

GAMES ON PUSHDOWN GRAPHS AND EXTENSIONS

Von der Fakultät für Mathematik, Informatik und
Naturwissenschaften der Rheinisch-Westfälischen Technischen
Hochschule Aachen zur Erlangung des akademischen Grades eines
Doktors der Naturwissenschaften genehmigte Dissertation

vorgelegt von

Thierry Cachat

diplômé d'études approfondies (DEA) en informatique

aus Montélimar, Drôme, Frankreich

Berichter: Prof. Dr. Wolfgang Thomas
Dr. Hab. Didier Caucal

Tag der mündlichen Prüfung: 18. Dezember 2003

Diese Dissertation ist auf den Internetseiten der Hochschulbibliothek online
verfügbar.

Abstract

Two player games are a standard model of reactive computation, where e.g. one player is the controller and the other is the environment. A game is won by a player if she has a winning strategy, *i.e.*, if she can win every play. Given a finite description of the game, our aim is to compute the winner and a winning strategy. For finite graphs these problems have been solved for a long time, although some complexity questions remain open.

We consider several classes of infinite graphs, from transition graphs of pushdown automata up to graphs of the Caucal hierarchy, and we investigate different winning conditions: reachability, recurrence (Büchi), parity, and the a called Σ_3 -condition.

Two kinds of techniques are developed: a symbolic approach based on finite automata recognizing infinite sets of configurations and a game simulation which reduces a given game into a simpler one and solves it. Different kinds of strategies are also constructed: either positional or based on pushdown stack memories.

Zusammenfassung

Reaktive Systeme werden oft durch Spiele mit zwei Personen modelliert, wo typischerweise ein Spieler der Steuerungsprogramm ist, und der andere die Umgebung. Ein Spieler gewinnt ein Spiel, wenn er eine Gewinnstrategie hat, so dass er alle Partien gewinnt. Gegeben die endliche Spezifikation eines Spiels ist es das Ziel, den Gewinner und eine Gewinnstrategie zu berechnen. Für endliche Spielgraphen sind diese Probleme seit langem gelöst, obwohl einige Komplexitätsfragen offen bleiben.

Wir betrachten verschiedene Klassen von unendlichen Graphen, von Kellerautomaten-Transitionsgraphen bis Graphen der Caucal-Hierarchie, und wir studieren verschiedene Gewinnbedingungen: Erreichbarkeits-, Rekurrenz- (Büchi-), Paritäts- und eine sogenannte „ Σ_3 -Bedingung“.

Zwei Arten von Methoden werden entwickelt: die symbolische Methode benutzt endliche Automaten, um unendliche Mengen von Konfigurationen zu erkennen; die Spielreduktion wandelt ein gegebenes Spiel in einem vereinfachten um, und löst es. Verschiedene Arten von Strategien werden auch konstruiert: entweder positionell oder mit Kellerspeicher.

Résumé

Les systèmes réactifs peuvent être modélisés de façon naturelle par des jeux à deux joueurs, où typiquement un joueur est le contrôleur, et l'autre est l'environnement. Un jeu est gagné par un joueur s'il a une stratégie gagnante, c'est-à-dire s'il peut gagner toutes les parties. Étant donné la description finie d'un jeu, notre but est de calculer le gagnant et une stratégie gagnante. Pour les graphes finis, ces problèmes sont résolus depuis longtemps, même si des questions de complexité restent ouvertes.

On considère différentes classes de graphes infinis, des graphes de transition des automates à pile jusqu'au graphes de la hiérarchie de Caucal, et on étudie différentes conditions de gain : accessibilité, récurrence (Büchi), parité, et une condition de type Σ_3 .

Deux sortes de techniques sont développées : l'approche symbolique, qui utilise des automates finis pour reconnaître des ensembles infinis de configuration, et la jeu-simulation, qui transforme un jeu donné en un autre plus simple pour le résoudre. Différentes sortes de stratégies sont aussi construites : soit positionnelles, soit avec une mémoire à pile.

Danksagung

Wolfgang Thomas hat mir eine Assistentenstelle und ein sehr interessantes Forschungsthema angeboten. Er hat mir immer in kurzer Zeit tiefe Hinweise gegeben. Die drei Jahren in Aachen waren wissenschaftlich und auch persönlich besonders interessant und angenehm.

Didier Caucal a guidé mes premiers pas vers la recherche, et a su m'aider à trouver la voie qui me convenait. Il a rapporté ce document en anglais, il ne l'a pas jeté, et il s'est accommodé de ma présentation en anglais.

Alle Mitarbeiter am Lehrstuhl für Informatik VII in Aachen waren sehr nett und haben mir oft geholfen.

Ohne Madeleine Paupard, unsere damalige Deutschlehrerin, hätte ich nicht so gerne Deutsch gelernt, und Deutschland kennengelernt.

Patricia Bouyer und Tanguy Urvoy haben Teile dieser Arbeit probegelesen.

Danke auch an alle anderen, die mir in dieser Zeit geholfen haben.

Contents

1	Introduction	1
2	Framework	9
2.1	Definitions of Games, Winning Conditions, Strategies	9
2.1.1	Basic Notations, Automata	9
2.1.2	Game Graph	10
2.1.3	Winning Conditions	10
2.1.4	Strategies	11
2.2	Motivations and Algorithmic Problems	12
2.2.1	Algorithmic Issues	12
2.2.2	Application to the Verification of Reactive Systems	13
2.2.3	Pushdown Game System	13
2.3	Logics and Decidability	14
2.3.1	The μ -calculus	14
2.3.2	Monadic Second Order Logic	15
2.3.3	Syntax of MSO	15
2.3.4	MSO Definability of Winning Strategies	17
2.3.5	Discussion and Extensions	19
2.4	Limitations to Solutions of Games	19
2.4.1	Intersection of Context-Free Languages	20
2.4.2	Other Graphs	20
2.4.3	A Recursive Game Graph	21
3	Symbolic Presentation of Winning Strategies	23
3.0.4	Technical Preliminaries, \mathcal{P} -Automata	23
3.1	Reachability Game: Computing the Attractor	25
3.1.1	Reachability	25
3.1.2	Determining Membership in the Attractor	30

3.2	Winning Strategy for Player 0	31
3.2.1	Preparation	31
3.2.2	Positional Min-rank Strategy	32
3.2.3	Proof of Lemma 3.2.5	35
3.2.4	Pushdown Strategy	40
3.2.5	Discussion	42
3.3	Back to the Basic Example of Chapter 1	42
3.4	Büchi Condition	44
3.4.1	Computation of Büchi_0^j for all $j > 0$	47
3.4.2	Computation of Büchi_0 Using an Acceleration	50
3.4.3	Simple Example	55
3.4.4	Example of convergence to the fixed-point	57
3.5	Safety Game and Co-Büchi Game	59
3.6	A Σ_3 Winning Condition	62
3.6.1	Motivation	62
3.6.2	Outline of the Solution	64
3.6.3	Details	66
3.7	Discussion, Examples	74
3.7.1	First Example	75
3.7.2	Second Example	78
3.8	Extensions	82
3.8.1	Modifying the Winning Condition	82
3.8.2	Prefix-recognizable Graphs	83
4	Game-Reduction and Parity Games over PDGS	87
4.1	Definition of the Game Simulation	88
4.2	Pushdown Games and Walukiewicz's Results	88
4.3	Example	95
4.4	Extension to a Uniform Solution	97
4.5	Parity Games on Prefix-Recognizable Graphs	98
4.5.1	Reduction to Parity Game on Pushdown Graph	99
5	Parity Games over Caucal Graphs	103
5.1	The Models	104
5.1.1	Higher Order Pushdown Game System	104
5.1.2	Caucal Hierarchy	105
5.1.3	Graph Automaton	107

5.1.4	The Modal μ -calculus	109
5.2	Game-Simulation Between HPDGS and Caucal Graphs	111
5.2.1	From HPDGS to Caucal Graphs	112
5.2.2	From Caucal Graphs to HPDGS	116
5.3	Reducing the Hierarchy Level	117
5.3.1	Proof of Lemma 5.3.2	121
5.4	Complexity and Strategies	126
6	Conclusion	129
6.1	Comparison of Both Approaches	129
6.2	Outlook	131
	Bibliography	133
	List of Figures	143
	Index	145

Chapter 1

Introduction

One of the main challenges in theoretical computer science today is the verification of software and hardware systems. Different methods have been developed to gain confidence in the correctness of a system: type-checking, testing, model-checking, proof systems, and many more. In this thesis we consider the paradigm of reactive computation, where two components, a controller and an environment, interact. This can be the controller of a machine, a plant, an airplane on the one hand, and the machine, plant, airplane itself together with its physical environment on the other hand. These can also be a program (as controller) and a user (as environment). Both controller and environment are allowed to perform certain actions which determine the evolution of the system. The goal of the controller is that for every possible choice of actions of the environment the resulting interaction satisfies certain requirements such as “no accident occurs”, “something is produced”, . . . From a formal point of view it is helpful and natural to see such systems as games with two players: one is the controller and the other is the environment. The controller wins if and only if the given requirement is satisfied.

In this thesis we start from a situation where the problem is already modeled as a two player game. We restrict ourselves to games that are

- turn based: the players play in alternation,
- with perfect information: both players know exactly the current position of the game and also the past of the game,
- without randomness.

(Chess fulfills these requirements, lots of card games use random and imperfect information, paper scissors stone is not turn based, . . .) We are interested in infinite

games, where the interactions can have, at least potentially, an infinite number of steps. More precisely a play is an ω -sequence of steps. This is necessary for modeling processes that can interact for an unbounded time, such as the above mentioned ones.

Our aim is to determine if the controller can win a given game whatever the environment is doing and to implement — if possible — a winning strategy for the controller. In the case of finite game graphs, these problems have been studied for a long time. The aim of this thesis is to consider some classes of infinite graphs. Such graphs are needed to model programs with unbounded variables or recursive procedure calls. We give first a simple example to motivate the definitions and the framework developed later, then discuss the historical background, and finally outline the main contributions of the thesis.

Basic Example

We present a folk game which is a variant of the game of Nim between two players, say Alain (A) and Brigitte (B). At the beginning there are 42 tokens on the table. Alain starts by removing 1 to 6 tokens. Then Brigitte removes also 1 to 6 tokens. And so on, in turn, until the last token is removed. The winner is the one who takes the last token. Clearly the number of possible plays, *i.e.*, sequences of actions from the starting point to the end, is quite large, and it seems not practical to explore it in a brute force manner.

To determine which player can win and what are the “good” moves, the solution is to observe that the first time B is playing, she can remove just enough tokens to let 35 tokens on the table. The second time she can reach the position where there are 28 tokens left, then 21, 14, 7 and finally win with 0. Indeed each time A is removing k tokens, $k \in \{1, \dots, 6\}$, B should answer by taking $(7 - k)$ tokens. Using this idea of complementation modulo 7 it is easy to see that the first player will always lose when starting from a multiple of 7. But it seems more difficult to find a trick if we change the winning condition and consider the following:

The player who takes the last token wins if and only if
he removes in total an even number of tokens.

The formal methods developed in this thesis permit to solve this problem, and more general ones, in an elegant way. Going back to the simplest version it is easy to represent the game by a graph where each position, or vertex, represents a “state” of the game: a number of token left and a letter A or B to know who should play next. The edges, or transitions, of the graph represent the moves allowed to the players.

See Figure 1.1, where only a part of the game graph that is relevant for us is drawn: for player B we consider only the “good” transitions, that realize a winning strategy.

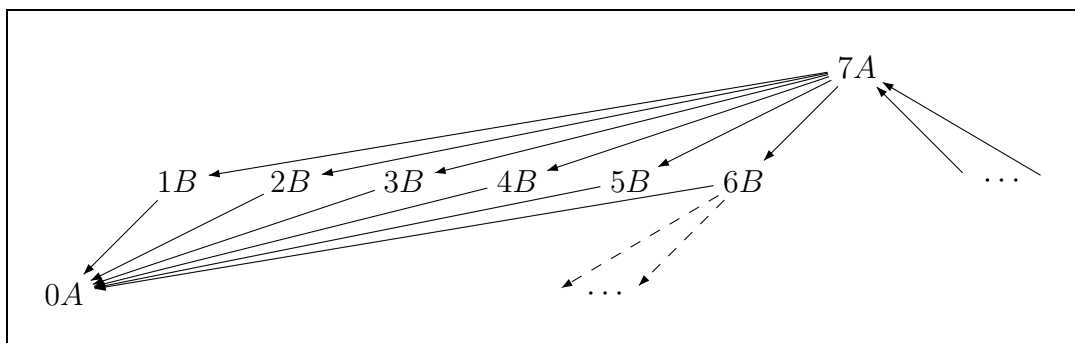


Figure 1.1: The final part of the game graph

Starting with 42 tokens, each play in this game is finite, more precisely the number of moves is bounded by 42. But we can also consider the game starting with 49 tokens, or any positive integer. So even in this basic example it is reasonable to define an infinite (game) graph where each vertex can be considered as starting position. Each vertex of this graph is composed of a natural number and a letter A or B : the vertex set is $\mathbb{N} \times \{A, B\}$. Here it is mathematically simple, because we just consider integers, but a general framework would still help. We have seen that starting with a multiple of 7, the second player has a winning strategy: always reach the next (inferior) multiple of 7. More formally B should move from $(7q + d, B)$ to $(7q, A)$ for all $q \geq 0$ and $d \in [1, 6]$. Here “multiple of 7” is a finite, formal description of an infinite set, and the strategy is also an infinite set of moves, for which we need a more general representation and automatized methods. We return to this example in the technical exposition in Section 3.3.

So far all tokens are identical and not ordered, it is also possible to organize them in a stack, where it is only possible to access to the top of the stack and to remove one token after the other. The subject of the thesis are games where the positions are given (partly) as contents of such pushdown stacks.

Historical Background

Games have been considered long ago in descriptive set theory (see e.g. [CDT02, Kec94] and references thereby), where they can be used to compare the complexity

of two sets. In this framework the mere existence of winning strategies is the central question and there is no need for algorithmic results and decidability procedures.

As we will see later, the example above can be formalized as a reachability game: the aim of one player is to *reach* the position where there is no more token on the table. The reachability is the simplest winning condition that we can consider. When dealing with a two-player game, one speaks of game reachability and one wants to compute the set of positions from which the first player can win whatever the second player do. If there is only one player, one speaks of simple reachability: the question is whether there is a way to reach the goal set (by choosing each transition along the path). In other words one wants to compute the set of predecessors of the goal set.

Some reachability problems for pushdown automata have been considered long ago. In 1964 Büchi solved the simple reachability problem in the framework of canonical systems [Büc64]: rewriting systems on words (seen as configurations) with a finite number of rules. He proved that the set of predecessors (resp. successors) of a regular set is regular. Then in 1970 together with Hosken he considered rewriting systems where some special rules can have many premises [BH70]. They proved again that the set of successors of a regular set is regular. Although they did not consider the framework of games, it is easy to translate a problem of game reachability to a rewriting system in their sense. Their proof uses Monadic Second Order Logic and is not easy to follow. The complexity is four time exponential. The (first) result of [Büc64] for the simple reachability was later improved to a polynomial time solution by Caucal [Cau90] and extended to a more general framework by Coquidé, Dauchet, Gilleron and Vágvolgyi [CDGV94]: they consider rewriting on finite trees, see also Löding [Löd03].

A major advance in the understanding of infinite graphs was achieved in 1969 by Rabin [Rab69], proving the decidability of properties expressed in monadic second-order logic (MSO) over the complete infinite binary tree. The complexity of the procedure is non elementary in the length of the MSO-formula. The binary tree seems rather abstract, but it was used later in 1996 to obtain decidability results on other classes of infinite graphs. Before that in 1985 Muller and Schupp proved the MSO-decidability over pushdown graphs. It is the first result in the model-checking of pushdown graphs, even if the non-elementary complexity is not practicable. It is usual to express a problem of game reachability in MSO and one can also express other winning conditions such as parity, see below.

