathcal{A}}$ Deterministic Lts associated with the CPDA $\mathcal{A}$, 40

$\mathcal{B}(U)$ Set of branches that have infinitely many prefixes in U, 116

$\mathcal{F}_P$ σ-algebra generated by the set of cylinders, 121

$[\![\mathcal{S}]\!]$ Value tree of recursion scheme $\mathcal{S}$, 28

μ_t Probability measure over runs induced by tree t, 143

$\preccurlyeq$ Approximation ordering on terms, 27

$\mathrm{ord}(\tau)$	Order of type τ, 24
$Outcomes(s_0, \varphi_E, \varphi_A)$	Set of all possible plays when the game starts on s_0 and when Éloïse and Abélard plays respectively accordingly to φ_E and φ_A, 128
$L^{=1}(\mathcal{A})$	Language of trees almost-surely accepted by probabilistic automaton $\mathcal{A}$, 144
$L^{=1}_{\mathrm{Qual}}(\mathcal{A})$	Language of trees almost-surely qualitatively accepted by probabilistic automaton $\mathcal{A}$, 144
pop_i	Remove the topmost $(i-1)$-stack, 37
$+\!\!+$	$s+\!\!+\, t$: push t on top of s, 37
$\Pr^{\varphi_E,\varphi_A}_{s_0}$	Probability measure induced by strategies φ_E and φ_A, 128
$push_1^{a,\ell}$	Push the symbol a on top of the top 1-stack and attach an ℓ-link to it, 39
$push_1^a$	Push the symbol a on top of the topmost 1-stack, 38
$\rightarrow_{\mathcal{S}}$	Rewriting relation induced by the recursion scheme $\mathcal{S}$, 26
n-SafeLTS	Set of Lts generated by order-n safe labelled recursion schemes, 48
SafeLTS	Set of Lts generated by labelled safe recursion schemes, 48
n-SafeTrees	Set of trees generated by order-n safe recursion schemes, 48
SafeTrees	Set of trees generated by safe recursion schemes, 48
t_u	Subtree of t rooted at u, 22
$target(s)$	Partial function reflecting the link structure of s, 71
$\mathrm{Terms}(A)$	Set of terms over A, 22
top_i	Return the topmost $(i-1)$-stack, 37
$\mathrm{Tree}(\mathcal{L})$	Tree associated with the Lts with silent transitions $\mathcal{L}$, 23
$\mathrm{Tree}^{\perp}(\mathcal{L})$	Tree associated with the Lts with silent transitions $\mathcal{L}$ with diverging behaviours indicated by a $\perp$-child, 24
$\mathrm{Trees}^{\infty}(A)$	Set of trees with directions in A, 22
$\|t\|_{\varphi}$	Set of nodes of tree t described by formula φ, 57
n-UnsafeLTS	Set of Lts generated by order-n labelled recursion schemes, 48
UnsafeLTS	Set of Lts generated by labelled recursion schemes, 48
n-UnsafeTrees	Set of trees generated by order-n (possibly unsafe) recursion schemes, 48
UnsafeTrees	Set of trees generated by recursion schemes, 48
$f : \tau$	f has type τ, 25
rew_1^a	Rewrite the top symbol to a without changing the target of the link, 39
t_{φ}	Tree obtained from t by marking those nodes where φ holds, 57

Bibliography

Personal References Cited in This Document

[BCHS12] Christopher H. Broadbent, Arnaud Carayol, Matthew Hague, and Olivier Serre. ``A Saturation Method for Collapsible Pushdown Systems''. In: *Proceedings of the 39th International Colloquium on Automata, Languages, and Programming (ICALP 2012)*. Vol. 7392. Lecture Notes in Computer Science. Springer-Verlag, 2012, pp. 165–176 (cited on pp. 7, 15, 71, 72, 83–85, 91).

[BCHS13] Christopher H. Broadbent, Arnaud Carayol, Matthew Hague, and Olivier Serre. ``C-SHORe: a collapsible approach to higher-order verification''. In: *Proceedings of the 18th ACM SIGPLAN International Conference on Functional Programming (ICFP 2013)*. ACM, 2013, pp. 13–24 (cited on pp. 7, 15, 71, 85, 87).

[BCOS10] Christopher H. Broadbent, Arnaud Carayol, C.-H. Luke Ong, and Olivier Serre. ``Recursion Schemes and Logical Reflexion''. In: *Proceedings of the 25th Annual IEEE Symposium on Logic in Computer Science (LiCS 2010)*. IEEE Computer Society, 2010, pp. 120–129 (cited on pp. 7, 15, 22, 59, 60, 71, 72, 74, 76, 80, 81).

[BSW03] Alexis-Julien Bouquet, Olivier Serre, and Igor Walukiewicz. ``Pushdown Games with Unboundedness and Regular Conditions''. In: *Proceedings of the 23rd International Conference on Foundations of Software Technology and Theoretical Computer Science (FST&TCS 2003)*. Vol. 2914. Lecture Notes in Computer Science. Springer-Verlag, 2003, pp. 88–99 (cited on p. 92).

[CCS14] Arnaud Carayol, Didier Caucal, and Olivier Serre. ``Reachability Games Generated by Graph Grammars''. 44 pages. Unpublished. 2014 (cited on p. 91).

[CHS11] Arnaud Carayol, Axel Haddad, and Olivier Serre. ``Qualitative Tree Languages''. In: *Proceedings of the 26th IEEE Symposium on Logic in Computer Science (LiCS 2011)*. IEEE Computer Society, 2011, pp. 13–22 (cited on pp. 7, 99, 101, 117–120, 122–125, 142–145, 147–150).

[CHS14a] Arnaud Carayol, Axel Haddad, and Olivier Serre. ``Counting Branches in Trees Using Games''. 34 pages. Submitted. 2014 (cited on pp. 7, 99, 108, 110–117).

[CHS14b] Arnaud Carayol, Axel Haddad, and Olivier Serre. ``Randomisation in Automata on Infinite Trees''. In: *ACM Transactions on Computational Logic* 15.3 (2014), p. 24 (cited on pp. 7, 99, 101).

[CLS12] Arnaud Carayol, Christof Löding, and Olivier Serre. ``Pure strategies in Imperfect Information Stochastic Games''. 2012 (cited on pp. 7, 132, 133, 154).

[CMHOS08] Arnaud Carayol, Antoine Meyer, Matthew Hague, C.-H. Luke Ong, and Olivier Serre. ``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, 2008, pp. 193–204 (cited on pp. 7, 15, 22, 59, 71, 72, 76, 79, 80).

[CS12a] Arnaud Carayol and Olivier Serre. ``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, 2012, pp. 165–174 (cited on pp. 7, 15, 22, 29, 30, 50, 51, 66, 68, 71, 72, 76, 81).

[CS12b] Arnaud Carayol and Olivier Serre. ``Higher-order recursion schemes and their automata models''. In: *Automata: from Mathematics to Applications*. Ed. by Jean-Éric Pin. European Mathematical Society, 2012. To appear (cited on pp. 7, 15).

[FPS13] Nathanaël Fijalkow, Sophie Pinchinat, and Olivier Serre. ``Emptiness Of Alternating Tree Automata Using Games With Imperfect Information''. In: *Proceedings of the 33rd International Conference on Foundations of Software Technology and Theoretical Computer Science (FST&TCS 2013)*. Vol. 24. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2013, pp. 299–311 (cited on pp. 7, 100, 133, 138, 141).

[GS09] Vincent Gripon and Olivier Serre. ``Qualitative Concurrent Stochastic Games with Imperfect Information''. In: *Proceedings of the 36th International Colloquium on Automata, Languages, and Programming (ICALP 2009)*. Vol. 5556. Lecture Notes in Computer Science. A revised corrected version of this work can be found at `http://arxiv.org/abs/0902.2108`. Springer-Verlag, 2009, pp. 200–211 (cited on pp. 7, 102, 103, 130).

[HMOS08] Matthew Hague, Andrzej S. Murawski, C.-H. Luke Ong, and Olivier Serre. ``Collapsible Pushdown Automata and Recursion Schemes''. In: *Proceedings of the 23rd Annual IEEE Symposium on Logic in Computer Science (LiCS 2008)*. IEEE Computer Society, 2008, pp. 452–461 (cited on pp. 7, 15, 17, 21, 50, 52, 54, 56, 71, 72, 76, 78).

[Ser03] Olivier Serre. ``Note on winning positions on pushdown games with omega-regular winning conditions''. In: *Information Processing Letters* 85 (2003), pp. 285–291 (cited on pp. 71, 72).

[Ser04a] Olivier Serre. ``Contribution à l'étude des jeux sur des graphes de processus à pile''. PhD thesis. Université Paris 7, 2004 (cited on pp. 71, 72, 80, 92).

[Ser04b] Olivier Serre. ``Games with Winning Conditions of High Borel Complexity''. In: *Proceedings of the 31st International Colloquium on Automata, Languages, and Programming (ICALP 2004)*. Vol. 3142. Lecture Notes in Computer Science. Springer-Verlag, 2004, pp. 1150–1162 (cited on pp. 92–94).

[Ser06a] Olivier Serre. ``Games with winning conditions of high Borel complexity''. In: *Theoretical Computer Science* 350.2-3 (2006), pp. 345–372 (cited on pp. 92–94).

[Ser06b] Olivier Serre. ``Parity Games Played on Transition Graphs of One-Counter Processes''. In: *Proceedings of the 9th International Conference on Foundations of Software Science and Computational Structures (FoSSaCS 2006)*. Vol. 3921. Lecture Notes in Computer Science. Springer-Verlag, 2006, pp. 337–351 (cited on p. 72).

Personal References Not Cited in This Document

[ALMS08] Benjamin Aminof, Axel Legay, Aniello Murano, and Olivier Serre. ``μ-calculus Pushdown Module Checking with Imperfect State Information''. In: *Proceedings of the 5th IFIP International Conference on Theoretical Computer Science (IFIP TCS 2008)*. Vol. 273. IFIP. Springer-Verlag, 2008, pp. 333–348.

[ALMSV13] Benjamin Aminof, Axel Legay, Aniello Murano, Olivier Serre, and Moshe Y. Vardi. ``Pushdown module checking with imperfect information''. In: *Information and Computation* 223 (2013), pp. 1–17.

[BLS06] Vince Bárány, Christof Löding, and Olivier Serre. ``Regularity Problems for Visibly Pushdown Languages''. In: *Proceedings of the 25th Symposium on Theoretical Aspects of Computer Science (STACS 2006)*. Vol. 3884. Lecture Notes in Computer Science. Springer-Verlag, 2006, pp. 420–431.

[BS12] Dietmar Berwanger and Olivier Serre. ``Parity games on undirected graphs''. In: *Information Processing Letters* 112.23 (2012), pp. 928–932.

[BS13] Dietmar Berwanger and Olivier Serre. ``Introduction''. In: *Technique et Science Informatiques* 32.9-10 (2013), pp. 907–908.

[GMSZ08] Blaise Genest, Anca Muscholl, Olivier Serre, and Marc Zeitoun. ``Tree Pattern Rewriting Systems''. In: *Proceedings of the 6th International Symposium on Automated Technology for Verification and Analysis (ATVA 2008)*. Vol. 5311. Lecture Notes in Computer Science. Springer-Verlag, 2008, pp. 332–346.

[LLS07] Christof Löding, Carsten Lutz, and Olivier Serre. ``Propositional dynamic logic with recursive programs''. In: *Journal of Logic and Algebraic Programming* 73.1-2 (2007), pp. 51–69.

[LMS04] Christof Löding, P. Madhusudan, and Olivier Serre. ``Visibly Pushdown Games''. In: *Proceedings of the 24th International Conference on Foundations of Software Technology and Theoretical Computer Science (FST&TCS 2004)*. Vol. 3328. Lecture Notes in Computer Science. Springer-Verlag, 2004, pp. 408–420.

[LS06] Christof Löding and Olivier Serre. ``Propositional Dynamic Logic with Recursive Programs''. In: *Proceedings of the 9th International Conference on Foundations of Software Science and Computational Structures (FoSSaCS 2006)*. Vol. 3921. Lecture Notes in Computer Science. Springer-Verlag, 2006, pp. 292–306.

[Ser02] Olivier Serre. ``Vectorial Languages and Linear Temporal Logic''. In: *Proceedings of the 2nd IFIP International Conference on Theoretical Computer Science (IFIP TCS 2002)*. Vol. 223. IFIP Conference Proceedings. Kluwer, 2002, pp. 576–587.

[Ser04c] Olivier Serre. ``Vectorial languages and linear temporal logic''. In: *Theoretical Computer Science* 310.1-3 (2004), pp. 79–116.

Other References Cited in This Document

[Aeh06] Klaus Aehlig. ``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)"*. Vol. 4207. Lecture Notes in Computer Science. Springer-Verlag, 2006, pp. 104–118 (cited on pp. 22, 88).

[AH00] Luca de Alfaro and Thomas A. Henzinger. ``Concurrent Omega-Regular Games''. In: *Proceedings of the 15th Annual IEEE Symposium on Logic in Computer Science (LiCS 2000)*. IEEE Computer Society, 2000, pp. 141–154 (cited on pp. 126, 130).

[AH92] Robert J. Aumann and Sergiu Hart, eds. *Handbook of Game Theory with Economic Applications*. Vol. 1. Elsevier, 1992 (cited on pp. 102, 104).

[AHK02] Rajeev Alur, Thomas A. Henzinger, and Orna Kupferman. ``Alternating-time temporal logic''. In: *Journal of the Association for Computing Machinery (ACM)* 49.5 (2002), pp. 672–713 (cited on p. 152).

[AHK07] Luca de Alfaro, Thomas A. Henzinger, and Orna Kupferman. ``Concurrent reachability games''. In: *Theoretical Computer Science* 386.3 (2007), pp. 188–217 (cited on pp. 126, 129, 130).

[Ale16] Pavel Sergeevich Alexandrov. ``Sur la puissance des ensembles mesurables''. In: *Comptes-Rendus de l'Académie des Sciences de Paris* 162 (1916), pp. 323–325 (cited on p. 109).

[AM95] Robert .J. Aumann and Michael Maschler. *Repeated games with incomplete information*. M.I.T. Press, 1995 (cited on p. 104).

[AMO05] Klaus Aehlig, Jolie de Miranda, and C.-H. Luke Ong. ``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)*. Vol. 3411. Lecture Notes in Computer Science. Springer-Verlag, 2005, pp. 490–501 (cited on pp. 21, 54).

[AN01] André Arnold and Damian Niwiński. *Rudiments of mu-calculus*. Vol. 146. Studies in Logic and the Foundations of Mathematics. Elsevier, 2001 (cited on pp. 13, 21, 80, 100).

[Arn99] André Arnold. ``The μ-calculus alternation-depth hierarchy is strict on binary trees''. In: *RAIRO Theoretical Informatics and Applications* 33.4/5 (1999), pp. 329–340 (cited on pp. 99, 152).

[Ati10] Mohamed Faouzi Atig. ``Global Model Checking of Ordered Multi-Pushdown Systems''. In: *Proceedings of the 30th International Conference on Foundations of Software Technology and Theoretical Computer Science (FST&TCS 2010)*. Vol. 8. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2010, pp. 216–227 (cited on p. 84).

[Bar84] Henk P. Barendregt. *The Lambda Calculus Its Syntax and Semantics*. Revised. Vol. 103. North Holland, 1984 (cited on p. 27).

[Bau01] Heinz Bauer. *Measure and Integration Theory*. Walter de Gruyter, 2001 (cited on pp. 117, 121, 144).

[Bau96] Heinz Bauer. *Probability Theory*. Walter de Gruyter, 1996 (cited on p. 117).

[BBG08] Christel Baier, Nathalie Bertrand, and Marcus Größer. ``On Decision Problems for Probabilistic Büchi Automata''. In: *Proceedings of the 11th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS 2008)*. Vol. 4962. Lecture Notes in Computer Science. Springer-Verlag, 2008, pp. 287–301 (cited on pp. 101, 142, 146–148).

[BC10] Laurent Braud and Arnaud Carayol. ``Linear Orders in the Pushdown Hierarchy''. In: *Proceedings of the 37th International Colloquium on Automata, Languages, and Programming (ICALP 2010)*. Vol. 6199. Lecture Notes in Computer Science. Springer-Verlag, 2010, pp. 88–99 (cited on p. 50).

[BCDHR08] Dietmar Berwanger, Krishnendu Chatterjee, Laurent Doyen, Thomas A. Henzinger, and Sangram Raje. ``Strategy Construction for Parity Games with Imperfect Information''. In: *Proceedings of the 19th International Conference on Concurrency Theory (CONCUR 2008)*. Vol. 5201. Lecture Notes in Computer Science. Springer-Verlag, 2008, pp. 325–339 (cited on p. 102).

[BCWDHR09] Dietmar Berwanger, Krishnendu Chatterjee, Martin De Wulf, Laurent Doyen, Thomas A. Henzinger, and Jean-François Raskin. ``Alpaga: A Tool for Solving Parity Games with Imperfect Information''. In: *Proceedings of 15th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2010)*. Vol. 5505. Lecture Notes in Computer Science. Springer-Verlag, 2009, pp. 58–61 (cited on pp. 100, 102, 134).

[BD08] Dietmar Berwanger and Laurent Doyen. ``On the Power of Imperfect Information''. In: *Proceedings of the 28th International Conference on Foundations of Software Technology and Theoretical Computer Science (FST&TCS 2008)*. Vol. 2. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2008, pp. 73–82 (cited on p. 102).

[BEM97] Ahmed Bouajjani, Javier Esparza, and Oded Maler. ``Reachability Analysis of Pushdown Automata: Application to Model-Checking''. In: *Proceedings of the 8th International Conference on Concurrency Theory (CONCUR 1997)*. Vol. 1243. Lecture Notes in Computer Science. Springer-Verlag, 1997, pp. 135–150 (cited on p. 84).

[Ben69] Michèle Benois. ``Parties rationnelles du groupe libre''. In: *Comptes-Rendus de l'Académie des Sciences de Paris, Série A* 269 (1969), pp. 1188–1190 (cited on p. 84).

[BG05] Christel Baier and Marcus Größer. ``Recognizing omega-regular Languages with Probabilistic Automata''. In: *Proceedings of the 20th Annual IEEE Symposium on Logic in Computer Science (LiCS 2005)*. IEEE Computer Society, 2005, pp. 137–146 (cited on pp. 101, 142, 144).

[BGB12] Christel Baier, Marcus Größer, and Nathalie Bertrand. ``Probabilistic ω-automata''. In: *Journal of the Association for Computing Machinery (ACM)* 59.1 (2012), p. 1 (cited on pp. 101, 103, 131–133, 142).

[BGG09] Nathalie Bertrand, Blaise Genest, and Hugo Gimbert. ``Qualitative Determinacy and Decidability of Stochastic Games with Signals''. In: *Proceedings of Logic in Computer Science (LiCS 2009)*. IEEE Computer Society, 2009, pp. 319–328 (cited on pp. 102, 103, 130–132, 154).

[BGR11] Vince Bárány, Erich Grädel, and Sasha Rubin. ``Automata-based presentations of infinite structures''. In: *Finite and Algorithmic Model Theory*. Vol. 379. London Mathematical Society Lecture Notes Series. Cambridge University Press, 2011, pp. 1–76 (cited on p. 20).