Then in 1996 Caucal [Cau96] applied the result of Rabin to obtain a new class of infinite graphs with a decidable MSO theory: the prefix-recognizable graphs. This

class extends properly the pushdown graphs of Muller and Schupp, and the equational graphs of Courcelle [Cou90]. The set of vertices of a given prefix-recognizable graph is a regular subset of the complete binary tree (where each vertex is a word over a two letter alphabet), it can be expressed in MSO. And the existence of an edge between two vertices is also expressed by an MSO formula. Then the model-checking of a formula on a prefix-recognizable graph is obtained by translating the formula to an equivalent one on the binary tree, and using the result of Rabin.

The quest for new classes of infinite graphs with a decidable monadic theory was followed later (see Caucal [Cau02], Knapik, Niwinski and Urzyczyn [KNU02], and references thereby). Another approach is to consider the model checking of the μ -calculus (or fragments of it), a logic that is less expressive than MSO, and allow better complexity bounds.

In 1997 Bouajjani, Esparza and Maler [BEM97] have given a direct solution of the alternating reachability (or game reachability) problem for pushdown graphs using finite automata that recognize sets of vertices. They apply this procedure to the model-checking of the logic CTL, which is less expressive than the μ -calculus. In the case of simple reachability (considering only one player) their construction is very close to the one of [Cau90].

Looking at more complex winning conditions than reachability, one should first recall some results about finite graphs. The Muller winning condition is specified by a family \mathcal{F} of vertex sets and requires that the vertices visited infinitely often in the considered play form a set in \mathcal{F} . The core result on finite-state games is the Büchi-Landweber Theorem ([BL69]). It says that for a game on a finite graph with Muller winning condition one can compute the “winning region” of player 0 (*i.e.*, the set of vertices from which player 0 has a winning strategy) and that the corresponding winning strategies are executable by finite automata. In the case of parity games a priority (natural number) is assigned to each vertex and Player 0 wins if and only if the smallest priority seen infinitely often is even. Parity games are interesting because the model-checking problem of the μ -calculus is polynomially equivalent to the problem of solving parity games, see Emerson, Jutla and Sistla [EJS93]: given a graph and a μ -calculus formula, one can construct a parity game such that the first player wins if and only if the formula is satisfied in the graph. It has been showed that for these games (either on finite or infinite graphs) that positional strategies suffice, where the choice of a player depends only on the current position and not on the past of the game, see Emerson and Jutla [EJ91], Thomas [Tho97]. This result is easily effective for finite graphs, but the case of infinite graphs is more difficult.

A major advance was achieved by Walukiewicz in [Wal96b], where parity games

on pushdown graphs are solved by an EXPTIME procedure and it is shown that strategies can be realized also by pushdown automata. For that he used a reduction to a *finite* graph and a refined winning condition involving claims for one player. This result have been extended by Kupferman and Vardi [Var98, KV00] to an exponential time model-checking procedure for prefix-recognizable graphs and the modal μ -calculus, where the strategies can be computed by finite automata with output. In this framework of game also weaker logics and winning conditions have been studied, see among others [CBMS01, EHRS00a, KPV02].

Contribution of the Thesis

In this thesis we develop two kinds of techniques: a symbolic approach based on finite automata recognizing infinite sets of configurations and a game simulation to reduce a given game to a simpler one and solve it. The symbolic approach is based originally on [BEM97, EHRS00a]. To cope with transition graphs of pushdown automata, which are in general infinite, the authors use finite automata that represent infinite sets of configurations. Here a configuration is a word pw where p is a control state and w is a stack content (a word over the stack alphabet). [BEM97] and [EHRS00a] provide a reachability analysis and a model-checking algorithm over pushdown graphs. They consider also alternating reachability which is equivalent to game reachability. We reuse and extend their algorithm to compute also winning strategies in two forms: positional strategies which require the analysis of the current configuration and are therefore computed in linear time in the length of the configuration and pushdown strategies where each step is computed in constant time. This lifts the model-checking algorithms to the level of program synthesis.

Then we consider a pushdown game with a Büchi winning condition: Player 0 has to reach infinitely often the “goal” set to win. This condition was not considered in [BEM97, EHRS00a] in the framework of games (or alternation). Again we can compute strategies. At last we study a new winning condition involving a Σ_3 -quantifier alternation: Player 0 wins if and only if there is some configuration that is visited infinitely often. Because the graph is infinite, this quantification on “some configuration” adds new expressive power. Our solution is an extension of the algorithm for Büchi games and can be considered as a new decidability result. Until now it is open whether one can deduce the winner of this Σ_3 game from the MSO-decidability.

The game simulation exists for a long time in mathematics, where it is not always used in an effective setting. The idea is to reduce a given game to a simpler one

that we can solve and to deduce from it the winner and a winning strategy in the first game. This method has been proposed in [Wal96b] where parity games on pushdown graphs are solved by reducing them to (exponentially larger) games on finite graphs. To cope with the infinity of the possible stack contents the idea is to force one player to claim what will happen later if the stack becomes smaller and to summarize this information. We reuse this reduction and present it with a new proof, hopefully more direct and intuitive. Moreover we explicitly construct from the finite game graph a pushdown strategy in the original game. In [Wal96b] the analysis was done only for a particular initial position (with empty stack). We extend this result to consider any initial position, thus providing a uniform solution, and prov that the winning region of a player forms a regular set of configurations. The winning region of a player is the set of configurations from which he can win. Another game simulation from prefix-recognizable graphs to pushdown graphs with a parity winning condition is obtained directly from the definition of the graph, yielding also to a uniform solution and a computation of pushdown strategies. Here a move in the prefix-recognizable game is simulated by a sequence of moves in the pushdown game.

The results of [Var98, KV00] can also be viewed as a game simulation. Technically the essential point is the reduction from two-way to one-way tree automata. Namely determining the existence of an accepting run is equivalent to determining the winner in an appropriate game where one player wants to prove that there exists an accepting run and the other wants to refute that. We extend this reduction from [Var98] to the case of trees of unbounded or infinite degree. This allows to solve parity games on the graphs of the infinite Caucal hierarchy [Cau02]. The interest of this result is that it allows to solve also parity games on higher order pushdown graphs [KNU02]. Namely given such a game one can reduce it to a parity game on a Caucal graph of the same level. This allows to have a new decision procedure for MSO on these graphs, with a better complexity bound.

Overview

In Chapter 2 we present the basic definitions and formalize the problems we are interested in, recalling some general results of decidability and pointing out the limits due to undecidability cases. The symbolic approach (related to [Cac02a, CDT02]) is developed in Chapter 3 for different winning conditions: first reachability, then Büchi and Σ_3 . Winning strategies are also constructed. Chapters 4 and 5 use game-reductions. Chapter 4 (related to [Cac02c]) is concerned with parity games

over pushdown graphs and prefix-recognizable graphs, and in Chapter 5 (related to [Cac03]) parity games over higher-order pushdown graphs and over Caucal graphs are shown to be interreducible and are solved. Note that Chapters 4 and 5 are independent from Chapter 3. The last chapter compares both approaches and gives some outlook.

Chapter 2

Framework

2.1 Definitions of Games, Winning Conditions, Strategies

2.1.1 Basic Notations, Automata

For a finite set C , we denote by $\mathcal{P}(C)$ the powerset of C (the set of all subsets of C). The set of non-negative integers is \mathbb{N} and ω denotes the first infinite ordinal. We write e.g. $\forall i < 3$ instead of $\forall i \in \mathbb{N}, i < 3$, because we only deal with non-negative integer numbers. The symbol \uplus denotes a disjoint union: $V = V_0 \uplus V_1$ means that $V = V_0 \cup V_1$ and $V_0 \cap V_1 = \emptyset$. We note $[n] = \{0, \dots, n-1\}$ for an integer $n > 0$.

We assume that the reader is familiar with the basic notions of language and automata theory, see e.g. [HU79] for an introduction. We write regular expressions in the usual way, for example $(a + b)^*c$ for letters a, b, c from a (finite) alphabet Σ . The empty word is ε and $\Sigma^{\leq 3} := \varepsilon + \Sigma + \Sigma^2 + \Sigma^3 = \bigcup_{i \leq 3} \Sigma^i$. A finite automaton \mathcal{A} is a tuple $(Q, \Sigma, \Delta, Q_0, F)$, where

- Q is a finite set of states,
- Σ is a finite alphabet,
- $\Delta \subseteq Q \times \Sigma \times Q$ is a set of transitions,
- $Q_0 \subseteq Q$ is a set of initial states and
- $F \subseteq Q$ is a set of final states.

The graph of an automaton is deterministic if additionally $Q_0 = \{q_0\}$ is a singleton and Δ is a transition function, denoted $\delta : Q \times \Sigma \rightarrow Q$.

Given a finite set P of atomic propositions, $\mathcal{B}^+(P)$ is the set of positive Boolean formulas built from the atoms in P , *i.e.*, built using the connectives \wedge and \vee *without* negation. They can be reduced to conjunctive (or disjunctive) normal form.

For a general introduction to graphs and games we refer to [GTW02].

2.1.2 Game Graph

A *game graph* (also called arena in [GTW02]) is a tuple (V_0, V_1, E) , where $V = V_0 \uplus V_1$ is a set of vertices partitioned into vertices of Player 0 and vertices of Player 1 ($V_0 \cap V_1 = \emptyset$) and $E \subseteq V \times V$ is a set of directed unlabeled edges. We write E in infix notation like $\pi_0 E \pi_1$. Starting in a given initial vertex $\pi_0 \in V$, a *play* in (V_0, V_1, E) proceeds as follows:

- if $\pi_0 \in V_0$, Player 0 picks the first transition (move) to π_1 with $\pi_0 E \pi_1$,
- else Player 1 does,

and so on from the new vertex π_1 . A play is a (possibly infinite) maximal sequence $\pi_0 \pi_1 \dots$ of successive vertices. If the play is finite because of a deadlock, then the player who should play next loses immediately.

It is assumed by some authors that $E \subseteq (V_0 \times V_1) \cup (V_1 \times V_0)$, but this is not essential [Wal96b]. At first sight it is simpler to consider finite game graphs, because they are effectively represented by their lists of vertices and edges.

A *game structure* or simply *game* is composed of a game graph and a winning condition.

2.1.3 Winning Conditions

We have seen how a play is generated, but we still have to determine the winner of a play. Along this thesis we will consider different *winning conditions*, from most particular to most general.

- First of all a *reachability game* is defined by a game graph (V_0, V_1, E) and a goal set $R \subseteq V$, where $V = V_0 \uplus V_1$. A play is won by Player 0 if it reaches a configuration of R :

$$\text{Player 0 wins } (\pi_i)_{i \geq 0} \text{ if } \exists i \geq 0 : \pi_i \in R .$$

In this case it is not needed to continue the play.

- In a *Büchi game* we have also a goal set $R \subseteq V$, and Player 0 has to visit R infinitely often:

Player 0 wins $(\pi_i)_{i \geq 0}$ if $\forall j, \exists i > j : \pi_i \in R$.

The Büchi condition is more general than the reachability condition in the sense that it is easy to transform a reachability game into an “equivalent” Büchi game. More precisely given a reachability game one can construct a Büchi game such that each play in one game corresponds to a play in the other game and the winner is the same. This notion will be formalized later in the context of game simulation. Conversely a Büchi game is a particular case of parity game with two colors.

- A *parity game (structure)* (V_0, V_1, E, Ω) is a game graph (V_0, V_1, E) extended by a priority function $\Omega : V \rightarrow [max]$ assigning to each vertex an integer between 0 and $max - 1$, where $max > 0$. For the winning condition we consider the min-parity version: Player 0 wins the play $\pi_0\pi_1 \dots$ if $\liminf_{k \rightarrow \infty} \Omega(\pi_k)$ is even, *i.e.*, if the minimal priority seen infinitely often in the play is even.
- In Section 3.6.3 we will consider a new winning condition —involving a Σ_3 -Formula—, where Player 0 has to visit some vertex infinitely often:

Player 0 wins $(\pi_i)_{i \geq 0}$ if and only if $\exists v \in V, \forall j > 0, \exists i \geq j : \pi_i = v$.

If the game graph is finite ($|V| < \infty$), this last condition is equivalent to a (trivial) Büchi condition where $R = V$, but we will see that in the case of infinite graphs it leads to new ideas and results.

Other winning conditions like those due to Muller, Rabin or Streett are not considered in this thesis. They are related to the set of vertices appearing infinitely often/only finitely often. See [GTW02, Ch. 1] for an introduction. All these winning conditions can also be considered as acceptance conditions when the graph is used as an automaton to read finite or infinite words (where the distinction between V_0 and V_1 is dropped).

2.1.4 Strategies

Informally speaking, given a game structure (a game graph and a winning condition), the aim is to determine if a player is guaranteed to win (whatever the other player is doing), and how. To formalize this we need the notion of strategy.

A *strategy* for Player 0 is a function $f : V^*V_0 \rightarrow V$ associating to each prefix $\pi_0\pi_1 \cdots \pi_n$ of a play such that $\pi_n \in V_0$ a “next move” π_{n+1} with $\pi_n E \pi_{n+1}$. A strategy is *positional* (or memoryless) if it depends only on the current vertex π_n , in this case one can write $f : V_0 \rightarrow V$. According to the general definition, a strategy is not always a finite object, but we will be interested in computable strategies, that admit a finite representation as program.

A strategy f for Player 0 is winning from a vertex π_0 if every play according to this strategy is winning.

For all plays $(\pi_i)_{i \geq 0}$ such that $\forall i \geq 0 : \pi_i \in V_0 \Rightarrow \pi_{i+1} = f(\pi_0\pi_1 \cdots \pi_i)$,
 $(\pi_i)_{i \geq 0}$ is won by Player 0 .

We say that Player 0 *wins the game from* the initial vertex π_0 if he has a *winning strategy* for this game: a strategy such that he wins every play from π_0 . The *winning region* of Player 0 in a given game is the set of vertices from which Player 0 wins the game. We know from [Mar75] that the games we will consider are *determined*: from every initial vertex one of the players has a winning strategy. But this result is not effective, and we are looking for algorithmic solutions.

2.2 Motivations and Algorithmic Problems

2.2.1 Algorithmic Issues

Now we can formalize the problems we are interested in. Given a finite representation of the game, the aim is to determine:

1. the winner from a given initial position π_0 , or
2. the winning region for Player 0 (and if possible an effective representation),
3. a winning strategy from a given node of the winning region, or
4. a procedure to obtain a winning strategy from each node of the winning region.

Note the difference between point 1 and 3: we can answer point 1 if we know that *there exists* a winning strategy from the vertex π_0 , say for Player 0, but in point 3 we want to actually compute such a winning strategy. For a finite graph point 2 is not much different from point 1: if one can iteratively determine the winner from each initial position, one gets an answer to point 2. We will see that for infinite graphs differences arise. Similarly between point 3 and 4. We will call *solution* of a game an answer to point 1 and 3 and *uniform solution* an answer to point 2 and 4.

2.2.2 Application to the Verification of Reactive Systems

A natural application of games is the modeling of reactive systems. In general such a system is composed of two agents: a *controller* and an *environment*. Typical examples are the controller of a power plant and the (physical) power plant itself. Or the controller of an airplane and the airplane itself with the air (resp. the surface). It can also be an operating system as controller, and the user(s) and software(s) as environment. The environment can change within certain limits defined by the modelisation and the controller has to react to the information that it gets from the environment. The controller fulfills its job if he satisfies some requirements, defined also by the modelisation, like: no accident occurs, energy is produced, . . .

This can be nicely represented by a two player game where one player is the controller and the other player is the environment. The game graph defines the limits of the behavior of both players, and the winning condition of the controller defines its goal. Determining the winner of the game answers the question whether *there exists* a controller, whereas computing a winning strategy realizes the controller synthesis. Another standard application will be discussed in Section 2.3.

2.2.3 Pushdown Game System

We will be interested in infinite graphs, but only if they have a finite (and effective) representation. A simple and natural class of infinite graphs is the class of pushdown graphs, already studied in 1985 by Muller and Schupp [MS85] for their structural properties. A major part of the work presented here concerns these graphs.

A *Pushdown Game System (PDGS)* \mathcal{P} is a tuple $(P_0, P_1, \Gamma, \Delta)$, where

- Γ is the finite stack alphabet,
- $P = P_0 \uplus P_1$ the partitioned finite set of control locations, where P_i indicates the game positions of Player i , and
- $\Delta \subseteq P \times \Gamma \times P \times \Gamma^*$ the finite set of (unlabeled) transition rules.

The name “Pushdown Game System” is derived from Pushdown System (PDS): In the sense of [BEM97] (P, Γ, Δ) is a PDS. A PDS is just like a pushdown automaton without input alphabet, without labels on the transitions, and without particular initial configuration (we can consider several), because we do not use them as accepting devices as in the classical automata theory [HU69].

A Pushdown Game System $(P_0, P_1, \Gamma, \Delta)$ defines a game graph (V_0, V_1, E) in the sense of the previous definition, where

$$\begin{aligned} V_0 &= P_0\Gamma^* , \\ V_1 &= P_1\Gamma^* , \\ E &= \{(p\gamma v, qwv) : (p, \gamma, q, w) \in \Delta \text{ and } v \in \Gamma^*\} . \end{aligned}$$

This means that each macro-state or *configuration* is a pair pu of a control location p and a stack content $u \in \Gamma^*$. The set of nodes of the *pushdown game graph* (V, E) is the set of *all* configurations: $V = P\Gamma^*$. For a given configuration the player is determined by the control state only.

If one needs a bottom stack symbol (\perp) one has to declare it explicitly in Γ and Δ , such that it can neither be put nor erased from the stack.

We can consider the same winning conditions as for finite graphs, as defined above: reachability, Büchi, parity. The difference is that we need effective representations of the goal set of states R for Büchi and reachability, and of the function Ω for parity. To give an idea the simplest way is to make the priority of a configuration depend only on the control state, respectively the membership in R .

2.3 Logics and Decidability

There are many connections between games and logics, we refer to [GTW02] for an overview. We will see here both directions: an application of games to logic, and an application of logic to games.

2.3.1 The μ -calculus

The μ -calculus is a modal logic that subsumes many others like CTL or LTL. Given a (labeled) Graph \mathcal{G} and a μ -calculus formula φ , one can transform the formula into an alternating parity automaton in a straightforward way, see [GTW02, ch. 10]. Then the product of the graph and the parity automaton defines a parity game such that the formula φ is satisfied in a given vertex π_0 of \mathcal{G} if and only if Player 0 wins the game from π_0 . Here the alternation between the two players is used to translate the existential and universal connectives of the formula, but the graph \mathcal{G} is not a game graph. This technique of *model-checking game* will be formalized in Section 5.1.3 with the help of alternating graph automata and the details about μ -calculus together with the translation to automata are exposed in Section 5.1.4.

On the other hand, given a parity game structure, there is a μ -calculus formula that express that a vertex is winning for Player 0 (see [Wal96b]). In other words the problem of determining the winner of a parity game is equivalent to the μ -calculus *model-checking* problem. In this framework a uniform solution (point 2. of 2.2.1) amount to a solution to the global model-checking problem: determining the set of nodes at which a given formula is true.

2.3.2 Monadic Second Order Logic

It is well known that one can express in *monadic second-order logic* (MSO) the property that a vertex is in the winning region of Player 0 in a parity game. More precisely MSO is more expressive than the μ -calculus, and the μ -calculus is equivalent to parity games, see [GTW02, ch. 14]. In this standard translation from μ -calculus to MSO the quantifier alternation depth (of \exists and \forall) is equal to the alternation depth of the fixed point operators (μ and ν) of the μ -calculus formula, which is in turn equal to the number of colors. This is relevant for the complexity of the model checking problem, because most of the decision procedures for MSO have a non-elementary time complexity in the number of alternations, see [GTW02, ch. 13] and [MS85].

Nevertheless we will present here, under the restrictions that the degree of the graph is bounded, an MSO-formula that defines the winning region of a parity game where the alternation depth is constant, independent of the number of colors. Moreover this formula permits to compute winning strategies for each player. We recall first the syntax and semantics of MSO. For a more general introduction to MSO we refer to [GTW02, ch. 12] and references thereby. We assume that the reader is familiar with formal logic, otherwise it is possible to skip these sections about MSO and go to Section 2.4.

2.3.3 Syntax of MSO

We want to define the monadic second-order logic over transition systems. A *transition system* is here a directed graph with vertex labels and edge labels. It is composed of:

- a vertex set V ,
- a finite set Prop of atomic propositions, such that $\forall P \in \text{Prop} : P \subseteq V$,

- a finite alphabet T for labeling transitions, such that $\forall a \in T : R_a \subseteq V \times V$ is a transition relation.

If $v \in P$ we say that the proposition $P \in \text{Prop}$ is true in $v \in V$. Formally the label of a vertex v is the set of atomic propositions true in v : $\{P \in \text{Prop} : v \in P\}$.

To write MSO formulas we have a countable set IV of individual variables, denoted x, y, z, \dots ranging over vertices, and a countable set SV of (monadic) set variables, denoted X, Y, Z, \dots ranging over sets of vertices. The syntax is defined inductively as the smallest set of formulas containing

- $P(x)$ for all $x \in IV$ and $P \in \text{Prop}$,
- $x \xrightarrow{a} y$ for all $x, y \in IV$ and $a \in T$,
- $x \in X$ for all $x \in IV$ and $X \in SV$,
- and closed under the connectives $\wedge, \neg, \exists x, \exists X$.

The semantics is defined in the usual way: $x \xrightarrow{a} y$ is true if the instances \hat{x} and \hat{y} of x and y are such that $R_a(\hat{x}, \hat{y})$; Px is true if $\hat{x} \in P$. We use also the abbreviations $\Rightarrow, \Leftrightarrow, \vee, \forall x, \forall X$, and we write $X(x)$ for $x \in X$.

A parity game structure (V_0, V_1, E, Ω) can be translated to a transition system in a straightforward way:

- $V = V_0 \uplus V_1$ is the set of vertices,
- $V_0 \subseteq V$ and $V_1 \subseteq V$ are seen as atomic propositions: $V_0, V_1 \in \text{Prop}$,
- assuming that Ω defines priorities between 0 and $max - 1$, we have an atomic proposition C_i for each $i < max$, such that $\Omega(v) = i \Leftrightarrow v \in C_i$, i.e., $C_i = \{v \in V : \Omega(v) = i\}$,
- the edge relation E will be encoded by several transition relations R_i as explained below.

The point is that if the graph (V, E) has bounded degree one can transform it into a deterministic transition system, where the transition function might be partial. We assume that the out-degree of the graph (V, E) is finite, bounded by m . To differentiate the successors v' of a vertex v , we label the edges from v by different numbers from $\{0, \dots, m-1\}$ and we write $R_i(v, v')$. Formally we have $E = \bigcup_{i < m} R_i$ and the union is disjoint.

Above in the definition of Pushdown Game System we have no labels on the transitions and no input alphabet. Of course it is possible to introduce a sufficiently large input alphabet, which would allow to regain a deterministic (partial) transition function.

2.3.4 MSO Definability of Winning Region and Winning Strategies in Parity Games

We suppose that the MSO-theory of $\mathcal{G} = (V, V_0, V_1, C_0, \dots, C_{max-1}, R_0, \dots, R_{m-1})$ is decidable and that V is composed of *concrete* elements like words. That is to say that we have a procedure that, given a MSO formula over \mathcal{G} , determine if it is true or false. Moreover we assume that this result is based on MSO-definability, which means that given a formula with free variables, one can compute an automaton recognizing the sets that satisfy the formula. This is the case e.g. for pushdown graphs, see [MS85]. We want to write an MSO formula $\text{WinningSet}(W)$ defining the winning region of Player 0. It is essential here that we can restrict ourselves to positional strategies for parity games, and the beginning of the construction can be used also for other games with positional winning strategies.

A *positional strategy* for Player 0 is a function defining for each vertex of Player 0 which successor he should choose. With the help of the edge labeling, a strategy f is represented by disjoint sets of vertices $S_0, \dots, S_{m-1} \subseteq V$ in the following sense:

$$f = \bigcup_{0 \leq i < m} R_i \cap (S_i \times V) .$$

In other words $f(x)$ is the vertex y such that $x \in S_i$ and $x \xrightarrow{i} y$. The following formula checks that the strategy is “deterministic”:

$$\text{Strategy}(S_0, \dots, S_{m-1}) := \forall x \in V : \bigwedge_{i < m} \left(S_i(x) \Leftrightarrow \bigwedge_{j \neq i} \neg S_j(x) \right) .$$

If we consider a strategy for Player 0, then its value on the vertices of Player 1 is not relevant. Given strategies S_0, \dots, S_{m-1} for Player 0 and S'_0, \dots, S'_{m-1} for Player 1, a *play* from $z \in V$ according to these strategies is uniquely defined. It is a path in the game graph represented by a set B of vertices, $B \subseteq V$, which can be finite or infinite, depending whether there is a loop. This set is the minimal set closed by

the “successor” relation of the strategies:

$$\begin{aligned}
& \text{ContainPlay}(z, B, S_0, \dots, S_{m-1}, S'_0, \dots, S'_{m-1}) := \\
& z \in B \wedge \forall x \in B \left(V_0(x) \Rightarrow \bigwedge_{i < m} (S_i(x) \Rightarrow \exists y \in B : x \xrightarrow{i} y) \right) \\
& \wedge \forall x \in B \left(V_1(x) \Rightarrow \bigwedge_{i < m} (S'_i(x) \Rightarrow \exists y \in B : x \xrightarrow{i} y) \right) , \\
& \text{Play}(z, B, S_0, \dots, S_{m-1}, S'_0, \dots, S'_{m-1}) := \\
& \text{ContainPlay}(z, B, S_0, \dots, S_{m-1}, S'_0, \dots, S'_{m-1}) \wedge \\
& \forall B' (\text{ContainPlay}(z, B', S_0, \dots, S_{m-1}, S'_0, \dots, S'_{m-1}) \Rightarrow B \subseteq B') .
\end{aligned}$$

We now look at the parity winning condition. To know whether this play from z is winning for Player 0, one has to check that there is an even priority c , such that after a certain position x , no priority smaller than c appears, and c appears infinitely often.

$$\begin{aligned}
& \text{WinningPlay}(z, B, S_0, \dots, S_{m-1}, S'_0, \dots, S'_{m-1}) := \\
& \text{Play}(z, B, S_0, \dots, S_{m-1}, S'_0, \dots, S'_{m-1}) \wedge \bigvee_{c \in \{0, 2, \dots\}} \exists x \in B \forall B_1 \left[\text{Play}(x, B_1, \dots) \right. \\
& \left. \Rightarrow \forall y_1 \in B_1 \left(\bigvee_{i \geq c} C_i(y_1) \wedge \forall B_2 (\text{Play}(y_1, B_2, \dots) \Rightarrow \exists y_2 \in B_2 : C_c(y_2)) \right) \right] .
\end{aligned}$$

Using the positional (memoryless) determinacy of [EJ91], a vertex is winning for Player 0 if he has a positional strategy to win every play starting from this vertex (against every positional strategy of Player 1):

$$\begin{aligned}
& \text{WinningVertex}(z, S_0, \dots, S_{m-1}) := \\
& \text{Strategy}(S_0, \dots, S_{m-1}) \wedge \forall S'_0, \dots, S'_{m-1} : (\text{Strategy}(S'_0, \dots, S'_{m-1}) \Rightarrow \\
& \exists B \text{ WinningPlay}(z, B, S_0, \dots, S_{m-1}, S'_0, \dots, S'_{m-1})) .
\end{aligned}$$

From the positional determinacy we know also that there is a strategy that is winning on the whole winning region. Such a “best” or “maximal” winning strategy must have the largest set of winning vertices.

$$\begin{aligned}
& \text{WinningStrategy}(S_0, \dots, S_{m-1}) := \forall T_1, \dots, T_{m-1}, \forall z \\
& (\text{WinningVertex}(z, T_0, \dots, T_{m-1}) \Rightarrow \text{WinningVertex}(z, S_0, \dots, S_{m-1})) .
\end{aligned}$$

The “best” strategy is not unique, although the set of winning vertices is unique, and “regular”.

$$\begin{aligned} \text{WinningSet}(W) &:= \\ &\forall z(z \in W \Leftrightarrow \exists(S_0, \dots, S_{m-1}) \text{WinningVertex}(z, S_0, \dots, S_{m-1})) . \end{aligned}$$

This proves that a “best” winning strategy is MSO-definable on a graph of bounded out-degree. The alternation depth of the formulas used is constant, whereas when using the classical translation from μ -calculus to MSO, the alternation depth is the number of colors.

This simple fact about MSO is a particular case of a result from [Cou03]. For *Higher Order Pushdown Graphs*, see Section 5.1.1 and [KNU02].

2.3.5 Discussion and Extensions

We will discuss next which restriction we have made and the possible extensions of this result. This construction is done here for the case of parity game and because we know from [EJ91] that there are positional winning strategies. It can be used also in the special case of Büchi or reachability condition. Moreover we will see later simple proofs that these games have positional winning strategies, based on the computation of attractors.

Another restriction was the boundedness of the degree. We will see however in Section 4.5 that it is sometimes possible to transform a graph with infinite degree into a graph with bounded degree that is “equivalent” in terms of games: a solution to one game give rise to a solution of the other game. This will be used also in Section 5.3.

A posteriori we can say that almost each game that is solved in this thesis can be “solved” by the MSO-formula defined in Section 2.3.4, but the interest of the other methods is to improve the complexity.

2.4 Limitations to Solutions of Games

We mention in this section some limitations to the solvability of games. We consider first a PDGS and a more general goal set, and show that the reachability game is undecidable. Then we consider other classes of graphs where it is known that reachability games are undecidable. In the last subsection we look in detail at an intriguing example.

2.4.1 Intersection of Context-Free Languages

In Section 3.1.1 we will solve the reachability game over pushdown graphs with a regular goal set: one can compute the set of vertices from which Player 0 has a strategy to reach the goal set. We mention here that for a context-free goal set, the situation is much more complicated. For formal definitions, see Section 3.1.

We just remark that the intersection of two context free languages may not be context free. If R_1 and R_2 are two context free languages over $\{a, b, c\}$ and the first move of Player 1 goes from pu to q_1u or q_2u , $u \in \{a, b, c\}^*$, and if the goal set is $q_1R_1 \cup q_2R_2$, then the winning region of Player 0 is $p(R_1 \cap R_2) \cup q_1R_1 \cup q_2R_2$. In this case the winning region is at least recursive: given a configuration one can determine if it is in the winning region. But we have no satisfactory representation of this region, because using classical arguments the emptiness problem for the winning region is undecidable. Given a Turing Machine, it is possible to define R_1 and R_2 such that $R_1 \cap R_2$ is the set of computations of the Turing Machine that ends in a final state, see [HU79, sec. 8.6].

It is possible to refine this example and to construct a pushdown game where in a first phase Player 0 has to push down on the stack a sequence of letters such that he claims that this sequence is a valid computation of a given Turing Machine that ends in a final state, then Player 1 can check that this is true using the same technique as above, going to state q_1 or q_2 . For a given initial configuration of the Turing Machine, and thus of the game, it is in general undecidable whether Player 0 can win this game.

2.4.2 Other Graphs

Infinite state systems come in many versions, with different definitions, see [Bou01] for an overview. Looking at internal representations (where vertices are concrete objects, see [CK02]), the vertices can be coded either by words, like in the case of (configurations of) pushdown graphs, or by more complicated structures like trees. The transition relation can be defined by rewriting rules with different policies. In this framework the pushdown graphs are equivalent to graphs defined by a finite set of prefix rewriting rules over words. Extensions to prefix-recognizable graphs will be presented in Chapter 4, where the prefix rewriting rules are defined by regular languages. In this context parity games are solvable.