[BK08] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. The MIT Press, 2008 (cited on pp. 97, 151).

[BKN13] Thomas Brázdil, Antonin Kucera, and Peter Novotný. ``Determinacy in Stochastic Games with Unbounded Payoff Functions''. In: *Proceedings of the 8th International Doctoral Workshop on Mathematical and Engineering Methods in Computer Science (MEMICS'12)*. Vol. 7721. Lecture Notes in Computer Science. Springer-Verlag, 2013, pp. 94–105 (cited on p. 101).

[BKR10] Vince Bárány, Lukasz Kaiser, and Alexander Rabinovich. ``Expressing Cardinality Quantifiers in Monadic Second-Order Logic over Trees''. In: *Fundamenta Informaticae* 100 (2010), pp. 1–18 (cited on pp. 99, 108).

[BM04] Ahmed Bouajjani and Antoine Meyer. ``Symbolic Reachability Analysis of Higher-Order Context-Free Processes''. In: *Proceedings of the 24th International Conference on Foundations of Software Technology and Theoretical Computer Science (FST&TCS 2004)*. Vol. 3328. Lecture Notes in Computer Science. Springer-Verlag, 2004, pp. 135–147 (cited on pp. 71, 72, 80, 86).

[BN95] Danièle Beauquier and Damian Niwiński. ``Automata on infinite trees with path counting constraints''. In: *Information and Computation* 120.1 (1995), pp. 117 –125 (cited on pp. 99, 108, 112, 114, 115).

[BNN91] Danièle Beauquier, Maurice Nivat, and Damian Niwiński. ``About the Effect of the Number of Successful Paths in an Infinite Tree on the Recognizability by a Finite Automaton with Büchi Conditions''. In: *Proceedings of the 8th International Conference on Fundamentals of Computation Theory (FCT'91)*. Vol. 529. Lecture Notes in Computer Science. Springer-Verlag, 1991, pp. 136–145 (cited on pp. 99, 108).

[BR02] Thomas Ball and Sriram K. Rajamani. ``The SLAM Project: Debugging System Software via Static Analysis''. In: *Proceedings of the 29th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 2002)*. ACM, 2002, pp. 1–3 (cited on p. 16).

[Bra98] Julian C. Bradfield. ``The Modal μ-Calculus Alternation Hierarchy is Strict''. In: *Theoretical Computer Science* 195.2 (1998), pp. 133–153 (cited on pp. 99, 152).

[Bro12] Christopher H. Broadbent. ``The Limits of Decidability for First Order Logic on CPDA Graphs''. In: *Proceedings of the 29th Symposium on Theoretical Aspects of Computer Science (STACS 2012)*. Vol. 14. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2012, pp. 589–600 (cited on p. 56).

[Büc77] J. Richard Büchi. ``Using Determinancy of Games to Eliminate Quantifiers''. In: *Proceedings of the first International Conference on Fundamentals of Computation Theory (FCT'77)*. Vol. 56. Lecture Notes in Computer Science. Springer-Verlag, 1977, pp. 367–378 (cited on p. 98).

[Cac02] Thierry Cachat. ``Uniform Solution of Parity Games on Prefix-Recognizable Graphs''. In: *4th International Workshop on Verification of Infinite-State Systems, Infinity 2002*. Vol. 68. Electronic Notes in Theoretical Computer Science. Elsevier Science Publishers, 2002 (cited on pp. 71, 72).

[Cac03a] Thierry Cachat. ``Games on Pushdown Graphs and Extensions''. PhD thesis. RWTH Aachen, 2003 (cited on pp. 71, 84).

[Cac03b] Thierry Cachat. ``Higher Order Pushdown Automata, the Caucal Hierarchy of Graphs and Parity Games''. In: *Proceedings of the 30th International Colloquium on Automata, Languages, and Programming (ICALP 2003)*. Vol. 2719. Lecture Notes in Computer Science. Springer-Verlag, 2003, pp. 556–569 (cited on pp. 22, 71, 72, 80).

[Car05] Arnaud Carayol. ``Regular Sets of Higher-Order Pushdown Stacks''. In: *Proceedings of the 30th Symposium, Mathematical Foundations of Computer Science (MFCS 2005)*. Vol. 3618. Lecture Notes in Computer Science. Springer-Verlag, 2005, pp. 168–179 (cited on pp. 74, 80, 89).

[Car06] Arnaud Carayol. ``Automates infinis, logiques et langages''. PhD thesis. Université de Rennes 1, 2006 (cited on pp. 68, 74, 80).

[Cau02] Didier Caucal. ``On infinite terms having a decidable monadic theory''. In: *Proceedings of the 27th Symposium, Mathematical Foundations of Computer Science (MFCS 2002)*. Vol. 2420. Lecture Notes in Computer Science. Springer-Verlag, 2002, pp. 165–176 (cited on pp. 21, 49, 50, 54, 63, 71, 72, 90).

[Cau07] Didier Caucal. ``Deterministic graph grammars''. In: *Logic and Automata: History and Perspectives*. Vol. Text in Logics and Games 2. Amsterdam University Press, 2007, pp. 169–250 (cited on p. 91).

[CD12] Krishnendu Chatterjee and Laurent Doyen. ``Partial-Observation Stochastic Games: How to Win When Belief Fails''. In: *Proceedings of the 27th Annual IEEE Symposium on Logic in Computer Science (LiCS 2012)*. IEEE Computer Society, 2012, pp. 175–184 (cited on pp. 103, 131, 132, 154).

[CDGH10] Krishnendu Chatterjee, Laurent Doyen, Hugo Gimbert, and Thomas A. Henzinger. ``Randomness for Free''. In: *Proceedings of the 35th Symposium, Mathematical Foundations of Computer Science (MFCS 2010)*. Vol. 6281. Lecture Notes in Computer Science. Springer-Verlag, 2010, pp. 246–257 (cited on pp. 102, 131).

[CDH09] Krishnendu Chatterjee, Laurent Doyen, and Thomas A. Henzinger. ``Probabilistic Weighted Automata''. In: *Proceedings of the 20th International Conference on Concurrency Theory (CONCUR 2009)*. Vol. 5710. Lecture Notes in Computer Science. Springer-Verlag, 2009, pp. 244–258 (cited on p. 101).

[CDHR07] Krishnendu Chatterjee, Laurent Doyen, Thomas A. Henzinger, and Jean-François Raskin. ``Algorithms for Omega-Regular Games with Imperfect Information''. In: *Logical Methods in Computer Science* 3.3 (2007) (cited on pp. 102, 131, 132, 137, 138, 140).

[CDT02] Thierry Cachat, Jacques Duparc, and Wolfgang Thomas. ``Solving Pushdown Games with a Σ_3 Winning Condition''. In: *"Proceedings of Computer Science Logic, 16th Annual Conference of the EACSL (CSL 2002)"*. Vol. 2471. Lecture Notes in Computer Science. Springer-Verlag, 2002, pp. 322–336 (cited on p. 92).

[CE81] Edmund M. Clarke and E. Allen Emerson. ``Design and Synthesis of Synchronization Skeletons Using Branching-Time Temporal Logic''. In: *Proceedings of Logic of Programs*. Vol. 131. Lecture Notes in Computer Science. Springer-Verlag, 1981, pp. 52–71 (cited on p. 97).

[CHP10] Krishnendu Chatterjee, Thomas A. Henzinger, and Nir Piterman. ``Strategy logic''. In: *Information and Computation* 208.6 (2010), pp. 677–693 (cited on p. 152).

[Chu62] Alonzo Church. ``Logic, arithmetic and automata''. In: *Proceedings of the International Congress of Mathematicians*. 1962, pp. 23–35 (cited on p. 97).

[CJH04] Krishnendu Chatterjee, Marcin Jurdziński, and Thomas A. Henzinger. ``Quantitative stochastic parity games''. In: *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2004)*. SIAM, 2004, pp. 121–130 (cited on pp. 121, 124).

[CKLV13] Thomas Colcombet, Denis Kuperberg, Christof Löding, and Michael Vanden Boom. ``Deciding the weak definability of Büchi definable tree languages''. In: *"Proceedings of Computer Science Logic, 27th Annual Conference of the EACSL (CSL 2013)"*. Vol. 23. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2013, pp. 215–230 (cited on p. 152).

[CKS81] Ashok K. Chandra, Dexter C. Kozen, and Larry J. Stockmeyer. ``Alternation''. In: *Journal of the Association for Computing Machinery (ACM)* 28.1 (1981), pp. 114–133 (cited on p. 100).

[CL07] Arnaud Carayol and Christof Löding. ``MSO on the Infinite Binary Tree: Choice and Order''. In: *"Proceedings of Computer Science Logic, 21st Annual Conference of the EACSL (CSL 2007)"*. Vol. 4646. Lecture Notes in Computer Science. Springer-Verlag, 2007, pp. 161–176 (cited on p. 64).

[CL08] Thomas Colcombet and Christof Löding. ``The Non-deterministic Mostowski Hierarchy and Distance-Parity Automata''. In: *Proceedings of the 35th International Colloquium on Automata, Languages, and Programming (ICALP 2008)*. Vol. 5126. Lecture Notes in Computer Science. Springer-Verlag, 2008, pp. 398–409 (cited on p. 152).

[CN78] Bruno Courcelle and Maurice Nivat. ``The Algebraic Semantics of Recursive Program Schemes''. In: *Proceedings of the 7th Symposium, Mathematical Foundations of Computer Science (MFCS 1978)*. Vol. 64. Lecture Notes in Computer Science. Springer-Verlag, 1978, pp. 16–30 (cited on pp. 20, 90).

[Col09] Thomas Colcombet. ``Private communication''. Rhodos Palace's swimming pool, Greece. 2009 (cited on pp. 103, 133).

[Col13] Thomas Colcombet. ``Fonctions régulières de coût''. Habilitation à diriger des recherches. Université Paris Diderot - Paris 7, 2013 (cited on p. 152).

[Con92] Anne Condon. ``The Complexity of Stochastic Games''. In: *Information and Computation* 96.2 (1992), pp. 203–224 (cited on p. 141).

[Cou78a] Bruno Courcelle. ``A Representation of Trees by Languages I''. In: *Theoretical Computer Science* 6 (1978), pp. 255–279 (cited on pp. 20, 31, 32, 90).

[Cou78b] Bruno Courcelle. ``A Representation of Trees by Languages II''. In: *Theoretical Computer Science* 7 (1978), pp. 25–55 (cited on pp. 20, 31, 32, 90).

[Cou89] Bruno Courcelle. ``The Monadic Second-Order Logic of Graphs, II: Infinite Graphs of Bounded Width''. In: *Mathematical Systems Theory* 21 (1989), pp. 187–221 (cited on p. 91).

[Cou94] Bruno Courcelle. ``Monadic Second-Order Definable Graph Transductions: A Survey.'' In: *Theoretical Computer Science* 126.1 (1994), pp. 53–75 (cited on pp. 55, 62).

[Cou95] Bruno Courcelle. ``The monadic second-order logic of graphs IX: machines and their behaviours''. In: *Theoretical Computer Science* 151 (1995), pp. 125–162 (cited on p. 21).

[CS08] Arnaud Carayol and Michaela Slaats. ``Positional Strategies for Higher-Order Pushdown Parity Games''. In: *Proceedings of the 33rd Symposium, Mathematical Foundations of Computer Science (MFCS 2008)*. Vol. 5162. Lecture Notes in Computer Science. Springer-Verlag, 2008, pp. 217–228 (cited on pp. 68, 71, 72, 89).

[CSV09] Rohit Chadha, A. Prasad Sistla, and Mahesh Viswanathan. ``Power of Randomization in Automata on Infinite Strings''. In: *Proceedings of the 20th International Conference on Concurrency Theory (CONCUR 2009)*. Vol. 5710. Lecture Notes in Computer Science. Springer-Verlag, 2009, pp. 229–243 (cited on p. 101).

[CW03] Arnaud Carayol and Stefan Wöhrle. ``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)*. Vol. 2914. Lecture Notes in Computer Science. Springer-Verlag, 2003, pp. 112–123 (cited on pp. 21, 49).

[CW07] Thierry Cachat and Igor Walukiewicz. ``The Complexity of Games on Higher Order Pushdown Automata''. In: *CoRR* abs/0705.0262 (2007) (cited on p. 78).

[CY90] Costas Courcoubetis and Mihalis Yannakakis. ``Markov Decision Processes and Regular Events (Extended Abstract)''. In: *Proceedings of the 17th International Colloquium on Automata, Languages, and Programming (ICALP 1990)*. Vol. 443. Lecture Notes in Computer Science. Springer-Verlag, 1990, pp. 336–349 (cited on pp. 121, 124).

[Dam77a] Werner Damm. ``Higher type program schemes and their tree languages''. In: *Theoretical Computer Science, 3rd GI-Conference*. Vol. 48. Lecture Notes in Computer Science. Springer-Verlag, 1977, pp. 51–72 (cited on p. 20).

[Dam77b] Werner Damm. ``Languages Defined by Higher Type Program Schemes''. In: *Proceedings of the 4th International Colloquium on Automata, Languages, and Programming (ICALP 1977)*. Vol. 52. Lecture Notes in Computer Science. Springer-Verlag, 1977, pp. 164–179 (cited on p. 20).

[Dam79] Werner Damm. ``An Algebraic Extension of the Chomsky-Hierarchy''. In: *Proceedings of the 8th Symposium, Mathematical Foundations of Computer Science (MFCS 1979)*. Vol. 74. Lecture Notes in Computer Science. Springer-Verlag, 1979, pp. 266–276 (cited on pp. 49, 50).

[Dam82] Werner Damm. ``The IO- and OI-Hierarchies.'' In: *Theoretical Computer Science* 20 (1982), pp. 95–207 (cited on pp. 20, 28, 47, 51, 52).

[DG86] Werner Damm and Andreas Goerdt. ``An automata-theoretical characterization of the OI-Hierarchy''. In: *Information and Computation* 71 (1986), pp. 1–32 (cited on p. 20).

[DR10] Laurent Doyen and Jean-François Raskin. ``Antichain Algorithms for Finite Automata''. In: *Proceedings of 16th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2010)*. Vol. 6015. Lecture Notes in Computer Science. Springer-Verlag, 2010, pp. 2–22 (cited on pp. 100, 137).

[EFT96] Heinz-Dieter Ebbinghaus, Jörg Flum, and Wolfgang Thomas. *Mathematical Logic*. Second edition. Undergraduate Texts in Mathematics. Springer-Verlag, 1996 (cited on pp. 11, 12).

[EH86] E. Allen Emerson and Joseph Y. Halpern. ``"Sometimes'' and "Not Never'' revisited: on branching versus linear time temporal logic''. In: *Journal of the Association for Computing Machinery (ACM)* 33.1 (1986), pp. 151–178 (cited on p. 97).

[EHRS00] Javier Esparza, David Hansel, Peter Rossmanith, and Stefan Schwoon. ``Efficient Algorithms for Model Checking Pushdown Systems''. In: *Proceedings of the 12th International Conference on Computer Aided Verification (CAV 2000)*. Lecture Notes in Computer Science. Springer-Verlag, 2000, pp. 232–247 (cited on p. 84).

[EJ91] E. Allen Emerson and Charanjit S. Jutla. ``Tree automata, mu-calculus and determinacy (extended abstract)''. In: *Proceedings of the 32nd Annual Symposium on Foundations of Computer Science (FoCS 1991)*. IEEE Computer Society, 1991, pp. 368–377 (cited on pp. 71, 72, 98).

[Eng83] Joost Engelfriet. ``Iterated Pushdown Automata and Complexity Classes''. In: *Proceedings of the Fifteenth Annual ACM Symposium on the Theory of Computing (STOC'83)*. ACM, 1983 (cited on p. 20).

[Eng91] Joost Engelfriet. ``Iterated Stack Automata and Complexity Classes''. In: *Information and Computation* 95.1 (1991), pp. 21–75 (cited on pp. 20, 49, 50).

[ES77] Joost Engelfriet and Erik Meineche Schmidt. ``IO and OI. I''. In: *Journal of Computer and System Sciences* 15.3 (1977), pp. 328–353 (cited on p. 20).

[ES78] Joost Engelfriet and Erik Meineche Schmidt. ``IO and OI. II''. In: *Journal of Computer and System Sciences* 16.1 (1978), pp. 67–99 (cited on p. 20).

[Eve57] Hugh Everett. ``Recursive games''. In: *Contributions to the theory of games*. Vol. 3. Annals of Mathematics Studies, no. 39. Princeton University Press, 1957, pp. 47–78 (cited on pp. 104, 129).

[FGW07] Jörg Flum, Erich Grädel, and Thomas Wilke. *Logic and Automata: History and Perspectives*. Amsterdam University Press, 2007 (cited on p. 21).

[FHMV95] Ronald Fagin, Joseph Y. Halpern, Yoram Moses, and Moshe. Y. Vardi. *Reasoning about Knowledge*. MIT Press, 1995 (cited on p. 137).

[Fra06] Séverine Fratani. ``Automates à piles de piles …de piles''. PhD thesis. Université de Bordeaux, 2006 (cited on pp. 68, 89).

[FWW97] Alain Finkel, Bernard Willems, and Pierre Wolper. ``A direct symbolic approach to model checking pushdown systems''. In: *Electronic Notes in Theoretical Computer Science* 9 (1997), pp. 27–37 (cited on p. 84).

[GH82] Yuri Gurevich and Leo Harrington. ``Trees, Automata, and Games''. In: *Proceedings of the Fourteenth Annual ACM Symposium on the Theory of Computing (STOC'82)*. ACM, 1982, pp. 60–65 (cited on pp. 98, 100, 123).

[Gim04] Hugo Gimbert. ``Parity and Exploration Games on Infinite Graphs''. In: *"Proceedings of Computer Science Logic, 18th Annual Conference of the EACSL (CSL 2004)"*. Vol. 3210. Lecture Notes in Computer Science. Springer-Verlag, 2004, pp. 56–70 (cited on p. 92).

[GL73] Stephen Garland and David Luckham. ``Program schemes, recursion schemes and formal languages''. In: *Journal of Computer and System Sciences* 7.2 (1973), pp. 119–160 (cited on p. 19).

[GO10] Hugo Gimbert and Youssouf Oualhadj. ``Probabilistic Automata on Finite Words: Decidable and Undecidable Problems''. In: *Proceedings of the 37th International Colloquium on Automata, Languages, and Programming (ICALP 2010)*. Vol. 6199. Lecture Notes in Computer Science. Springer-Verlag, 2010, pp. 527–538 (cited on pp. 103, 133).

[Grä08] Erich Grädel. ``Banach-Mazur Games on Graphs''. In: *Proceedings of the 28th International Conference on Foundations of Software Technology and Theoretical Computer Science (FST&TCS 2008)*. Vol. 2. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2008, pp. 364–382 (cited on pp. 99, 115).

[GS83] Yuri Gurevich and Saharon Shelah. ``Rabin's Uniformization Problem''. In: *Journal of Symbolic Logic* 48.4 (1983), pp. 1105–1119 (cited on pp. 63, 64).