Christof Löding has studied in [Löd03] the graphs generated by ground tree rewriting, where each vertex of the graph is represented by a finite tree and the transitions are defined by substitutions of subtree. For these graphs the simple

reachability (where Player 0 makes every decision) is decidable, as well as the model-checking of some fragments of the temporal logic CTL. But it is undecidable in general whether Player 0 has a winning strategy in the reachability game associated to a regular set of vertices. More precisely it is undecidable whether from a given vertex every infinite path visits a fixed set of states. For rational graphs [Mor99] even the simple reachability is undecidable: the reachability game with only one player and a finite goal set.

Other models have also been considered in the verification community. In lossy channel system different components, each modeled by a finite automaton, communicate via lossy channels. Petri nets are another model of concurrent computation where transitions can be fired in parallel at different places of the net. For all these systems model-checking have been widely studied, in particular the simple reachability, whereas the class of graphs where games are decidable is more restricted in general.

The Caucal graphs studied in Chapter 5 are a good track for games: they extend the pushdown and prefix-recognizable graphs and MSO is still decidable, one can solve parity games and compute winning strategies.

2.4.3 A Recursive Game Graph

This example might be considered as artificial, because it is directly related to the halting problem of Turing Machines. It is due to Wolfgang Thomas in [Tho95]. The game graph is a tree, depicted in Figure 2.1. Below the node named i in the figure, there is a switch of colors on the k -th level of the two infinite branches if and only if the i -th Turing Machine M_i halts on the empty tape after k steps. The nodes of Player 0 (in V_0) are circles, the nodes of Player 1 are square. The goal set is the set of nodes that are black filled, and we consider a Büchi condition for Player 0: Player 0 has to visit the goal set infinitely often.

On the rightmost branch, only Player 1 makes choices, but if he stays indefinitely on this branch, he will lose, because the branch is black. At some point he has to choose to go to a node i , $i > 0$. Then Player 0 has to choose the left or right branch. For this he has to know if the i -th Turing Machine ever halts or not, but this problem is known to be undecidable.

This game graph is recursive in the sense that we can find an effective representation of it. For example using words over $\{a, b\}$ to represent vertices, this graph is a subtree of the complete binary tree: the left child of a node $u \in \{a, b\}^*$ is ua and the right child, if it exists, is ub . Then there is a recursive function that computes

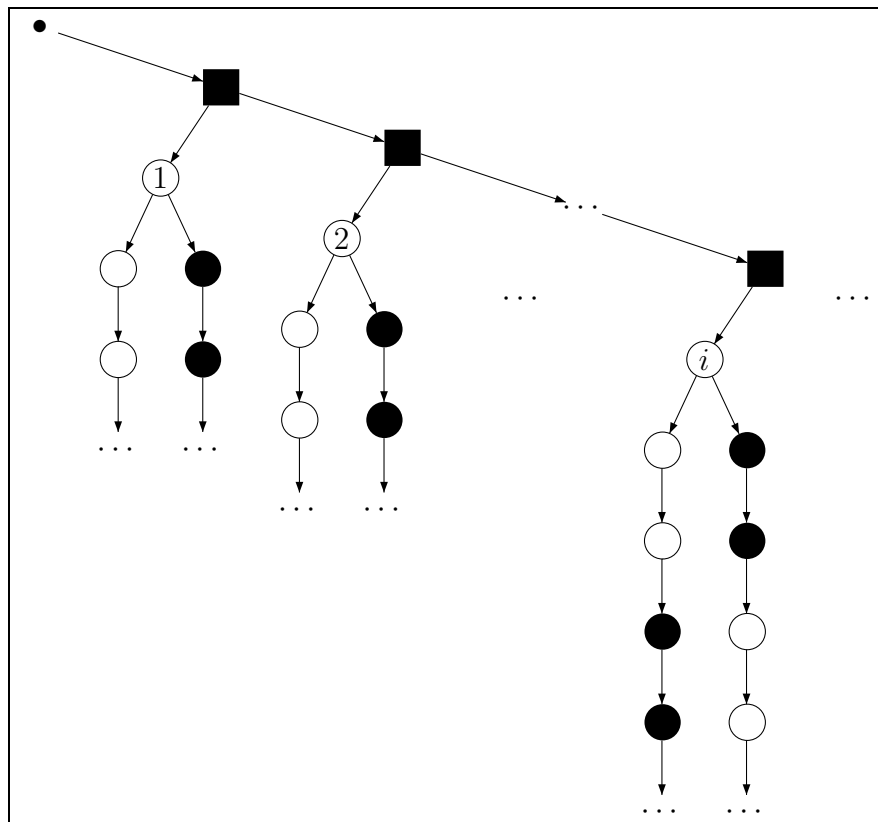


Figure 2.1: Example of a recursive game graph

the color, player and “successors” of any vertex, given its representation.

It is claimed in [Tho95] that Player 0 has a winning strategy that is not effective. The fact is that from a node $i > 0$, one of the branch is winning for Player 0, but he cannot compute which one (and this is typically a classical-logic disjunction involving the law of excluded middle — *Tertium Non Datur*).

Note that this non-computable winning strategy exists for the game starting from the root ; but given a node that is in one of both branches under a node $i > 0$, in general we cannot determine if *there is* a winning strategy from this node. And if there is one, then it is computable, because the strategy is trivial from these nodes. For the same reason even the reachability game associated to the goal set does not admit an effective global (uniform) solution: one cannot compute the whole set of nodes from which the goal set is reachable.

Chapter 3

Symbolic Presentation of Winning Strategies

In this chapter we use a symbolic approach to solve games on pushdown graphs. It is based on the construction of an alternating automaton (with a finite state space) allowing to describe infinite sets of configurations. The framework we use is based on an algorithm by Bouajjani, Esparza and Maler [BEM97] (see also [BEF⁺00, EHRS00a]). We lift their results from CTL and LTL model-checking over pushdown systems to the level of program synthesis, and the synthesis is realized by computing winning strategies of two kinds: positional ones which however require linear execution time in each step, and strategies with pushdown memory where a step can be executed in constant time.

After defining the automata that we will use, we will consider reachability games (Section 3.1), then discuss two kinds of strategies (Section 3.2) and solve the example of the introduction (Section 3.3), later on Büchi games (Section 3.4) and then a new winning condition that we call Σ_3 (Section 3.6).

3.0.4 Technical Preliminaries, \mathcal{P} -Automata

On the symbolic level we will consider here configurations of the PDGS as words, and the transition rules as prefix rewriting rules. For this reason we use now the infix symbol \hookrightarrow for the edge relation of the game graph. Rewriting the definition, given a PDGS $(P_0, P_1, \Gamma, \Delta)$, the associated game graph is (V, \hookrightarrow) , where the set of nodes is the set of *all* configurations: $V = P\Gamma^*$, and the arcs of the game graph are exactly the pairs

$$p\gamma v \hookrightarrow qwv, \text{ for } (p, \gamma, q, w) \in \Delta \text{ and } v \in \Gamma^* .$$

Referring to the case $v = \varepsilon$ we also write rules of Δ in the form $p\gamma \hookrightarrow qw$. In the following γ is always a single letter from Γ .

We describe sets of configurations (and thus also winning conditions in pushdown games) by finite automata. We are thus interested in regular sets of configurations. We define them from alternating \mathcal{P} -automata [EHR00a]: They are alternating word automata with a special convention about initial states.

Definition 3.0.1 *Given a PDGS $\mathcal{P} = (P_0, P_1, \Gamma, \Delta)$, $P = P_0 \cup P_1$, a \mathcal{P} -automaton \mathcal{A} is a tuple $(Q, \Gamma, \longrightarrow, P, F)$, where*

- Q is a finite set of states,
- $\longrightarrow \subseteq Q \times \Gamma \times 2^Q$ a set of transitions, labeled by stack letters of \mathcal{P} ,
- $P \subseteq Q$ a set of initial states (which are taken here as the control locations of \mathcal{P}), and
- $F \subseteq Q$ a set of final states.

For each $p \in P$ and $w \in \Gamma^*$, the automaton \mathcal{A} accepts a configuration pw if and only if there exists a successful \mathcal{A} -run on w from p . Usually a transition has the form $r \xrightarrow{\gamma} \beta$, where β is a positive Boolean formula over Q in Disjunctive Normal Form. To simplify the exposition we allow AND-transitions $r \xrightarrow{\gamma} r_1 \wedge \dots \wedge r_n$, written as $r \xrightarrow{\gamma} \{r_1, \dots, r_n\}$. A transition $r \xrightarrow{\gamma} S$ indicates a move from state r via letter $\gamma \in \Gamma$ simultaneously to all states of S , i.e. by a universal branching of runs. Existential branching (disjunction) are captured by nondeterminism. So a transition like $r \xrightarrow{\gamma} (r_1 \wedge r_2) \vee (r_3 \wedge r_4)$ is represented here by *two* transitions $r \xrightarrow{\gamma} \{r_1, r_2\}$ and $r \xrightarrow{\gamma} \{r_3, r_4\}$.

We define the global transition relation of \mathcal{A} , the reflexive and transitive closure of \longrightarrow , denoted $\longrightarrow^* \subseteq Q \times \Gamma^* \times 2^Q$, as follows:

- $r \xrightarrow{\epsilon}^* \{r\}$, (ϵ is the empty word),
- $r \xrightarrow{\gamma} \{r_1, \dots, r_n\} \wedge \forall i, r_i \xrightarrow{w}^* S_i \Rightarrow r \xrightarrow{\gamma w}^* \bigcup_i S_i$.

The automaton \mathcal{A} accepts the word pw if and only if there *exists* a run $p \xrightarrow{w}^* S$ with $S \subseteq F$, i.e., all finally reached states are final.

In section 3.2 we will need the description of a run. The run trees of an alternating automaton \mathcal{A} (where the branching captures the AND-transitions) can be transformed to “run DAGs” (Directed Acyclic Graphs, see [KV97, LT00]). In such a run DAG, the states occurring on each level of the tree are collected in a set, and

a transition $r \longrightarrow \{r_1, \dots, r_k\}$ connects state r of level i with states $\{r_1, \dots, r_k\}$ of level $i + 1$. Note that every transition of level i is labeled by the same i -th letter of the input word. Let Φ be the set of partial functions from Q to the transition relation \longrightarrow of \mathcal{A} . A run DAG from state p labeled by $w = \gamma_0 \dots \gamma_n$ is described by a sequence $\sigma_0, \dots, \sigma_n$ of elements of Φ and a sequence Q_0, Q_1, \dots, Q_n of subsets of Q , such that $Q_i = \text{Dom}(\sigma_i)$, and from each $q \in Q_i$ the transition $\sigma_i(q)$ is used from q :

$$Q_0 = \{p\} \xrightarrow[\sigma_0]{\gamma_0} Q_1 \xrightarrow[\sigma_1]{\gamma_1} \dots Q_n \xrightarrow[\sigma_n]{\gamma_n} S .$$

So σ_i describes the step $Q_i \xrightarrow{\gamma_i} Q_{i+1}$ by the transitions used. We write shortly $\{p\} \xrightarrow[\sigma]{w} S$, assuming $\sigma = \sigma_0, \dots, \sigma_n$, or just $\{p\} \xrightarrow{w} S$ to denote the run.

3.1 Reachability Game: Computing the Attractor

3.1.1 Reachability

We consider a regular *goal set* $R \subseteq P\Gamma^*$, defined by a \mathcal{P} -automaton \mathcal{A}_R . Player 0 wins a play if and only if it reaches a configuration of R . Our goal is to compute the winning region W_0 of this game: the set of nodes from which Player 0 can force the play to reach the set R or a deadlock for Player 1. To do this we define the operator χ which — using a different terminology — computes the set of “*controllable predecessors*”. Given any set $T \subseteq V$, let

$$\chi(T) = \{u \in V_0 \mid \exists v, u \hookrightarrow v, v \in T\} \cup \{u \in V_1 \mid \forall v, u \hookrightarrow v \Rightarrow v \in T\} ,$$

Now $\chi(T)$ is the set of vertices from which Player 0 has a strategy to reach T in exactly one move (or win because Player 1 is in a deadlock). The set W_0 is clearly the “0-attractor of R ” (see [Tho95]), denoted $\text{Attr}_0(R)$: the least fixed point of the function

$$T \mapsto R \cup \chi(T) .$$

For more background on fixed points, see [GTW02, Ch. 20]. One can also define $\text{Attr}_0(R)$ by induction:

$$\begin{aligned} \text{Attr}_0^0(R) &= R, \\ \text{Attr}_0^{i+1}(R) &= \text{Attr}_0^i(R) \cup \{u \in V_0 \mid \exists v, u \hookrightarrow v, v \in \text{Attr}_0^i(R)\} \\ &\quad \cup \{u \in V_1 \mid \forall v, u \hookrightarrow v \Rightarrow v \in \text{Attr}_0^i(R)\}, \\ \text{Attr}_0(R) &= \bigcup_{i \in \mathbb{N}} \text{Attr}_0^i(R). \end{aligned}$$

According to this definition, we adopt the convention that if the play is in a deadlock (before reaching R), the Player who should play has lost. As the degree of the game graph is finite, an induction on ω is sufficient. The definition by fixed point is not effective in general if we consider infinite graphs.

Remark 3.1.1 *From this definition it is clear also that both players have positional winning strategies on their winning regions:*

- *Player 1 wins from a configuration $u \in V$ if and only if $u \notin \text{Attr}_0(R)$. A winning strategy consists, from any vertex $u \in V_1 \setminus \text{Attr}_0(R)$, to stay outside of $\text{Attr}_0(R)$: choose v such that $u \hookrightarrow v$ and $v \notin \text{Attr}_0(R)$.*
- *Player 0 wins from a configuration $u \in V$ if and only if $u \in \text{Attr}_0(R)$. A winning strategy consists, from a vertex $u \in V_0 \cap \text{Attr}_0(R)$, to determine the minimal i such that $u \in \text{Attr}_0^i(R)$ and move to $\text{Attr}_0^{i-1}(R)$.*

Our task is to transform a given automaton \mathcal{A}_R recognizing R into an automaton $\mathcal{A}_{\text{Att}(R)}$ recognizing $\text{Attr}_0(R)$. Without loss of generality, we can assume that there is no transition in \mathcal{A}_R leading to an initial state (a state of P).

Algorithm 3.1.2 (saturation procedure)

Input: a PDGS \mathcal{P} , a \mathcal{P} -automaton \mathcal{A}_R that recognizes the goal set R , without transition to the initial states.

Output: a \mathcal{P} -automaton $\mathcal{A}_{\text{Att}(R)}$ that recognizes $\text{Attr}_0(R)$.

Let $\mathcal{A}_{\text{Att}(R)} := \mathcal{A}_R$. Transitions are added to $\mathcal{A}_{\text{Att}(R)}$ according to the following saturation procedure.

repeat

(Player 0) if $p \in P_0$, $p\gamma \hookrightarrow qv$ and $q \xrightarrow{v}^* S$ in $\mathcal{A}_{\text{Att}(R)}$, then add a new transition $p \xrightarrow{\gamma} S$.

(Player 1) if $p \in P_1$, $\left\{ \begin{array}{l} p\gamma \hookrightarrow q_1v_1 \\ \vdots \\ p\gamma \hookrightarrow q_nv_n \end{array} \right.$ are all the moves (rules) starting from

$p\gamma$ and $\left\{ \begin{array}{l} q_1 \xrightarrow{v_1} S_1 \\ \vdots \\ q_n \xrightarrow{v_n} S_n \end{array} \right.$ in $\mathcal{A}_{Att(R)}$, then add a new transition $p \xrightarrow{\gamma} \bigcup_i S_i$.

until no new transition can be added.