[GTW02] Erich Grädel, Wolfgang Thomas, and Thomas Wilke, eds. *Automata, Logics, and Infinite Games: A Guide to Current Research [outcome of a Dagstuhl seminar, February 2001]*. Vol. 2500. Lecture Notes in Computer Science. Springer-Verlag, 2002 (cited on pp. 21, 98, 100, 123, 130).

[GZ07] Hugo Gimbert and Wiesław Zielonka. ``Perfect Information Stochastic Priority Games''. In: *Proceedings of the 34th International Colloquium on Automata, Languages, and Programming (ICALP 2007)*. Lecture Notes in Computer Science. Springer-Verlag, 2007, pp. 850–861 (cited on p. 135).

[Had12] Axel Haddad. ``IO vs OI in Higher-Order Recursion Schemes''. In: *Proceedings 8th Workshop on Fixed Points in Computer Science*. Vol. 77. Electronic Proceedings in Theoretical Computer Science. 2012, pp. 23–30 (cited on pp. 28, 61).

[Had13] Axel Haddad. ``Shape-Preserving Transformations of Higher-Order Recursion Schemes''. PhD thesis. Université Paris Diderot - Paris 7, 2013 (cited on pp. 28, 61, 68, 89, 90).

[Hag08] Matthew Hague. ``Global Model Checking of Higher-Order Pushdown Systems''. PhD thesis. University of Oxford, 2008 (cited on pp. 71, 72, 91).

[Hag13] Matthew Hague. ``Saturation of Concurrent Collapsible Pushdown Systems''. In: *Proceedings of the 33rd International Conference on Foundations of Software Technology and Theoretical Computer Science (FST&TCS 2013)*. Vol. 24. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2013, pp. 313–325 (cited on p. 92).

[HMU07] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation*. 3rd ed. Upper Saddle River, NJ: Pearson Addison-Wesley, 2007 (cited on p. 10).

[HO00] J. Martin E. Hyland and C.-H. Luke Ong. ``On Full Abstraction for PCF: I. Models, observables and the full abstraction problem, II. Dialogue games and innocent strategies, III. A fully abstract and universal game model''. In: *Information and Computation* 163 (2000), pp. 285–408 (cited on pp. 21, 54).

[HO08] Matthew Hague and C.-H. Luke Ong. ``Symbolic Backwards-Reachability Analysis for Higher-Order Pushdown Systems''. In: *Logical Methods in Computer Science* 4.4 (2008) (cited on pp. 71, 72, 80).

[HO10] Matthew Hague and C.-H. Luke Ong. ``Analysing mu-calculus properties of pushdown systems''. In: *Proceedings of the 17th International SPIN Workshop (SPIN 2010)*. 2010, pp. 187–192 (cited on pp. 84, 87).

[HO11] Matthew Hague and C.-H. Luke Ong. ``A saturation method for the modal μ-calculus over pushdown systems''. In: *Information and Computation* 209.5 (2011), pp. 799–821 (cited on pp. 71, 72, 91).

[Hor08] Florian Horn. ``Random Games''. PhD thesis. Université Paris Diderot - Paris 7 & RWTH Aachen, 2008 (cited on p. 130).

[IM08] Kazuhiro Inaba and Sebastian Maneth. ``The Complexity of Tree Transducer Output Languages''. In: *Proceedings of the 28th International Conference on Foundations of Software Technology and Theoretical Computer Science (FST&TCS 2008)*. Vol. 2. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2008, pp. 244–255 (cited on p. 20).

[Ind76] Klaus Indermark. ``Schemes with Recursion on Higher Types''. In: *Proceedings of the 5th Symposium, Mathematical Foundations of Computer Science (MFCS 1976)*. Vol. 45. Lecture Notes in Computer Science. Springer-Verlag, 1976, pp. 352–358 (cited on p. 20).

[Jan12] Petr Jancar. ``Decidability of DPDA Language Equivalence via First-Order Grammars''. In: *Proceedings of the 27th Annual IEEE Symposium on Logic in Computer Science (LiCS 2012)*. IEEE Computer Society, 2012, pp. 415–424 (cited on p. 90).

[JW96] David Janin and Igor Walukiewicz. ``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)*. Vol. 1119. Lecture Notes in Computer Science. Springer-Verlag, 1996, pp. 263–277 (cited on p. 60).

[Kar10] Alexander Kartzow. ``Collapsible Pushdown Graphs of Level 2 are Tree-Automatic''. In: *Proceedings of the 27th Symposium on Theoretical Aspects of Computer Science (STACS 2010)*. Vol. 5. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2010, pp. 501–512 (cited on p. 55).

[Kla94] Nils Klarlund. ``Progress measures, immediate determinacy, and a subset construction for tree automata''. In: *Annals of Pure and Applied Logic* 69.2-3 (1994), pp. 243–268 (cited on p. 98).

[KNU01] Teodor Knapik, Damian Niwiński, and Paweł Urzyczyn. ``Deciding monadic theories of hyperalgebraic trees''. In: *Proceedings of the 5th conference on Typed Lambda Calculi and Applications (TLCA 2001)*. Vol. 2044. Lecture Notes in Computer Science. Springer-Verlag, 2001, pp. 253–267 (cited on pp. 21, 47, 49, 50).

[KNU02] Teodor Knapik, Damian Niwiński, and Paweł Urzyczyn. ``Higher-Order Pushdown Trees Are Easy''. In: *Proceedings of the 5th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS 2002)*. Vol. 2303. Lecture Notes in Computer Science. Springer-Verlag, 2002, pp. 205–222 (cited on pp. 21, 47, 49–52, 54, 63).

[KNUW05] Teodor Knapik, Damian Niwiński, Pawel Urzyczyn, and Igor Walukiewicz. ``Unsafe grammars and panic automata''. In: *Proceedings of the 32nd International Colloquium on Automata, Languages, and Programming (ICALP 2005)*. Vol. 3580. Lecture Notes in Computer Science. Springer-Verlag, 2005, pp. 1450–1461 (cited on pp. 21, 22, 50, 54, 71, 72, 78).

[KO09] Naoki Kobayashi and C.-H. Luke Ong. ``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, 2009, pp. 179–188 (cited on pp. 22, 54, 68, 81, 88, 89).

[Kob09a] Naoki Kobayashi. ``Model-checking higher-order functions''. In: *Proceedings of the 11th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming (PPDP 2009)*. ACM, 2009, pp. 25–36 (cited on pp. 85, 87).

[Kob09b] Naoki Kobayashi. ``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, 2009, pp. 416–428 (cited on pp. 22, 85, 88).

[Kob11a] Naoki Kobayashi. ``A Practical Linear Time Algorithm for Trivial Automata Model Checking of Higher-Order Recursion Schemes''. In: *Proceedings of the 14th International Conference on Foundations of Software Science and Computational Structures (FoSSaCS 2011)*. Vol. 6604. Lecture Notes in Computer Science. Springer-Verlag, 2011, pp. 260–274 (cited on pp. 85, 87–89).

[Kob11b] Naoki Kobayashi. ``Higher-Order Model Checking: From Theory to Practice''. In: *Proceedings of the 26th Annual IEEE Symposium on Logic in Computer Science (LiCS 2011)*. IEEE Computer Society, 2011, pp. 219–224 (cited on p. 17).

[Kob12] Naoki Kobayashi. GTRecS2*: A model checker for recursion schemes based on games and types*. A tool available at `http://www-kb.is.s.u-tokyo.ac.jp/~koba/gtrecs2`. 2012 (cited on pp. 85, 87–89).

[KP12] Alexander Kartzow and Pawel Parys. ``Strictness of the Collapsible Pushdown Hierarchy''. In: *Proceedings of the 37th Symposium, Mathematical Foundations of Computer Science (MFCS 2012)*. Vol. 7464. Lecture Notes in Computer Science. Springer-Verlag, 2012, pp. 566–577 (cited on p. 51).

[Kri07] Jean-Louis Krivine. ``A call-by-name lambda-calculus machine''. In: *Higher-Order and Symbolic Computation* 20.3 (2007), pp. 199–207 (cited on pp. 22, 88).

[KS81] P.R. Kumar and T.H. Shiau. ``Existence of value and randomized strategies in zero-sum discrete-time stochastic dynamic games''. In: *SIAM Journal of Control and Optimization* 19.5 (1981), pp. 617–634 (cited on p. 129).

[KSU11] Naoki Kobayashi, Ryosuke Sato, and Hiroshi Unno. ``Predicate abstraction and CEGAR for higher-order model checking''. In: *Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2011)*. ACM, 2011, pp. 222–233 (cited on pp. 16, 85, 87).

[Kuc11] Antonín Kucera. *Turn-Based Stochastic Games*. Ed. by Krzysztof R. Apt and Erich Grädel. Lectures in Game Theory for Computer Scientists. Cambridge University Press, 2011, pp. 146–184 (cited on pp. 101, 141).

[KVW00] Orna Kupferman, Moshe Y. Vardi, and Pierre Wolper. ``An automata-theoretic approach to branching-time model checking''. In: *Journal of the Association for Computing Machinery (ACM)* 47.2 (2000), pp. 312–360 (cited on p. 136).

[Len96] Giacomo Lenzi. ``A Hierarchy Theorem for the μ-Calculus''. In: *Proceedings of the 23rd International Colloquium on Automata, Languages, and Programming (ICALP 1996)*. Vol. 1099. Lecture Notes in Computer Science. Springer-Verlag, 1996, pp. 87–97 (cited on pp. 99, 152).

[LLM10] Arnaud Da Costa Lopes, François Laroussinie, and Nicolas Markey. ``ATL with Strategy Contexts: Expressiveness and Model Checking''. In: *Proceedings of the 30th International Conference on Foundations of Software Technology and Theoretical Computer Science (FST&TCS 2010)*. Vol. 8. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2010, pp. 120–132 (cited on p. 152).