Note that $\mathcal{A}_{Att(R)}$ has exactly the same state space as \mathcal{A}_R . The algorithm eventually stops because there are only finitely many possible new transitions, and the “saturation” consists in adding as many transitions as possible. The idea of adding a new transition $p \xrightarrow{\gamma} S$ for $p \in P_0$ is that, if $qv \in Attr_0(R)$, and $p\gamma \hookrightarrow qv$, then $p\gamma \in Attr_0(R)$ too, and then $p\gamma$ should have the same behavior as qv in the automaton. For $p \in P_1$, $Attr_0(R)$ is defined by a conjunction, expressed in $\mathcal{A}_{Att(R)}$ by the AND-transition. The algorithm and the proof are a generalization of [BEF⁺00, EHR00a] from nondeterministic automata (for simple reachability) to alternating automata (for game reachability). In [BEF⁺00], one deals with the case $P = P_0$, and the “winning region” is the set of “predecessors” of R , denoted $pre^*(R)$. Note that the simple reachability can be solved in polynomial time (see also [Sch02]), whereas the reachability games are EXPTIME-complete (see [Wal96b]). In [BEM97], alternating (pushdown) automata were already considered, but they were not used to solve a game, and winning strategies were not treated.

Clearly the algorithm runs in time $\mathcal{O}(|\Delta| 2^{c|Q|^2})$, where $|\Delta|$ is the sum of the lengths of the rules in Δ . An implementation of Algorithm 3.1.2 was developed in [Cor01].

Theorem 3.1.3 *The automaton $\mathcal{A}_{Att(R)}$ constructed by Algorithm 3.1.2 recognizes the set $Attr_0(R)$, if \mathcal{A}_R recognizes R .*

Proof: We consider the step-by-step construction of $\mathcal{A}_{Att(R)}$:

$$\mathcal{A}_R = \mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_2, \dots, \mathcal{A}_m = \mathcal{A}_{Att(R)},$$

where $\forall i < m$, “ $\mathcal{A}_{i+1} = \mathcal{A}_i \cup \{p \xrightarrow{\gamma} S\}$ ”, that is to say, exactly one transition is added.

The set $Attr_0(R)$ was defined by induction:

$$\begin{aligned} Attr_0^0 &= R, \\ Attr_0^{i+1} &= Attr_0^i \cup \{pw \mid p \in P_0, \exists pw \hookrightarrow qv, qv \in Attr_0^i\} \\ &\quad \cup \{pw \mid p \in P_1, \forall pw \hookrightarrow qv, qv \in Attr_0^i\}, \\ Attr_0(R) &= \bigcup_{i \in \mathbb{N}} Attr_0^i. \end{aligned}$$

We note $L(\mathcal{A}_m)$ the language recognized by \mathcal{A}_m .

First part: $L(\mathcal{A}_m) \supseteq Attr_0(R)$.

We use an induction on i to show that $\forall i \geq 0, L(\mathcal{A}_m) \supseteq Attr_0^i$.

- For $i = 0$, $R = Attr_0^0 = L(\mathcal{A}_0) \subseteq L(\mathcal{A}_m)$, because the transitions of \mathcal{A}_0 are still present in \mathcal{A}_m .
- Induction hypothesis: $L(\mathcal{A}_m) \supseteq Attr_0^i$ for some i .
- Then consider $pw \in Attr_0^{i+1} \setminus Attr_0^i$.

- First case: $p \in P_0$.

By the definition of $Attr_0^{i+1}$,

$$\exists pw \hookrightarrow qv, qv \in Attr_0^i \subseteq L(\mathcal{A}_m).$$

Thus there is a path $q \xrightarrow{v} S$, $S \subseteq F$. We decompose the transition \hookrightarrow : $\exists \gamma \in \Gamma$, $w = \gamma u$, $v = v'u$, $p\gamma \hookrightarrow qv'$; and the path $q \xrightarrow{v} S$:

$$q \xrightarrow{v'} S' \xrightarrow{u} S.$$

By definition of Algorithm 3.1.2,

$$q \xrightarrow{v'} S' \wedge p\gamma \hookrightarrow qv' \Rightarrow \exists \text{ transition } p \xrightarrow{\gamma} S' \text{ in } \mathcal{A}_m.$$

As a consequence there is a path $p \xrightarrow{\gamma} S' \xrightarrow{u} S$ in \mathcal{A}_m , and so $p \xrightarrow{w} S \subseteq F$, $pw \in L(\mathcal{A}_m)$.

- Second case: $p \in P_1$.

By the definition of $Attr_0^{i+1}$, $\forall pw \hookrightarrow qv, qv \in Attr_0^i$. More precisely,

$$\text{all the arcs starting from } p\gamma u \text{ are } \left\{ \begin{array}{l} p\gamma u \hookrightarrow q_1 v_1 u \\ \vdots \\ p\gamma u \hookrightarrow q_n v_n u \end{array} \right.$$

and $\forall j, q_j v_j u \in Attr_0^i \subseteq L(\mathcal{A}_m)$.

Thus there are paths $q_j \xrightarrow{v_j u} S_j \subseteq F$, that we decompose into

$$q_j \xrightarrow{v_j} S'_j \xrightarrow{u} S_j .$$

In the construction of \mathcal{A}_m , a new transition $p \xrightarrow{\gamma} \bigcup_j S'_j$ was added, and so

$$p \xrightarrow{\gamma} \bigcup_j S'_j \xrightarrow{u} \bigcup_j S_j \subseteq F \Rightarrow p\gamma u \in L(\mathcal{A}_m) .$$

From the induction hypothesis we can conclude that $L(\mathcal{A}_m) \supseteq Attr_0(R)$.

Second part: $L(\mathcal{A}_m) \subseteq Attr_0(R)$.

For the detailed proof we refer to the proof of Lemma 3.2.5, where the additional parameter $Cost$ and the number of moves are treated. We thus give here only the sketch of the proof.

We can prove by induction on m that $\forall p \in P, w \in \Gamma^*$, if $p \xrightarrow{w} S$ in \mathcal{A}_m , then starting from pw , Player 0 can reach a configuration in the set

$$\left\{ p'w' \in P\Gamma^* \mid \exists S' \subseteq S, p' \xrightarrow{\mathcal{A}_0} S' \right\}$$

whatever Player 1 does (in between). That is to say the play starting from pw will reach after some steps the given set *if Player 0 wants to*, but Player 0 can not choose which element of this set will be reached (this is more or less the choice of Player 1).

We denote $\xrightarrow{\mathcal{A}_0}$ the global transition relation of \mathcal{A}_0 .

In particular if $p \xrightarrow{w} S \subseteq F$ in \mathcal{A}_m , then from pw Player 0 can reach

$$\left\{ p'w' \mid p' \xrightarrow{\mathcal{A}_0} S' \subseteq S \subseteq F \right\} \subseteq R ,$$

which proves that he can win. ■

We still have to say a few words about deadlocks. According to the definitions of $Attr_0$, if the play is in a deadlock, the player who is on has immediately lost. In a pushdown graph, there are two types of deadlocks: when the stack is empty, or when the first letter of the stack and the control state does not permit to make a transition. The second case is not a problem: if there is *no* transition $p\gamma \hookrightarrow q\mu$, and $p \in P_1$, the saturation algorithm adds a transition $p \xrightarrow{\gamma} \emptyset$ (allowing to accept every configuration starting with $p\gamma$). But we have to think about the case of empty stack. To follow strictly the definition of $Attr_0$ the states p have to be set as accepting for

each $p \in P_1$ (empty stack for Player 1). If one needs a bottom stack symbol (\perp), then it has to be defined explicitly in Γ and Δ and treated as a stack letter (that can neither be erased nor pushed).

We have chosen a *regular* goal set R , and proved that $Attr_0(R)$ is also regular. If we consider a context free goal set R , the situation diverges for the cases of simple reachability and game reachability:

Proposition 3.1.4 *If the goal set R is a context free language, then $pre^*(R)$ is also context free, but $Attr_0(R)$ is not necessarily context free.*

The first part can be deduced from [Cau90]. The second part was already exposed in Section 2.4.1.

3.1.2 Determining Membership in the Attractor

In [KV00] and [Wal96b] (see Chapter 4), given a PDGS and an initial position of the game, an EXPTIME procedure (in the size of the description of the game) determines if it is in the winning region W_0 of Player 0. In contrast our solution is uniform: after a single EXPTIME procedure, we can determine in linear time if any given configuration is in the winning region W_0 .

To use the results of the preceding subsection, we still have to determine whether a given configuration belongs to $Attr_0(R)$. We can use a polynomial time algorithm, that searches backward all the accepting runs of the automaton $\mathcal{A}_{Attr(R)}$ (from now on, we skip corresponding claims for Player 1). We repeat here the classical algorithm because variants of it will be used in the next section. The correctness proof is easy and omitted here.

Algorithm 3.1.5 (Membership)

Input: an alternating \mathcal{P} -automaton $\mathcal{B} = (Q, \Gamma, \longrightarrow, P, F)$ recognizing $Attr_0(R) = L(\mathcal{B})$, a configuration $pw \in P\Gamma^*$, $w = a_1 \dots a_n$.

Output: Answer whether $pw \in L(\mathcal{B})$ or $pw \notin L(\mathcal{B})$.

Let $S := F$;

for $i := n$ **down to** 1 **do** $S := \{s \in Q \mid \exists (s \xrightarrow{a_i} X) \text{ in } \mathcal{B}, X \subseteq S\}$ **end for**

If $p \in S$, answer “ $pw \in L(\mathcal{B})$ ” else answer “ $pw \notin L(\mathcal{B})$ ”

The space complexity of Algorithm 3.1.5 is $\mathcal{O}(|Q|)$, the time complexity is $\mathcal{O}(nm|Q|)$ where m is the number of transitions of \mathcal{B} , and n is the length of the input configuration.

3.2 Winning Strategy for Player 0

Given an initial configuration, we can now determine if Player 0 can win the game starting from it. Concretely, it remains to determine which moves Player 0 has to choose in order to win, depending on the moves of Player 1. In the following subsections we first give some preparation, then present a positional strategy, prove that it is winning, also achieving the proof of Theorem 3.1.3, and then present a pushdown winning strategy.

3.2.1 Preparation

A move of Player 0 consists in a choice of a PDGS-Rule. Given a configuration $pw \in Attr_0(R)$, our aim is to extract such a choice from an accepting run of $\mathcal{A}_{Att(R)}$ on pw . In Algorithm 3.1.2, a new transition $p \xrightarrow{\gamma} S$ of $\mathcal{A}_{Att(R)}$ is generated by a (unique) rule $p\gamma \hookrightarrow qv$ of the PDGS under consideration, if $p \in P_0$. We extend now the algorithm so that it computes the partial function $Rule$ from \longrightarrow to Δ . This function remembers the link between a new transition of the finite automaton for $Attr_0(R)$ and the rule of the Pushdown Graph that was used to construct it. We shall write in the algorithm $Rule(p \xrightarrow{\gamma} S) := p\gamma \hookrightarrow qv$. For transitions $p \xrightarrow{\gamma} S$ of the original automaton \mathcal{A} , $Rule(p \xrightarrow{\gamma} S)$ is undefined.

Now, given a configuration $p\gamma w \in V_0$ accepted by $\mathcal{A}_{Att(R)}$, with a run $\{p\} \xrightarrow{\gamma} S \xrightarrow{w_*} T$ (and if $p\gamma w \notin R$), a first idea would be to choose the move $Rule(p \xrightarrow{\gamma} S) = p\gamma \hookrightarrow qv$, (*hoping* to get closer to R). Unfortunately this does not, in general, define a winning strategy. Still it ensures that we remain in the winning region. The following example illustrates this situation.

Example 3.2.1 Let $\Gamma = \{a\}$, $P = P_0 = \{p\}$, $P_1 = \emptyset$ and $\Delta = \{pa \hookrightarrow p, pa \hookrightarrow paa\}$.

It is clear that Player 0 can add and remove as many a 's as he wants. Let $R = \{pa^3\}$ (R is regular), then the winning region is $Attr_0(R) = pa^+$, as shown by the automaton in Figure 3.1, from Algorithm 3.1.2. There are two different runs of $\mathcal{A}_{Att(R)}$ that accept paa : through transitions (1)(3), or through (2). The strategy associated to (1)(3) plays to pa , and therefore is not successful. To define a winning strategy using finite automaton $\mathcal{A}_{Att(R)}$, we need to select the most suitable run on a given configuration, or to remember information about an accepting run, to play coherently the following moves. We give two solutions: the first one, a positional strategy, associates a cost to each transition added while constructing $\mathcal{A}_{Att(R)}$, in

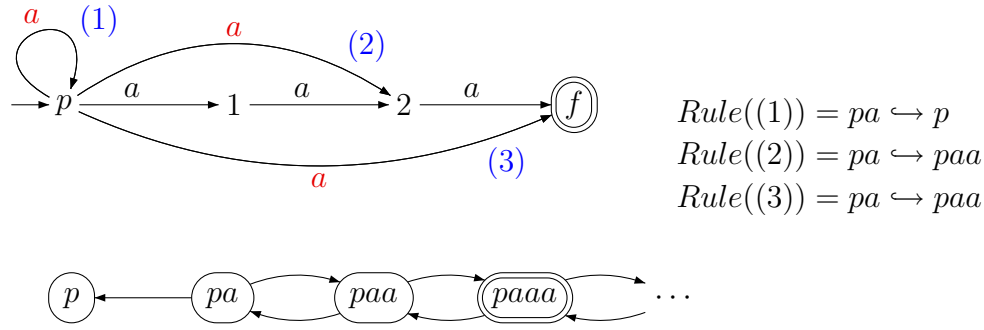


Figure 3.1: Automaton from Algorithm 3.1.2 and Example 3.2.1, function $Rule$, game graph

order to compute the distance to R . The second one, a pushdown strategy, uses a stack to remember how $\mathcal{A}_{Att(R)}$ accepts the current configuration.

3.2.2 Positional Min-rank Strategy

The *rank* of a configuration pw is the smallest i such that $pw \in Attr_0^i(R)$ (it is ∞ if $pw \notin Attr_0(R) = W_0$). It is the “distance” of the configuration pw to R . In the following we consider only configurations in W_0 . Then Player 0 will be able, from a configuration in $Attr_0^i(R)$, to move to $Attr_0^{i-1}(R)$, and Player 1 does this with each possible move. In order to implement this, during the construction of $\mathcal{A}_{Att(R)}$ we will attribute to each $\mathcal{A}_{Att(R)}$ -transition τ a cost $Cost(\tau)$. Initially, each transition of \mathcal{A}_R has the cost 0 (with these transitions $\mathcal{A}_{Att(R)}$ recognizes configurations that are already in R).

The function $Cost$ from the transition relation \longrightarrow of $\mathcal{A}_{Att(R)}$ to \mathbb{N} is extended to a function $Cost^*$ from the run DAGs to \mathbb{N} . Given a fixed run $\{q\} \xrightarrow{w} S$ of the automaton \mathcal{A}_i (obtained at step i in the construction of $\mathcal{A}_{Att(R)}$), its cost $Cost^*(\{q\} \xrightarrow{w} S)$ is the maximal sum of the costs of the transitions along a single path (branch) of the run DAG $\{q\} \xrightarrow{w} S$. Inductively $Cost^*$ is defined by the following clauses:

$$\begin{aligned}
 Cost^*(\{q\} \xrightarrow{\varepsilon} \{q\}) &= 0 \\
 Cost^*(\{q\} \xrightarrow{\gamma} \{q_1, \dots, q_n\} \xrightarrow{u} \bigcup_i S_i) &= \\
 &Cost(q \xrightarrow{\gamma} \{q_1, \dots, q_n\}) + \max_{1 \leq i \leq n} (Cost^*(\{q_i\} \xrightarrow{u} S_i)) .
 \end{aligned}$$

When adding a new transition $p \xrightarrow{\gamma} S$ to \mathcal{A}_i , to obtain \mathcal{A}_{i+1} , its cost is computed by an extension of Algorithm 3.1.2, using the costs of the existing transitions. In the main loop of Algorithm 3.1.2, we add the following assignments:

- if $p \in P_0 \dots$, let $Cost(p \xrightarrow{\gamma} S) := 1 + Cost^*(\{q\} \xrightarrow{v} S)$,
- if $p \in P_1 \dots$, let $Cost(p \xrightarrow{\gamma} \bigcup_i S_i) := 1 + \max_j(Cost^*(\{q_j\} \xrightarrow{v_j} S_j))$.

We repeat now the algorithm with the new features.

Algorithm 3.2.2 (saturation procedure with functions $Cost$ and $Rule$)

Input: a PDGS \mathcal{P} , a \mathcal{P} -automaton \mathcal{A}_R that recognizes the goal set R , without transition to the initial states.

Output: a \mathcal{P} -automaton $\mathcal{A}_{Att(R)}$ that recognizes $Attr_0(R)$, with functions $Cost$ and $Rule$)

Let $\mathcal{A}_{Att(R)} := \mathcal{A}_R$. Transitions are added to $\mathcal{A}_{Att(R)}$ according to the following saturation procedure.

repeat

(Player 0) if $p \in P_0$, $p\gamma \hookrightarrow qv$ and $q \xrightarrow{v} S$ in $\mathcal{A}_{Att(R)}$, then add a new transition $p \xrightarrow{\gamma} S$.

Let $Rule(p \xrightarrow{\gamma} S) := p\gamma \hookrightarrow qv$ and $Cost(p \xrightarrow{\gamma} S) := 1 + Cost^*(\{q\} \xrightarrow{v} S)$,

(Player 1) if $p \in P_1$, $\begin{cases} p\gamma \hookrightarrow q_1v_1 \\ \vdots \\ p\gamma \hookrightarrow q_nv_n \end{cases}$ are all the moves (rules) starting from

$p\gamma$ and $\begin{cases} q_1 \xrightarrow{v_1} S_1 \\ \vdots \\ q_n \xrightarrow{v_n} S_n \end{cases}$ in $\mathcal{A}_{Att(R)}$, then add a new transition $p \xrightarrow{\gamma} \bigcup_i S_i$.

Let $Cost(p \xrightarrow{\gamma} \bigcup_i S_i) := 1 + \max_j(Cost^*(\{q_j\} \xrightarrow{v_j} S_j))$.

until no new transition can be added.

The significance of $Cost^*$ follows clearly from next proposition:

Proposition 3.2.3 For any configuration $pw \in Attr_0(R)$,

$$rank(pw) = \min\{Cost^*(\{p\} \xrightarrow{w} S) \mid \{p\} \xrightarrow{w} S \subseteq F \text{ in } \mathcal{A}_{Att(R)}\} .$$

In the Example 3.2.1, one gets $Cost((1)) = 1$, $Cost((2)) = 1$, $Cost((3)) = 2$. So using the transitions (1) and (3) is not the best way to accept paa , and transition (2) is taken. We are now able to define the desired strategy.

Min-rank Strategy for Player 0

Input: alternating automaton $\mathcal{A}_{Attr_0(R)}$ for $Attr_0(R)$, functions $Rule$ and $Cost$ (as computed from Algorithm 3.1.2 from PDGS \mathcal{P}), configuration $pw \in Attr_0(R)$, $p \in P_0$.

Output: “next move” from configuration pw .

Find an accepting run $\{p\} \xrightarrow{w} S \subseteq F$ of $\mathcal{A}_{Attr_0(R)}$ with minimal cost $Cost(\{p\} \xrightarrow{w} S)$.

If the cost is 0, $pw \in R$ and the play is won, else decompose this run: $w = \gamma w'$, $\{p\} \xrightarrow{\gamma} T \xrightarrow{w'} S$, and choose the rule $Rule(p \xrightarrow{\gamma} T)$.

Theorem 3.2.4 *Given a PDGS with a reachability condition, a regular goal set R , and an alternating automaton $\mathcal{A}_{Attr_0(R)}$ for $Attr_0(R)$, functions $Rule$ and $Cost$ (as computed from Algorithm 3.1.2), the min-rank strategy is positional, winning from all configurations of the winning region W_0 of Player 0. It can be computed in time $\mathcal{O}(n)$ in the length n of the input configuration.*

For the proofs of Theorem 3.2.4 and Proposition 3.2.3 we need the following lemma:

Lemma 3.2.5 *If a node pw is accepted by $\mathcal{A}_{Attr_0(R)}$ with a run $\{p\} \xrightarrow{w} S \subseteq F$, then from this node Player 0 can join R in at most $Cost^*(\{p\} \xrightarrow{w} S)$ steps.*

The proof of this lemma is in the next subsection (3.2.3).

Proof: (Proposition 3.2.3) From the previous facts it follows that

$$rank(pw) \leq \min\{Cost^*(\{p\} \xrightarrow{w} S) \mid \{p\} \xrightarrow{w} S \subseteq F\} .$$

For the converse inequality, it is easy to show by induction that for all $i \geq 0$

$$rank(pw) = i \Rightarrow \min\{Cost^*(\{p\} \xrightarrow{w} S) \mid \{p\} \xrightarrow{w} S \subseteq F\} \geq i .$$

■

Proof: (Theorem 3.2.4) According to the strategy, if $p \in P_0$, Player 0 has to find the cheapest run on \mathcal{A}_m . Decompose this run: $w = \gamma w'$,

$$\{p\} \xrightarrow{\gamma} T \xrightarrow{w'} S$$

choose the transition $Rule(p \xrightarrow{\gamma} T) = p\gamma \leftrightarrow qv$ that were used to construct the transition $p \xrightarrow{\gamma} T$ in $\mathcal{A}_{Attr_0(R)}$. After that the play is in the state qvw' . We can

remark that

$$Cost^* (\{p\} \xrightarrow{\gamma} T \xrightarrow{w'}_* S) = 1 + Cost^* (\{q\} \xrightarrow{v} T \xrightarrow{w'}_* S) .$$

So the “distance” to R has decreased. It is the same if $p \in P_1$ (for each possible move of Player 1). The maximal number of steps needed to reach R is decreasing, so the play will eventually reach R , see Lemma 3.2.5. (and it is possible, if Player 1 plays “badly”, that the number of steps decreases faster) ■

The Theorem 3.2.4 has been proved without using the Proposition 3.2.3, with the previous lemma, but the proposition states that the strategy is optimal.

Algorithm 3.1.5 can be easily extended to compute the distance to R and the strategy. By Proposition 3.2.3, the min-rank strategy is optimal in the sense that it finds a shortest path to R . It reevaluates its choices at each step of the game (particularly if Player 1 goes much “closer” to R than needed). We will present in Section 3.2.4 a strategy that is not necessarily optimal but easier to compute. Before that the next subsection gives the proof of Lemma 3.2.5.

3.2.3 Proof of Lemma 3.2.5

This is also the second part of the proof of Theorem 3.1.3, if we forget all about the notions of cost ($Cost$) and number of moves.

We will prove by induction on m that $\forall p \in P, w \in \Gamma^*$, if there is a run $\{p\} \xrightarrow{w}_* S$ in \mathcal{A}_m , then starting from pw , Player 0 can reach a configuration in the set

$$\left\{ p'w' \in P\Gamma^* \mid \exists S' \subseteq S, p' \xrightarrow{\mathcal{A}_0} w' S' \right\}$$

after no more than $Cost(p \xrightarrow{w}_* S)$ moves, whatever Player 1 does. We are counting the moves of both players. In particular if there is a run $\{p\} \xrightarrow{w}_* S \subseteq F$ in \mathcal{A}_m , then from pw Player 0 can *win* after no more than $Cost^*(\{p\} \xrightarrow{w}_* S)$ moves.

Recall first that we have supposed that in \mathcal{A}_0 no transition is leading to an initial state (a state of $P \subseteq Q$), and so

$$p' \xrightarrow{\mathcal{A}_0} w' S', S' \cap P \neq \emptyset \implies w' = \epsilon, p' \in S' . \quad (3.1)$$

We denote

$$\begin{aligned} \xrightarrow{w}_m & \text{ the global transition relation of } \mathcal{A}_m , \\ \xrightarrow{w}_0 & \text{ the global transition relation of } \mathcal{A}_0 = \mathcal{A}_R . \end{aligned}$$

We write $p \xrightarrow{w}^* S$ for the *existence* of a run from p to S labeled by w , whereas $\{p\} \xrightarrow{w}^* S$ denotes a fixed run. Induction on m :

- For $m = 0$, $p \xrightarrow{w}^* S \iff p \xrightarrow{w}_0^* S$, and from pw Player 0 can guarantee that pw is reached with 0 move. In fact $Cost^*(\{p\} \xrightarrow{w}_0^* S) = 0 \geq 0$.
- Assume it is true for some $m \geq 0$,
- then we are considering \mathcal{A}_{m+1} :

$$\xrightarrow{m+1} = \xrightarrow{m} \cup \{t\}, t = p_0 \xrightarrow{\gamma} S_0,$$

$Cost(t)$ is defined by Algorithm 3.2.2.

Let $p \xrightarrow{w}^* S$.

We are now using an induction on j , the number of times that t is used in the run $\{p\} \xrightarrow{w}^* S$.

- If $j = 0$, then $p \xrightarrow{w}^* S$. From the induction hypothesis on m , we get the result.
- Suppose it is true for some $j \geq 0$.
- Consider that t is used $j + 1$ times in $\{p\} \xrightarrow{w}^* S$. Decompose $w = u\gamma v$, such that:

$$\begin{array}{ccccc} \{p\} & \xrightarrow{u}^* & T_1 & \xrightarrow{\gamma} & T_2 & \xrightarrow{v}^* & S \\ & & p_0 & \xrightarrow{\gamma} & S_0 & & \end{array} \quad (3.2)$$

with $p_0 \in T_1$, $S_0 \subseteq T_2$, and $t = (p_0 \xrightarrow{\gamma} S_0)$ “is used” in $T_1 \xrightarrow{\gamma} T_2$, for the “first time” in the run $\{p\} \xrightarrow{w}^* S$.

From the induction on m , $\{p\} \xrightarrow{u}^* T_1$ implies:

$$\begin{array}{l} pu \text{ guarantees } \{p_1 v_1 \mid p_1 \xrightarrow{v_1}_0^* T'_1 \subseteq T_1\} \\ \text{with no more moves than } Cost^*(\{p\} \xrightarrow{u}^* T_1). \end{array} \quad (3.3)$$

From (3.2) we have also

$$\begin{array}{ccc} T_1 \setminus \{p_0\} & \xrightarrow{\gamma} & T'_2 & \xrightarrow{v}^* & T_3 \subseteq S \\ & & | \cap & & \\ & & T_2 & & \end{array}$$

The new transition t is used in the last formula (see \xrightarrow{v}^*) less often than in (3.2) (less than $j + 1$ times), so from the induction hypothesis on j , we obtain:

from each configuration $t_1 \gamma v$ in $(T_1 \setminus \{p_0\}) \gamma v$ Player 0 can reach the set

$$\begin{array}{l} \{p_3 w_3 \mid p_3 \xrightarrow{w_3}_0^* T'_3 \subseteq T_3\} \\ \text{with no more moves than } Cost^*(\{t_1\} \xrightarrow{\gamma} T''_2 \xrightarrow{v}^* T'_3). \end{array} \quad (3.4)$$

For the rest of the proof, we will distinguish two cases:

case 0: $p_0 \in P_0$. By definition of \mathcal{A}_{m+1} ,

$$p_0\gamma \hookrightarrow q_0w_0, \quad (3.5)$$

$$q_0 \xrightarrow{\frac{w_0}{m}^*} S_0$$

$$\text{and } Cost(t) = 1 + Cost^* \left(\{q_0\} \xrightarrow{\frac{w_0}{m}^*} S_0 \right). \quad (3.6)$$

Together with (3.2) we get

$$\begin{array}{c} \{q_0\} \xrightarrow{\frac{w_0}{m}^*} S_0 \xrightarrow{\frac{v}{m+1}^*} U_0 \subseteq S \\ | \cap \\ T_2 \end{array}$$

The new transition t is used in the last formula (see $\frac{v}{m+1}^*$) less often than in (3.2), so from the induction hypothesis on j , we obtain:

$$\begin{array}{l} q_0w_0v \text{ guarantees } \{t_0x_0 \mid t_0 \xrightarrow{\frac{x_0}{0}^*} D_0 \subseteq U_0\} \\ \text{with no more moves than } Cost^* \left(\{q_0\} \xrightarrow{\frac{w_0}{m}^*} S_0 \xrightarrow{\frac{v}{m+1}^*} U_0 \right). \end{array} \quad (3.7)$$

case 1: $p_0 \in P_1$. By definition of \mathcal{A}_{m+1} ,

$$\left\{ \begin{array}{l} p_0\gamma \hookrightarrow q_1w_1 \\ \vdots \quad \quad \quad \vdots \\ p_0\gamma \hookrightarrow q_nw_n \end{array} \right. \text{ are all the moves from } p_0\gamma \text{ and } \forall i \ q_i \xrightarrow{\frac{w_i}{m}^*} S_i, \quad (3.8)$$

$$S_0 = \bigcup_i S_i,$$

$$Cost(t) = 1 + \max_i \left(Cost^* \left(\{q_i\} \xrightarrow{\frac{w_i}{m}^*} S_i \right) \right) \quad (3.9)$$

Together with (3.2) we get

$$\begin{array}{c} \forall i, \quad \{q_i\} \xrightarrow{\frac{w_i}{m}^*} S_i \xrightarrow{\frac{v}{m+1}^*} U_i \subseteq S \\ | \cap \\ S_0 \\ | \cap \\ T_2 \end{array}$$

The new transition t is used less often than in (3.2), so we have (induction on j):

$$\begin{aligned} \forall i, q_i w_i v \text{ guarantees } \{t_i x_i \mid t_i \xrightarrow{0}^* D_i \subseteq U_i\} \\ \text{with no more moves than } Cost^* \left(\{q_i\} \xrightarrow{m}^* S_i \xrightarrow{m+1}^* U_i \right). \end{aligned} \quad (3.10)$$

Now we can put all proved facts together.

From (3.3), we have in particular: $p_1 v_1 \gamma v$ guarantees a configuration $p_1 v_1 \gamma v$ such that $p_1 \xrightarrow{0}^* T'_1 \subseteq T_1$, with no more than $Cost^* (\{p\} \xrightarrow{m}^* T_1)$ moves. We distinguish the two following cases

either the path $\{p_1\} \xrightarrow{0}^* T'_1 \subseteq T_1$ is not leading to any initial state of \mathcal{A}_0 , *i.e.*, $T'_1 \cap P = \emptyset$, then with (3.2) the path

$$\begin{array}{ccccccc} \{p_1\} & \xrightarrow{0}^* & T'_1 & \xrightarrow{m+1} & T''_2 & \xrightarrow{m+1}^* & S' \subseteq S \\ & & | \cap & & | \cap & & \\ & & T_1 & & T_2 & & \end{array}$$

uses only transitions that were already in \mathcal{A}_0 (the new transitions are always starting from an initial state, of $P \subseteq Q$). It follows that

$$p_1 \xrightarrow{0}^{v_1 \gamma v} S' \subseteq S,$$

Note that $Cost^* (\{p_1\} \xrightarrow{0}^{v_1 \gamma v} S') = 0 = Cost^* (\{p_1\} \xrightarrow{m+1}^{v_1 \gamma v} S')$.

or the path $\{p_1\} \xrightarrow{0}^* T'_1 \subseteq T_1$ is actually leading to an initial state of \mathcal{A}_0 (in $T'_1 \cap P$), then $v_1 = \epsilon$, $p_1 \in T'_1$ (see (3.1)), and $p_1 v_1 \gamma v = p_1 \gamma v$. We consider two sub-cases (α) and (β).