[Löd11] Christof Löding. ``Infinite Games and Automata Theory''. In: *Lectures in Game Theory for Computer Scientists*. Ed. by Krzysztof R. Apt and Erich Grädel. Cambridge University Press, 2011, pp. 38–73 (cited on p. 98).

[Mas74] A. N. Maslov. ``The hierarchy of indexed languages of an arbitrary level''. In: *Soviet mathematics Doklady* 15 (1974), pp. 1170–1174 (cited on p. 20).

[Mas76] A. N. Maslov. ``Multilevel Stack Automata''. In: *Problems of Information Transmission* 12 (1976), pp. 38–43 (cited on pp. 20, 37).

[MH84] Satoru Miyano and Takeshi Hayashi. ``Alternating Finite Automata on omega-Words''. In: *Theoretical Computer Science* 32 (1984), pp. 321–330 (cited on p. 100).

[Mir06] Jolie de Miranda. ``Structures generated by higher-order grammars and the safety constraint''. PhD thesis. University of Oxford, 2006 (cited on pp. 19, 47).

[MMV10] Fabio Mogavero, Aniello Murano, and Moshe Y. Vardi. ``Reasoning About Strategies''. In: *Proceedings of the 30th International Conference on Foundations of Software Technology and Theoretical Computer Science (FST&TCS 2010)*. Vol. 8. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2010, pp. 133–144 (cited on p. 152).

[Mos85] Andrzej W. Mostowski. ``Regular expressions for infinite trees and a standard form of automata''. In: *Computation theory*. Vol. 208. Lecture Notes in Computer Science. Springer-Verlag, 1985, pp. 157–168 (cited on pp. 98, 152).

[Mos91] Andrzej W. Mostowski. *Games with forbidden positions*. Tech. rep. 78. Uniwersytet Gdański, Instytut Mathmatyki, 1991 (cited on p. 98).

[MS85] David. E. Muller and Paul. E. Schupp. ``The Theory of Ends, Pushdown Automata, and Second-Order Logic''. In: *Theoretical Computer Science* 37 (1985), pp. 51–75 (cited on pp. 70, 72, 91).

[MS95] David E. Muller and Paul E. Schupp. ``Simulating Alternating Tree Automata by Nondeterministic Automata''. In: *Theoretical Computer Science* 141.1&2 (1995), pp. 69–107 (cited on pp. 98, 100, 137–139).

[Muc85] Andrej A. Muchnik. ``Games on infinite trees and automata with blind alleys. A new proof of decidability for the monadic theory of two successor functions''. In: *Semiotics and information science*. Vol. 24. Moscow: Akad. Nauk SSSR Vsesoyuz. Inst. Nauchn. i Tekhn. Inform., 1985, pp. 16–40, 142. An English version was later published in the Bulletin of EATCS Vol. 48, pp. 220-267, 1992 (cited on p. 98).

[Niv72] Maurice Nivat. ``On the interpretation of recursive program schemes''. In: *Symposia Matematica*. 1972 (cited on p. 19).

[Niw86] Damian Niwiński. ``On Fixed-Point Clones (Extended Abstract)''. In: *Proceedings of the 13th International Colloquium on Automata, Languages, and Programming (ICALP 1986)*. Vol. 226. Lecture Notes in Computer Science. Springer-Verlag, 1986, pp. 464–473 (cited on pp. 99, 152).

[NRO12] Robin P. Neatherway, Steven James Ramsay, and C.-H. Luke Ong. ``A traversal-based algorithm for higher-order model checking''. In: *Proceedings of the 17th ACM SIGPLAN International Conference on Functional Programming (ICFP 2012)*. 2012, pp. 353–364 (cited on pp. 17, 85, 87–89).

[NW03] Damian Niwinski and Igor Walukiewicz. ``A gap property of deterministic tree languages''. In: *Theoretical Computer Science* 1.303 (2003), pp. 215–231 (cited on p. 144).

[NW05] Damian Niwinski and Igor Walukiewicz. ``Deciding Nondeterministic Hierarchy of Deterministic Tree Automata''. In: *Electronic Notes in Theoretical Computer Science* 123 (2005), pp. 195–208 (cited on p. 152).

[Ong06] C.-H. Luke Ong. ``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, 2006, pp. 81–90 (cited on pp. 21, 54, 57, 88).

[OR11] C.-H. Luke Ong and Steven James Ramsay. ``Verifying higher-order functional programs with pattern-matching algebraic data types''. In: *Proceedings of the 38th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 2011)*. ACM, 2011, pp. 587–598 (cited on pp. 85, 88).

[OR94] Martin J. Osborne and Ariel Rubinstein. *A course in game theory*. The MIT Press, 1994 (cited on p. 102).

[Par11] Paweł Parys. ``Collapse Operation Increases Expressive Power of Deterministic Higher Order Pushdown Automata''. In: *Proceedings of the 28th Symposium on Theoretical Aspects of Computer Science (STACS 2011)*. Vol. 9. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2011, pp. 603–614 (cited on pp. 36, 47, 53).

[Par12] Paweł Parys. ``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, 2012, pp. 521–530 (cited on pp. 36, 47, 53).

[Paz71] A. Paz. *Introduction to probabilistic automata*. Academic Press New York, 1971 (cited on p. 101).

[Pin07] Sophie Pinchinat. ``A Generic Constructive Solution for Concurrent Games with Expressive Constraints on Strategies''. In: *Proceedings of the 5th International Symposium on Automated Technology for Verification and Analysis (ATVA 2007)*. Vol. 4762. Lecture Notes in Computer Science. Springer-Verlag, 2007, pp. 253–267 (cited on p. 152).

[PP04] Dominique Perrin and Jean-Éric Pin. *Infinite Words*. Vol. 141. Pure and Applied Mathematics. Elsevier, 2004 (cited on pp. 10, 105, 106).

[Puc13] Bernd Puchala. ``Synthesis of Winning Strategies for Interaction under Partial Information''. PhD thesis. RWTH Aachen University, 2013 (cited on p. 100).

[Rab63] Michael O. Rabin. ``Probabilistic Automata''. In: *Information and Control* 6.3 (1963), pp. 230–245 (cited on pp. 101, 142).

[Rab69] Michael O. Rabin. ``Decidability of second-order theories and automata on infinite trees''. In: *Transactions of the American Mathematical Society* 141 (1969), pp. 1–35 (cited on pp. 21, 62, 97, 98, 106).

[Rab72] Michael O. Rabin. *Automata on infinite objects and Church's problem*. Conference Board of the Mathematical Sciences Regional Conference Series in Mathematics, No. 13. Providence, R.I.: American Mathematical Society, 1972, pp. iii+22 (cited on p. 97).

[Rei79] John H. Reif. ``Universal Games of Incomplete Information''. In: *Proceedings of the 11th Annual ACM Symposium on the Theory of Computing (STOC'79)*. ACM, 1979, pp. 288–308 (cited on pp. 100, 137).

[Rei99] Robert D. Reisz. ``Decomposition Theorems for Probabilistic Automata over Infinite Objects''. In: *Informatica, Lith. Acad. Sci.* 10.4 (1999), pp. 427–440 (cited on p. 101).

[Ren12] Jérôme Renault. ``The value of Repeated Games with an informed controller''. In: *Mathematics of Operations Research* 37 (2012), pp. 154–179 (cited on p. 104).

[RSJM05] Thomas W. Reps, Stefan Schwoon, Somesh Jha, and David Melski. ``Weighted pushdown systems and their application to interprocedural dataflow analysis''. In: *Science of Computer Programming* 58.1-2 (2005), pp. 206–263 (cited on p. 84).

[Rub79] Ariel Rubinstein. ``Equilibrium in Supergames with the Overtaking Criterion''. In: *Journal of Economic Theory* 21 (1979), pp. 1–9 (cited on p. 104).

[Sch02] Stefan Schwoon. ``Model-checking Pushdown Systems''. PhD thesis. Technical University of Munich, 2002 (cited on pp. 16, 84).

[SES08] Dejvuth Suwimonteerabuth, Javier Esparza, and Stefan Schwoon. ``Symbolic Context-Bounded Analysis of Multithreaded Java Programs''. In: *Proceedings of the 15th International SPIN Workshop (SPIN 2008)*. 2008, pp. 270–287 (cited on p. 84).

[Sha53] Lloyd S. Shapley. ``Stochastic games''. In: *Proceedings of the National Academy of Science USA*. Vol. 39. 1953, pp. 1095–1100 (cited on p. 104).

[SP81] Micha Sharir and Amir Pnueli. ``Two approaches to interprocedural data flow analysis''. In: *Program Flow Analysis: Theory and Applications*. Ed. by Steven S. Muchnick and Neil D. Jones. Prentice-Hall, 1981. Chap. 7, pp. 189–234 (cited on p. 86).

[SSE06] Dejvuth Suwimonteerabuth, Stefan Schwoon, and Javier Esparza. ``Efficient Algorithms for Alternating Pushdown Systems with an Application to the Computation of Certificate Chains''. In: *Proceedings of the 4th International Symposium on Automated Technology for Verification and Analysis (ATVA 2006)*. Vol. 4218. Lecture Notes in Computer Science. Springer-Verlag, 2006, pp. 141–153 (cited on pp. 84, 85, 87).

[Sti00a] Colin Stirling. *Decidability of Bisimulation Equivalence for Pushdown Processes*. Tech. rep. EDI-INF-RR-0005. School of Informatics, University of Edinburgh, 2000 (cited on pp. 20, 23, 90).