(α) If $p_1 \in T_1 \setminus \{p_0\}$, then by (3.4),

$$\begin{aligned} p_1 \gamma v \text{ guarantees } \{p_3 w_3 \mid p_3 \xrightarrow{0}^* T'_3 \subseteq T_3 \subseteq S\} \\ \text{with no more moves than } Cost^* \left(\{t_1\} \xrightarrow{m} T''_2 \xrightarrow{m+1}^* T'_3 \right). \end{aligned}$$

(β) If $p_1 = p_0$ ($\in T_1 \cap P$)

case 0: $p_0 \in P_0$, Player 0 can choose, from $p_0 \gamma v$, to go to $q_0 w_0 v$ (see (3.5)), *in one move* and with (3.7),

$$\begin{aligned} q_0 w_0 v \text{ guarantees } \{t_0 x_0 \mid t_0 \xrightarrow{0}^* D_0 \subseteq U_0 \subseteq S\} \\ \text{with no more moves than } Cost^* \left(\{q_0\} \xrightarrow{m}^* S_0 \xrightarrow{m+1}^* U_0 \right). \end{aligned}$$

case 1: $p_0 \in P_1$, Player 0 just can wait, but from $p_0\gamma v$ Player 1 can only choose *with one move* one of the states $q_i w_i v$ (see (3.8)), and with (3.10),

$$q_i w_i v \text{ guarantees } \{t_i x_i \mid t_i \xrightarrow[0]{x_i} D_i \subseteq U_i \subseteq S\}$$

$$\text{with no more moves than } Cost^* (\{q_i\} \xrightarrow[m]{w_i} S_i \xrightarrow[m+1]{v} U_i) .$$

In every case $pu\gamma v$ guarantees $p_1 v_1 \gamma v$, that guarantees (or is already in)

$$\left\{ p' w' \mid p' \xrightarrow[0]{w'} S' \subseteq S \right\} .$$

By transitivity of “guarantees”, we have

$$\text{from } pu\gamma v \text{ Player 0 can reach } \left\{ p' w' \mid p' \xrightarrow[\mathcal{A}_0]{w'} S' \subseteq S \right\} ,$$

with no more than (“either”)

$$Cost^* (\{p\} \xrightarrow[m]{u} T_1)$$

respectively (“or”, α)

$$Cost^* (\{p\} \xrightarrow[m]{u} T_1) + Cost^* (\{t_1\} \xrightarrow[m]{\gamma} T_2'' \xrightarrow[m+1]{v} T_3')$$

respectively (“or”, β , case 0), see (3.6)

$$Cost^* (\{p\} \xrightarrow[m]{u} T_1) + 1 + Cost^* (\{q_0\} \xrightarrow[m]{w_0} S_0 \xrightarrow[m+1]{v} U_0)$$

$$= Cost^* (\{p\} \xrightarrow[m]{u} T_1) + Cost(t) + Cost^* (S_0 \xrightarrow[m+1]{v} U_0)$$

respectively (“or”, β , case 1), see (3.9)

$$Cost^* (\{p\} \xrightarrow[m]{u} T_1) + 1 + Cost^* (\{q_i\} \xrightarrow[m]{w_i} S_i \xrightarrow[m+1]{v} U_i)$$

$$= Cost^* (\{p\} \xrightarrow[m]{u} T_1) + Cost(t) + Cost^* (S_i \xrightarrow[m+1]{v} U_i)$$

moves.

That is in every case no more than $Cost^*(\{p\} \xrightarrow[m+1]{w} S)$ moves, by the definition of the cost and by (3.2).

The property is proved for \mathcal{A}_{m+1} . From the induction we can conclude that it is proved for each $m \geq 0$.

(One can simplify a little bit the end of the proof in the case where \mathcal{A}_0 is a finite automaton.)

3.2.4 Pushdown Strategy

A *pushdown strategy* for Player 0, as defined in [Wal96b], is a deterministic pushdown automaton with input and output. It “reads” the moves of Player 1 (elements of Δ) and outputs the moves (choices) of Player 0, like a pushdown transducer. For simplicity, we will restrict our presentation to the following form of pushdown strategy:

Definition 3.2.6 *Given a PDGS $(P_0, P_1, \Gamma, \Delta)$, $P = P_0 \uplus P_1$, where Δ_i is the set of transition rules in Δ departing from Player i configurations, a pushdown strategy for Player 0 in this game is a deterministic pushdown automaton $\mathcal{S} = (P, A, \Pi)$, where $A = \Gamma \times \Sigma$, Σ is any alphabet, $\Pi \subseteq ((P_1 \times A \times \Delta_1) \times (P \times A^*)) \cup ((P_0 \times A) \times (P \times A^* \times \Delta_0))$ is a finite set of transition rules.*

A transition of \mathcal{S} either reads a move of Player 1 or outputs a move for Player 0, in both cases updating its stack. We will now define a pushdown strategy, starting from the automaton $\mathcal{A}_{Att(R)}$. Given a configuration $pw \in Attr_0(R)$, there is an accepting run $\{p\} \xrightarrow{w}^* S$ of $\mathcal{A}_{Att(R)}$:

$$Q_0 = \{p\} \xrightarrow{\gamma_0/\sigma_0} Q_1 \xrightarrow{\gamma_1/\sigma_1} \dots Q_n \xrightarrow{\gamma_n/\sigma_n} S, \quad w = \gamma_0\gamma_1 \dots \gamma_n.$$

Our aim is to store in the stack of the strategy the description of this run. The corresponding configuration of \mathcal{S} is $p(\gamma_0, \sigma_0) \dots (\gamma_n, \sigma_n)$. We fix for the alphabet Σ the set Φ (see Section 3.0.4).

At the beginning of the play, if the initial configuration pw is in $Attr_0(R)$, we have to initialize the stack of \mathcal{S} with the description of an accepting run of $\mathcal{A}_{Att(R)}$ (not necessarily the cheapest according to the costs defined above). Algorithm 3.1.5 can initialize the stack at the same time when searching an accepting run (in linear time). We define now the unique transition rule of Π from $(p, (\gamma_0, \sigma_0))$ or $(p, (\gamma_0, \sigma_0), \delta_1)$ (with $\delta_1 \in \Delta_1$). By construction $\sigma_0(p)$ is the “good” transition $\tau = p \xrightarrow{\gamma_0} Q_1$ used in the run of $\mathcal{A}_{Att(R)}$.

- If $p \in P_0$ then output the move $Rule(\sigma_0(p)) = p\gamma_0 \hookrightarrow qv$ that corresponds to τ . Remove the first letter of the stack. Push on the stack the description of the run $\{q\} \xrightarrow{v}^* Q_1$ used in Algorithm 3.1.2 to generate τ . Go to control state q .
- If $p \in P_1$ and Player 1 chooses the transition $\delta_1 = p\gamma_0 \hookrightarrow qv$ in Δ_1 , by construction of the automaton $\mathcal{A}_{Att(R)}$, $q \xrightarrow{v}^* S$ and S is a subset Q_1 . Go to control state q , remove the first letter of the stack, push the description of the run $\{q\} \xrightarrow{v}^* S$ used in Algorithm 3.1.2) to generate τ .

For Example 3.2.1 (Figure 3.1), we can see that the pushdown strategy is winning even if the initialization is not optimal. The configuration paa is in the winning region and an accepting run is coded on the stack of the strategy:

$$p(a, \{(p, 1)\})(a, \{(p, 3)\}) .$$

According to the strategy, the following play is generated (the symbol “ $-$ ” denotes a value that is not relevant):

$$\begin{array}{lll} \text{(by Rule((1)) = } pa \hookrightarrow p) & \text{proceed to} & p(a, \{(p, 3)\}) \\ \text{(by Rule((3)) = } pa \hookrightarrow paa) & \text{proceed to} & p(a, \{(p, 2)\})(a, -) \\ \text{(by Rule((2)) = } pa \hookrightarrow paa) & \text{proceed to} & p(a, -)(a, -)(a, -) . \end{array}$$

Theorem 3.2.7 *Given a PDGS with a reachability condition, a regular goal set R , and an alternating automaton $\mathcal{A}_{Attr_0(R)}$ for $Attr_0(R)$ (as computed from Algorithm 3.1.2), one can construct effectively a pushdown strategy that is winning from each node of the winning region W_0 of Player 0. Its transition function is defined uniformly for the whole winning region. The initialization of the stack is possible in linear time in the length of the initial game position, and the computation of the “next move” is in constant time (for fixed Δ).*

Although there is no need to compute costs to define this strategy, it is useful to refer to the costs of the previous subsection for the correctness proof. The strategy for Player 1 in this “safety game” is much easier to define and compute: he just has to stay in $V \setminus Attr_0(R)$.

Proof: We consider a play according to the strategy: a path in the transition graph of the strategy. This is a sequence $\pi_0 \cdots \pi_n$ of configurations of the strategy such that the stack of π_0 describes an accepting run of $\mathcal{A}_{Attr_0(R)}$. By definition of Algorithm 3.1.2 it is easy to show by induction on n that:

- the stack of π_n describes also an accepting run of $\mathcal{A}_{Attr_0(R)}$,
- the cost of this run is strictly smaller than the cost of the run described in π_{n-1} (each move of the strategy replaces the first segment of the run by a “cheaper” run).

■

3.2.5 Discussion

The notion of “bounded model-checking” has been considered by several authors, see e.g. [KV01]. Here the aim is to reach a goal in less than j steps (where for example $j = 100$). In our game framework it is possible, using the computation of costs, to determine whether Player 0 can reach the goal in less than j moves. Note that this requires in our setting much more computational effort than just to check whether Player 0 can win. More specifically one can construct a finite automaton recognizing $Attr_0^j(R)$ by unfolding partly the automaton recognizing $Attr_0(R)$ and summing up the costs.

The stack of the pushdown strategy needs to be initialized at the beginning of a play (in linear time in the length of the configuration); then the computation of the “next move” is done in constant time (execution of one transition of the strategy). In contrast, the min-rank strategy needs for each move a computation in linear time in the length of the configuration. So we can say that in the case of pushdown graphs a positional strategy can be more expensive than a strategy with memory. This effect does not appear over finite-state game graphs, where a positional strategy is not an algorithm, but just a finite set.

Moreover when we want to compare two algorithms “solving” a game, we see that different criteria of complexity/efficiency arise. We can look at time or space complexity of the algorithms for:

- determining if a given vertex is in the winning region of Player 0,
- computing a uniform (finite) description of the whole winning region of Player 0,
- using this description (apply to a given vertex),
- computation of a finite description of a winning strategy,
- usage of this strategy: initialization, complexity of computing the “next move”, information needed.

3.3 Back to the Basic Example of Chapter 1

The basic example from the introduction (Chapter 1), the game between Alain and Brigitte, can be modeled as a pushdown game and now be solved. The pushdown store will represent the number of tokens that are on the table and the control location will represent the state of the players. Only two symbols are needed in

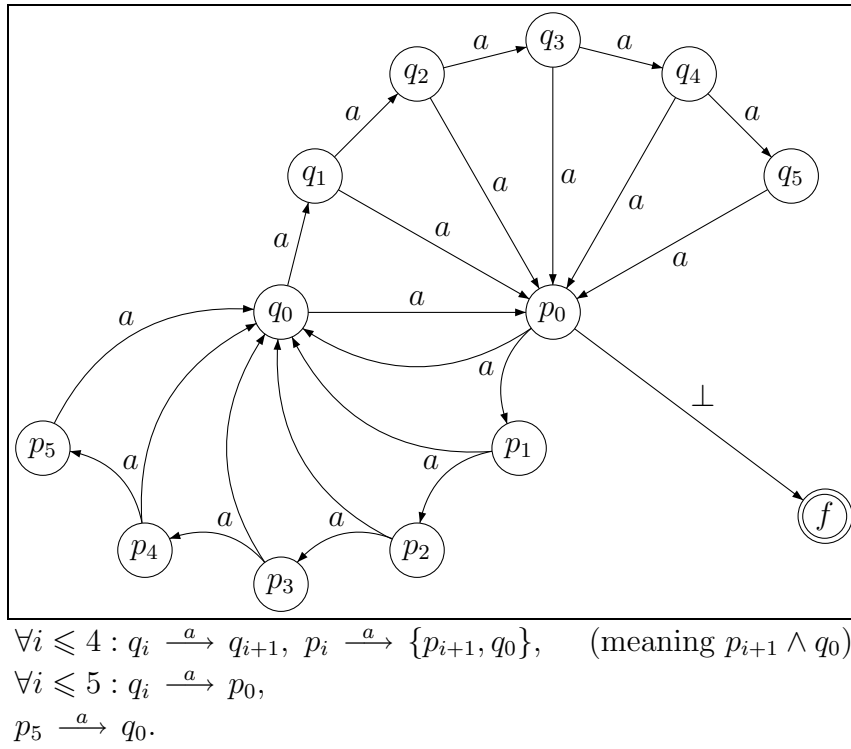


Figure 3.2: Alternating automaton recognizing $Attr_0(R)$ for Example 3.3.1

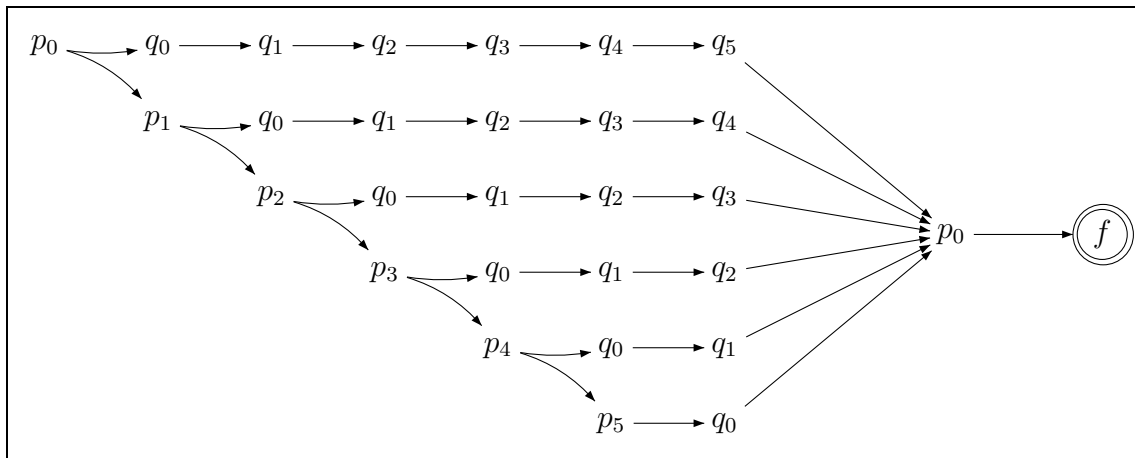


Figure 3.3: The run DAG of this automaton on the word $a^7 \perp$. The transition labels are omitted

the stack alphabet: a letter a which represents one token and a “bottom stack symbol” \perp to know when the stack is empty (looking carefully at the definitions and conventions, the bottom stack symbol can be avoided here). The starting position is $p_0a^{42}\perp$. By definition of the PDS, stack symbols can be removed only one by one, so we need intermediate states to check that Alain removes at most 6 tokens. In state p_0 with top letter a , Alain can remove just one token (the top symbol) and give control to Brigitte (move $p_0a \hookrightarrow q_0$) or remove one token and go to p_1 where he still has the control and can remove more tokens (move $p_0a \hookrightarrow p_1$). Similarly from p_1 he can remove one token and go to q_0 (Brigitte) or p_2 . And so on until p_5 , where he has no choice. For Brigitte states q_0 to q_5 do the same thing. We want to determine the winning region of Brigitte. The goal set of Brigitte is $\{p_0\perp\}$: she has to remove the last token and, by convention, give control to Alain who is stuck. To conform to previous sections, B should be Player 0.