[Sti00b] Colin Stirling. ``Schema Revisited''. In: *"Proceedings of Computer Science Logic, 14th Annual Conference of the EACSL (CSL 2000)"*. Vol. 1862. Lecture Notes in Computer Science. Springer-Verlag, 2000, pp. 126–138 (cited on p. 20).

[Sti01] Colin Stirling. ``Decidability of DPDA equivalence''. In: *Theoretical Computer Science* 255.1-2 (2001), pp. 1–31 (cited on pp. 20, 90).

[Sti06] Colin Stirling. ``Second-Order Simple Grammars''. In: *Proceedings of the 17th International Conference on Concurrency Theory (CONCUR 2006)*. Vol. 4137. Lecture Notes in Computer Science. Springer-Verlag, 2006, pp. 509–523 (cited on p. 90).

[Sti09] Colin Stirling. ``Dependency Tree Automata''. In: *Proceedings of the 12th International Conference on Foundations of Software Science and Computational Structures (FoSSaCS 2009)*. Vol. 5504. Lecture Notes in Computer Science. Springer-Verlag, 2009, pp. 92–106 (cited on p. 72).

[SW11] Sylvain Salvati and Igor Walukiewicz. ``Krivine Machines and Higher-Order Schemes''. In: *Proceedings of the 38th International Colloquium on Automata, Languages, and Programming (ICALP 2011)*. Vol. 6756. Lecture Notes in Computer Science. Springer-Verlag, 2011, pp. 162–173 (cited on pp. 22, 54, 88).

[SW12] Sylvain Salvati and Igor Walukiewicz. ``Recursive Schemes, Krivine Machines, and Collapsible Pushdown Automata''. In: *Proceedings of Reachability Problems - 6th International Workshop*. Vol. 7550. Lecture Notes in Computer Science. Springer-Verlag, 2012, pp. 6–20 (cited on pp. 22, 88, 90).

[SW13a] Sylvain Salvati and Igor Walukiewicz. ``Evaluation is MSOL-compatible''. In: *Proceedings of the 33rd International Conference on Foundations of Software Technology and Theoretical Computer Science (FST&TCS 2013)*. Vol. 24. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2013, pp. 103–114 (cited on p. 90).

[SW13b] Sylvain Salvati and Igor Walukiewicz. ``Using Models to Model-Check Recursive Schemes''. In: *Proceedings of the 11th International Conference on Typed Lambda Calculi and Applications (TLCA'13)*. Vol. 7941. Lecture Notes in Computer Science. Springer-Verlag, 2013, pp. 189–204 (cited on p. 61).

[Sén02] Géraud Sénizergues. ``L(A)=L(B)? A simplified decidability proof''. In: *Theoretical Computer Science* 281.1-2 (2002), pp. 555–608 (cited on pp. 20, 90).

[Sén97] Géraud Sénizergues. ``The Equivalence Problem for Deterministic Pushdown Automata is Decidable''. In: *Proceedings of the 24th International Colloquium on Automata, Languages, and Programming (ICALP 1997)*. Vol. 1256. Lecture Notes in Computer Science. Springer-Verlag, 1997, pp. 671–681 (cited on pp. 20, 90).

[Tho97] Wolfgang Thomas. ``Languages, Automata, and Logic''. In: *Handbook of Formal Language Theory*. Ed. by G. Rozenberg and A. Salomaa. Vol. III. Springer-Verlag, 1997, pp. 389–455 (cited on pp. 11, 21, 98).

[UTK10] Hiroshi Unno, Naoshi Tabuchi, and Naoki Kobayashi. ``Verification of Tree-Processing Programs via Higher-Order Model Checking''. In: *Proceedings of 8th Asian Symposium on Programming Languages and Systems (APLAS 2010)*. Vol. 6461. Lecture Notes in Computer Science. Springer-Verlag, 2010 (cited on p. 85).

[Var98] Moshe Y. Vardi. ``Reasoning about The Past with Two-Way Automata''. In: *Proceedings of the 25th International Colloquium on Automata, Languages, and Programming (ICALP 1998)*. Vol. 1443. Lecture Notes in Computer Science. Springer-Verlag, 1998, pp. 628–641 (cited on pp. 71, 72).

[VV12] Hagen Völzer and Daniele Varacca. ``Defining Fairness in Reactive and Concurrent Systems''. In: *Journal of the Association for Computing Machinery (ACM)* 59.3 (2012), p. 13 (cited on pp. 99, 115, 118).

[VW07] Moshe Y. Vardi and Thomas Wilke. ``Automata: from logics to algorithms''. In: *Logic and Automata: History and Perspectives*. Amsterdam University Press, 2007, pp. 629–736 (cited on p. 98).

[Wag77] Klaus W. Wagner. ``Eine topologische Charakterisierung einiger Klassen regulärer Folgenmengen''. In: *Elektronische Informationsverarbeitung und Kybernetik* 13.9 (1977), pp. 473–487 (cited on pp. 99, 152).

[Wal01] Igor Walukiewicz. ``Pushdown processes: games and model-checking''. In: *Information and Computation* 157 (2001), pp. 234–263 (cited on p. 22).

[Wal04] Igor Walukiewicz. ``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, 2004, pp. 356–366 (cited on pp. 80, 98).

[Wal96] Igor Walukiewicz. ``Pushdown Processes: Games and Model Checking''. In: *Proceeding of the 8th International Conference on Computer Aided Verification (CAV 1996)*. Vol. 1102. Lecture Notes in Computer Science. Springer-Verlag, 1996, pp. 62–74 (cited on pp. 70, 72, 80).

[WDHR06] Martin De Wulf, Laurent Doyen, Thomas A. Henzinger, and Jean-François Raskin. ``Antichains: A New Algorithm for Checking Universality of Finite Automata''. In: *Proceedings of the 18th International Conference on Computer Aided Verification (CAV 2006)*. Vol. 4144. Lecture Notes in Computer Science. Springer-Verlag, 2006, pp. 17–30 (cited on pp. 100, 137).

[WDMR08] Martin De Wulf, Laurent Doyen, Nicolas Maquet, and Jean-François Raskin. ``Alaska''. In: *Proceedings of the 6th International Symposium on Automated Technology for Verification and Analysis (ATVA 2008)*. Vol. 5311. Lecture Notes in Computer Science. Springer-Verlag, 2008, pp. 240–245 (cited on pp. 100, 137).

[Wei14] Thomas Weidner. ``Private communication''. 2014 (cited on p. 144).

[Wil01] Thomas Wilke. ``Alternating Tree Automata, Parity Games and Modal μ-Calculus''. In: *Bulletin of the Belgian Mathematical Society* 8.2 (2001), pp. 359–391 (cited on pp. 13, 80, 98, 100).

[WT07] Stefan Wöhrle and Wolfgang Thomas. ``Model Checking Synchronized Products of Infinite Transition Systems''. In: *Logical Methods in Computer Science* 3.4 (2007) (cited on pp. 13, 55).

[Wöh05] Stefan Wöhrle. ``Decision problems over infinite graphs higher-order pushdown systems and synchronized products''. PhD thesis. RWTH Aachen, 2005 (cited on p. 42).

[YY93] Alexander Yakhnis and Vladimir Yakhnis. ``Gurevich-Harrington's games defined by finite automata''. In: *Annals of Pure and Applied Logic* 62.3 (1993), pp. 265–294 (cited on p. 98).

[Zei94] Suzanne Zeitman. ``Unforgettable forgetful determinacy''. In: *Journal of Logic and Computation* 4.3 (1994), pp. 273–283 (cited on p. 98).

[Zie98] Wiesław Zielonka. ``Infinite Games on Finitely Coloured Graphs with Applications to Automata on Infinite Trees''. In: *Theoretical Computer Science* 200.1-2 (1998), pp. 135–183 (cited on pp. 11, 98, 130).

Contents

1 Preliminaries **9**
1 Some Basic Objects . . . 9
2 Words and Graphs . . . 9
3 Finite Automata, Regular Languages and ω-Regular Languages . . . 9
4 Two-Player Perfect Information Turn-Based Games on Graphs . . . 10
5 Relational Structures and Logics . . . 11

2 Higher-Order Recursion Schemes & Collapsible Pushdown Automata **15**
1 Modelling Higher-Order Programs . . . 15
1.1 Higher-Order Recursion Schemes . . . 16
1.2 Collapsible Pushdown Automata . . . 17
1.3 A Collapsible Pushdown Approach to Verification . . . 19
2 Historical Background . . . 19
2.1 Recursive Applicative Program Schemes . . . 19
2.2 Extension of Schemes to Higher-Orders . . . 20
2.3 Higher-Order Recursion Schemes as Generators of Infinite Structures . . . 20
2.4 Decidability of Monadic Second Order Logic . . . 21
3 Definitions . . . 22
3.1 Trees and Terms . . . 22
3.2 Labelled Transition Systems . . . 23
3.3 Simply Typed Terms . . . 24
4 Higher-Order Recursion Schemes . . . 25
4.1 Rewriting System Associated with a Recursion Scheme . . . 26
4.2 Value Tree of a Recursion Scheme . . . 27
4.3 Labelled Recursion Schemes . . . 29
5 Some Examples of Trees Generated by Schemes . . . 31
5.1 The Tree Pref($\{a^n b^n c \mid n \geqslant 1\}$) . . . 31
5.1.1 Generating T_0 with a Labelled Recursion Scheme . . . 31
5.1.2 Generating A Tree with Branch Language $\{a^n b^n c \mid n \geqslant 1\}$ with a Recursion Scheme . . . 31
5.2 The Tree Pref($\{a^n b^n c^n d \mid n \geqslant 1\}$) . . . 32
5.2.1 Generating the Tree with a Labelled Recursion Scheme . . . 32
5.2.2 Generating a Tree with Branch Language $\{a^n b^n c^n d \mid n \geqslant 1\}$ with a Recursion Scheme . . . 33
5.3 The Tree Pref($\{a^n c b^{2^n} d \mid n \geqslant 1\}$) . . . 33
5.3.1 Generating the Tree with a Labelled Recursion Scheme . . . 33

5.3.2 Generating a Tree with Branch Language $\{a^n cb^{2^n} d \mid n \geqslant 1\}$ with a Recursion Scheme . . . 33
5.4 The Trees Corresponding to the Tower of Exponentials of Height k. . . . 34
5.5 The Tree of the Urzyczyn Language . . . 36
6 Collapsible Pushdown Automata . . . 37
6.1 Higher-Order Stacks and their Operations . . . 37
6.2 Stacks With Links and their Operations . . . 38
6.3 Higher-Order Pushdown Automata and Collapsible Pushdown Automata . . . 39
7 Some Examples of Collapsible Pushdown Automata and Associated Trees . . . 42
7.1 The Tree Pref($\{a^n b^n c \mid n \geqslant 1\}$) . . . 42
7.2 The Tree Pref($\{a^n cb^{2^n} d \mid n \geqslant 1\}$) . . . 42
7.3 The Trees Corresponding to the Tower of Exponentials of Height k. . . . 43
7.4 A "True" CPDA Example . . . 44
7.5 The Tree of the Urzyczyn Language . . . 45
8 The Safe and the Unsafe Hierarchies . . . 47
8.1 The Safety Constraint . . . 47
8.2 Hierarchies of Trees, Hierarchies of Labelled Transition Systems . . . 48
8.3 The Safe Hierarchies & the Caucal Hierarchies . . . 49
8.4 The Equi-Expressivity Theorem . . . 50
8.5 Back to Safety: Damm's View of Safety . . . 51
8.6 The Safe *vs* Unsafe Conjecture . . . 52
8.6.1 The Safe *vs* Unsafe Conjecture For Labelled Transition Systems . . . 52
8.6.2 The Safe *vs* Unsafe Conjecture For Trees . . . 53
8.7 Summary . . . 53
9 A Glimpse Into Logic . . . 54
9.1 Model-Checking . . . 54
9.1.1 The Case of Trees . . . 54
9.1.2 The Case of Graphs/LTS . . . 54
9.2 Effective Reflection / Global Model-Checking . . . 57
9.2.1 Definitions . . . 57
9.2.2 μ-Calculus Effective Reflection . . . 59
9.2.3 MSO Effective Reflection . . . 60
9.2.4 First Application of MSO Effective Reflection: Avoiding Divergent Computations . . . 61
9.2.5 Second Application of MSO Effective Reflection: an à la Caucal Result for Unsafe Schemes . . . 62
9.3 Effective Selection Property/ Synthesis . . . 63
9.3.1 A Case Study: Choice Functions . . . 63
9.3.2 Effective Selection Property . . . 66
9.3.3 Selection *vs* Reflection . . . 68
10 CPDA Parity Games: Decidability, Winning Regions & Winning Strategies . . . 69
10.1 CPDA Parity Games . . . 69
10.2 Some History and Known Results . . . 70
10.3 Regular Sets of Stacks with Links . . . 71
10.4 CPDA Strategies . . . 74
10.5 Main Result & Some Words About its Proof . . . 76
11 Back to Logic . . . 80
11.1 Labelled Transition Systems . . . 80

11.2 Trees 81
12 Saturation Techniques and the C-SHORe Tool 82
12.1 C-SHORe Tool and Optimisations 84
12.1.1 Checking Properties of Recursion Schemes in Practice: Related Work . . 85
12.1.2 Initial Forward Analysis 86
12.1.3 Experimental Results 87
13 Open Problems and Perspectives 87
13.1 Comparing the Models and Approaches 88
13.2 Pushing the Decidability of Model-Checking 88
13.3 Regularities 89
13.4 Shape Preserving Transformation 89
13.5 *À la* Caucal Hierarchies for Recursion Schemes and Transfer Theorem 90
13.6 An Equi-Expressivity Result for IO-Schemes 90
13.7 The Equivalence Problem for Higher-Order Recursion Schemes 90
13.8 Graph Grammars for Collapsible/Higher-Order Pushdown Graphs 91
13.9 Saturation Techniques and Tools 91
13.10 Boundedness Conditions 92

3 Tree Automata and Imperfect Information Games **97**
1 Overview and Historical Background 97
1.1 Finite Automata on Infinite Trees, Logic and Games 97
1.2 Three Levers to Define New Classes of Tree Automata 98
1.3 Imperfect Information Games 102
1.3.1 The Randomised Strategies Setting 103
1.3.2 The Pure Strategies Setting 103
1.3.3 Related Work in Classical Game Theory 104
1.4 Structure of this Chapter 104
2 Basic Objects, Tree Automata and Regular Tree Languages 104
2.1 Trees and Branches 104
2.2 Non-Deterministic Tree Automata and Regular Tree Languages 105
2.3 Decision Problems, Acceptance Game and Emptiness Game 106
3 Counting Branches in Trees: Automata with Cardinality Constraints 108
3.1 Automata with Cardinality Constraints 108
3.2 Counting Rejecting Branches 109
3.2.1 The Case of Languages $L^{\mathrm{Rej}}_{\leqslant \mathrm{Count}}(\mathcal{A})$ 109
3.2.2 The Case of Languages $L^{\mathrm{Rej}}_{\mathrm{Fin}}(\mathcal{A})$ 110
3.2.3 Languages $L^{\mathrm{Rej}}_{\leqslant \mathrm{Count}}(\mathcal{A})$ and $L^{\mathrm{Rej}}_{\mathrm{Fin}}(\mathcal{A})$ *vs* Büchi Tree Languages 111
3.3 Counting Accepting Branches 112
3.3.1 The Case of Languages $L^{\mathrm{Acc}}_{\infty}(\mathcal{A})$ 112
3.3.2 The Case of Languages $L^{\mathrm{Acc}}_{Uncount}(\mathcal{A})$ 114
4 Requiring Large Sets of Accepting Branches: Automata with Topological Bigness Constraints 115
4.1 Automata with Topological Bigness Constraints 116
4.2 Acceptance Game & Regularity 116
5 Measuring Branches in Trees: Qualitative Tree Languages 117
5.1 Definitions 117
5.2 Pumping Lemma and Closure Properties 119
5.3 Regular Tree Languages and Qualitative Tree Languages are Incomparable . . . 119

5.4 Markov Decision Processes . . . 120
5.5 The Emptiness Problem . . . 121
5.6 Variants: Value of a Tree, Positive Tree Languages . . . 123
6 Imperfect Information Stochastic Games . . . 125
6.1 Definitions . . . 126
6.1.1 Concurrent Arenas . . . 126
6.1.2 Strategies . . . 127
6.1.3 Probability Space and Outcomes of Strategies . . . 127
6.1.4 Objectives, Value of a Game . . . 128
6.1.5 Examples . . . 129
6.2 Decision Problems in the Randomised Strategies Setting . . . 130
6.3 Decision Problems in the Pure Strategies Setting . . . 131
7 Emptiness Of Alternating Tree Automata Using Games With Imperfect Information . 133
7.1 Definitions . . . 134
7.1.1 Stochastic Turn-Based Games . . . 134
7.1.2 Alternating Parity Tree Automata . . . 135
7.1.3 Half-Perfect-Information Turn-Based Stochastic Parity Games . . . 137
7.2 Revisiting The Emptiness Using Imperfect-Information Game . . . 137
7.2.1 A Half-Perfect-Information Emptiness Game . . . 138
7.2.2 Comparison with the Standard Approach . . . 138
7.3 Checking Emptiness Using an Imperfect-Information Game: The Case of $L^{=1}(\mathcal{A})$ for Büchi Condition . . . 141
7.3.1 A Positionality Result for Chronological Games . . . 141
7.4 The Reduction . . . 141
8 Beyond Non-Deterministic Automata: the Probabilistic Setting . . . 142
8.1 Definitions . . . 142
8.1.1 Probabilistic Tree Automata . . . 142
8.1.2 Measurability of AccRuns($\mathcal{A}$) and QualAccRuns($\mathcal{A}$) . . . 143
8.1.3 Almost-Surely Accepted Trees . . . 144
8.1.4 Examples . . . 145
8.2 An Acceptance Game for Qualitative Probabilistic Tree Automata . . . 145
8.3 Decidability and Undecidability Results . . . 145
8.3.1 Partial Observation Markov Decision Processes . . . 146
8.3.2 Decidability Status of the Emptiness Problem for Probabilistic Tree Automata . . . 146
8.4 Expressiveness: Comparison with Regular Tree Languages & Qualitative Tree Languages . . . 147
8.5 Variants . . . 148
8.6 Summary of the Decidability Results on Probabilistic Tree Automata . . . 150
9 Open Problems and Perspectives . . . 150
9.1 Qualitative Tree Languages *vs* Regular Tree Languages . . . 150
9.2 The Class of Qualitative Tree Languages *vs* Distribution $\mu_{\mathbf{p}}$ . . . 151
9.3 Qualitative Tree Languages & Applications . . . 151
9.4 Mostowski-Like Hierarchies . . . 152
9.5 Toward a Simulation Theorem for Büchi Alternating Qualitative Tree Automata 153
9.6 Probabilistic Tree Automata: Emptiness of Languages of the Form $L^{=1}(\mathcal{A})$ . . . 153
9.7 Decidability Frontier for Imperfect Information Games . . . 154

List of Symbols & Notations **157**

Bibliography **161**
Personal References Used in This Document 161
Personal References Not Used in This Document 163
Other References Used in This Document 163

Contents **181**