Example 3.3.1 *Formally we have:*

$$\begin{aligned} \Gamma &= \{a, \perp\}, P_A = \{p_0, \dots, p_5\}, P_B = \{q_0, \dots, q_5\}, R = \{p_0\perp\}, \\ \Delta &= \{p_i a \hookrightarrow q_0, \quad q_i a \hookrightarrow p_0 \quad : \forall i \leq 5\} \cup \\ &\quad \{p_i a \hookrightarrow p_{i+1}, \quad q_i a \hookrightarrow q_{i+1} \quad : \forall i \leq 4\} \end{aligned}$$

The result of Algorithm 3.1.2 is depicted in Figure 3.2. The run DAG of Figure 3.3 can be extended to any number of tokens that is a multiple of 7. The strategy (either min-rank or pushdown) is easy to compute because there is at most one accepting run.

Looking at the variant we have mentioned in the introduction, where the winning condition is now:

The player who takes the last token wins if and only if
he removes in total an even number of tokens,

it is also possible to model this game as a pushdown game. One can code in the control states whether Player 0 has removed an even number of tokens. More formally a finite automaton with two state can determine if a word has even length and one can construct the product of the above pushdown system by this finite automaton.

3.4 Büchi Condition

Given \mathcal{P} and R as in the preceding sections, the (Büchi) winning condition is the following:

Player 0 wins a play if it meets infinitely often the goal set R , or ends in a deadlock for Player 1.

To determine the winning region of this game one defines $Attr_0^+$, a variant of $Attr_0$. Given $T \subseteq V$, let $Attr_0^+(T)$ be the least fixed point of the function

$$U \mapsto \chi(T \cup U) .$$

Alternatively, one can define $Attr_0^+$ by induction: let

$$\begin{aligned} X_0(T) &= \emptyset , \\ X_{i+1}(T) &= X_i(T) \cup \{u \in V_0 \mid \exists v, u \leftrightarrow v, v \in T \cup X_i(T)\} \\ &\quad \cup \{u \in V_1 \mid \forall v, u \leftrightarrow v \Rightarrow v \in T \cup X_i(T)\} , \\ Attr_0^+(T) &= \bigcup_{i \geq 0} X_i(T) \quad (\text{the degree of the game graph is bounded}). \end{aligned}$$

The above definition follows from [Tho95] (where the definition of the X_i 's is adjusted). The following claims are easy:

Proposition 3.4.1 *$Attr_0^+(T)$ is the set of nodes from which Player 0 can join T in at least one move, whatever Player 1 does.*

Proposition 3.4.2 *The winning region of the Büchi game with goal set R , denoted $Büchi_0(R)$, is the greatest fixed point of the function*

$$S \mapsto Attr_0^+(S \cap R) .$$

This fixed point is also obtained by induction: let

$$\begin{aligned} Büchi_0^0(R) &= V , \\ Büchi_0^{\alpha+1}(R) &= Attr_0^+(Büchi_0^\alpha(R) \cap R) \quad \text{for any ordinal } \alpha , \\ Büchi_0^\lambda(R) &= \bigcap_{\alpha < \lambda} Büchi_0^\alpha(R) \quad \text{for a limit ordinal } \lambda . \end{aligned}$$

Then there is an ordinal α such that $Büchi_0^{\alpha+1}(R) = Büchi_0^\alpha(R)$ [GTW02, Ch. 20]. And for any such α we have $Büchi_0^\alpha(R) = Büchi_0(R)$. Here it is possible that the least such α is greater than ω , even in the case of pushdown graphs (with a finite degree), see example in Section 3.4.4. When R is clear from the context we will write simply $Büchi_0^i$ and $Büchi_0$. In the case of finite graphs, this induction can be effectively carried out: the sequence $(Büchi_0^i)_{i > 0}$ is strictly decreasing until it reaches

a fixed point. For pushdown graphs, the regular languages Büchi₀ⁱ are also smaller and smaller, but the question of convergence is nontrivial. Following the symbolic approach, we will construct finite automata that are strictly “decreasing” until they reach a fixed point in a finite number of steps.

Again from this definition Player 0 has a positional winning strategy on his winning region, associated to the attractor. But here the winning strategy of Player 1 is not so easy to describe, see Section 3.5. This is a hint that parity games — even with two colors — are not so easy to understand.

From now on we will proceed in two steps: firstly Algorithm 3.4.4 computes an automaton that recognizes Büchi₀ⁱ for a given (integer) i , secondly Algorithm 3.4.8 computes directly an automaton for Büchi₀. The first algorithm helps to understand the second, but is not a pre-computation.

In this section we will consider a simple goal set of the form $R = F\Gamma^*$ for some $F \subseteq P$. So the membership in R only depends on the control state of a configuration. This is not an essential restriction as shown by the next proposition.

Proposition 3.4.3 *Given a PDGS $\mathcal{P} = (P_0, P_1, \Gamma, \Delta)$ that defines a game and a regular set R of configurations, we can reduce to the case of simple goal sets by proceeding to a new PDGS $\mathcal{P} \times \mathcal{D}$, where \mathcal{D} is a finite automaton recognizing R .*

Proof: There is a deterministic (complete) finite automaton $\mathcal{D} = (Q, \Gamma, \delta, q_0, E)$ that recognizes R backward: reading words from right to left. We can simulate \mathcal{D} in a new PDGS $\mathcal{P} \times \mathcal{D} = ((P_0 \times Q, P_1 \times Q, \Gamma \times Q, \Delta')$ where a configuration

$$p\gamma_0 \cdots \gamma_n$$

of \mathcal{P} is associated, via the unique run

$$q_{n+2} \xleftarrow{p} q_{n+1} \xleftarrow{\gamma_0} q_n \cdots \xleftarrow{\gamma_n} q_0$$

of \mathcal{D} to the configuration of $\mathcal{P} \times \mathcal{D}$

$$(p, q_{n+1})(\gamma_0, q_n) \cdots (\gamma_n, q_0) .$$

Then we know that $p\gamma_0 \cdots \gamma_n \in R \Leftrightarrow \delta(q_{n+1}, p) \in E$. Let $F = \{(p, q) \in P \times Q \mid \delta(q, p) \in E\}$ and define the transitions of $\mathcal{P} \times \mathcal{D}$ to be

$$\begin{aligned} \langle (p, q)(\gamma, q_1) \hookrightarrow (p', q')(\gamma_n, q_n) \cdots (\gamma_1, q_1) \rangle \in \Delta' \Leftrightarrow \\ (p\gamma \hookrightarrow p'\gamma_n \cdots \gamma_1) \in \Delta, q_{i+1} = \delta(q_i, \gamma_i), q' = \delta(q_n, \gamma_n) . \end{aligned}$$

(if $n = 0$, $q' = q_1$) If the initialization is done correctly, the stack will contain a correct run at every step, and \mathcal{P} will have the same behavior as $\mathcal{P} \times \mathcal{D}$ with the property that R corresponds to $R' = F\Gamma^*$. ■

This reduction is a very simple example of game reduction in the sense of Chapter 4. The proof shows that, on a theoretical point of view, for the model-checking over pushdown systems it is not difficult to consider “regular state properties”, as remarked by several authors [FWW97, KV00, EKS01]. These extra properties express that a configuration is in a given regular set of configurations. So from now on we consider a simple goal set $R = F\Gamma^*$ for some $F \subseteq P$. Of course it is regular.

3.4.1 Computation of Büchi₀^j for all $j > 0$

Algorithm 3.1.2 is modified (by new steps involving ε -transitions) to compute $Attr_0^+(R)$. The intersection with R is easy to compute, and Algorithm 3.4.4 determines successively Büchi₀¹, Büchi₀², Büchi₀³ . . .

Algorithm 3.4.4 (computation of Büchi₀^j)

Input: PDGS $\mathcal{P} = (P_0, P_1, \Gamma, \Delta)$, $j \geq 1$ and $F \subseteq P$ that defines the goal set $R = F\Gamma^*$.

Output: a \mathcal{P} -automaton \mathcal{B}_j that recognizes Büchi₀^j.

Initialization: the state space of \mathcal{B}_j is $\{f\} \cup (P \times [1, j])$, and f is the unique final state (we write (p, i) as p^i). For all $\gamma \in \Gamma$, $f \xrightarrow{\gamma} f$. For all $p \in P$, p^0 is set to be f .

for $i := 1$ to j **do**

(compute generation i , consider now the p^i 's as initial states.)

Add an ε -transition from p^i to p^{i-1} for each $p \in F$ (only). (If $i = 1$, p^0 is f .)

Add new transitions to \mathcal{B}_j according to Algorithm 3.1.2:

repeat

(Player 0) if $p \in P_0$, $p\gamma \hookrightarrow qw$ and $q^i \xrightarrow{w} S$ in the current automaton, then add a new transition $p^i \xrightarrow{\gamma} S$.

(Player 1) if $p \in P_1$, $\{p\gamma \hookrightarrow q_1w_1, \dots, p\gamma \hookrightarrow q_nw_n\}$ are all the moves (rules) starting from $p\gamma$ and $\forall k$, $q_k^i \xrightarrow{w_k} S_k$ in the current automaton, then add a new transition $p^i \xrightarrow{\gamma} \bigcup_k S_k$.

until no new transition can be added

remove the ε -transitions. (generation number i is done)

end for

The notion of “generation” introduced in the algorithm is illustrated in the following example.

Example 3.4.5 We consider $\Delta = \{pa \leftrightarrow p\}$, $R = p\Gamma^*$ ($F = \{p\}$), $P = P_0 = \{p, q\}$, $P_1 = \emptyset$. Figure 3.4 shows the result of the algorithm where $j = 4$. The i -th generation is given by the states of column i . The a -edges from p^i are added in the construction of generation i , since $pa \leftrightarrow p$ and $p^i \xrightarrow{\varepsilon_*} p^i$, $p^i \xrightarrow{\varepsilon_*} p^{i-1}$. The dashed ε -transitions are removed after each generation.

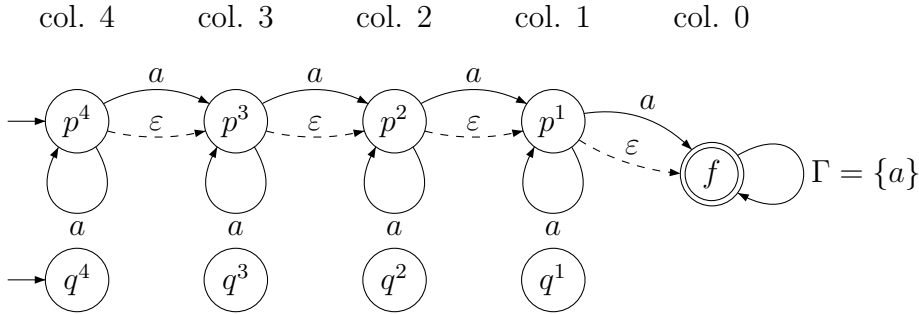


Figure 3.4: Automaton from Algorithm 3.4.4 and Example 3.4.5

Proposition 3.4.6 Algorithm 3.4.4 constructs a \mathcal{P} -automaton \mathcal{B}_j that recognizes exactly Büchi_0^j (using the states p^j as initial states).

Proof: By induction on j :

We note \mathcal{B}_j the automaton obtained after the j -th generation, with initial states p^j , $p \in P$.

- Using the convention that p^0 is f for all $p \in P$, the automaton \mathcal{B}_0 recognizes $P\Gamma^* = V = \text{Büchi}_0^0$

- Induction hypothesis: \mathcal{B}_j recognizes Büchi_0^j

- Then $\text{Büchi}_0^j \cap R$ is recognized on \mathcal{B}_j from the states $p^j, p \in F$, because of the simple structure of R (i.e., $pw \in \text{Büchi}_0^j \cap R$ if pw is accepted by \mathcal{B}_j and $p \in F$).

By the help of the ε -transitions, the automaton \mathcal{B}_{j+1} recognizes exactly $\text{Büchi}_0^j \cap R$ before the saturation procedure starts. The construction of \mathcal{B}_{j+1} from \mathcal{B}_j was proved in Section 3.1 to compute the attractor: \mathcal{B}_{j+1} recognizes then $\text{Attr}_0(\text{Büchi}_0^j \cap R)$. After we remove the ε -transitions, it defines $\text{Attr}_0^+(\text{Büchi}_0^j \cap R)$: the configurations of $\text{Attr}_0(\text{Büchi}_0^j \cap R)$ such that after one move we are in $\text{Attr}_0(\text{Büchi}_0^j \cap R)$.

In other words if a configuration pw can be accepted by \mathcal{B}_{j+1} *only* by using an ε -transition, then it is in $\text{Büchi}_0^j \cap R$, but not in $\text{Attr}_0^+(\text{Büchi}_0^j \cap R)$. ■

To compare different generations one defines a function ϕ which is the translation one column to the right of the elements of S in the figure above, with the convention that f remains unchanged. For all finite sets $S \subseteq P \times \mathbb{N}$ let

$$\phi(S) = \begin{cases} \{q^i \mid q^{i+1} \in S\} \cup \{f\} & \text{if } f \in S \text{ or } \exists q^1 \in S \\ \{q^i \mid q^{i+1} \in S\} & \text{else} \end{cases}$$

The next proposition states that there are less and less transitions in the higher generations (we already know that $\text{Büchi}_0^{i+1} \subseteq \text{Büchi}_0^i$ by monotonicity).

Proposition 3.4.7 *In Algorithm 3.4.4, for all $u \in \Gamma^*$, $p \in P$, $i \geq 1$,*

$$p^{i+1} \xrightarrow{u_*} S \Rightarrow p^i \xrightarrow{u_*} \phi(S) .$$

Proof: We proceed by induction on i .

It is a direct consequence of the same property over the transitions (by induction on the length of the word).

- For $i = 1$, we use another induction on the number of transitions starting from p^2 added by the saturation procedure.

- At the beginning of the second iteration, one has no other transitions from the q^2 's than the $q^2 \xrightarrow{\varepsilon} \{q^2\}$, and for all $q \in F$, $q^2 \xrightarrow{\varepsilon} \{q^1\}$. And from the q^1 's there may be paths $q^1 \xrightarrow{w_*} S$ such that $S \subseteq P \times \{1\} \cup \{f\}$, hence $\phi(S) = \{f\}$.

Respectively, at the beginning of the first iteration, before the saturation procedure starts, one had $q^1 \xrightarrow{\varepsilon} \{q^1\}$, if $q \in F$, then $q^1 \xrightarrow{\varepsilon} \{f\}$, and $\forall \gamma, f \xrightarrow{\gamma} \{f\}$.

- as a new transition $p^2 \xrightarrow{\gamma} S$ is added, it is through an existing path $q^2 \xrightarrow{w_*} S$, by induction hypothesis one has also $q^1 \xrightarrow{w_*} \phi(S)$ during the saturation procedure of the iteration one, so the transition $p^1 \xrightarrow{\gamma} \phi(S)$ was already added.

- induction hypothesis: $\forall S, p^i \xrightarrow{a} S \Rightarrow p^{i-1} \xrightarrow{a} \phi(S)$
- the proof for $i + 1$ is similar to the case $i = 1$

■

Nevertheless the sequence (Büchi_0^i) might be strictly decreasing, so that the previous

algorithm can not reach a fixed point: for the example above one gets $\text{Büchi}_0^i = pa^i\Gamma^*$, $\forall i > 0$. But on the symbolic level we can modify Algorithm 3.4.4 to obtain an automaton for Büchi_0 directly.

3.4.2 Computation of Büchi_0 Using an Acceleration

We use the projection $\pi^i(S)$ of the states in $P \times [1, i]$ to the i -th column (except for f). For all $i > 0$ and sets $S \subseteq P \times [1, i] \cup \{f\}$, let

$$\pi^i(S) = \{q^i \mid \exists i \geq k > 0, q^k \in S\} \cup \{f \mid f \in S\} .$$

This projection is used in the next algorithm to replace some transitions and will allow to have faster convergence to the winning set. It can be compared to known methods of “acceleration”.

Algorithm 3.4.8 (computation of Büchi_0)

Input: PDGS \mathcal{P} , $F \subseteq P$ that defines the goal set $R = F\Gamma^*$

Output: a \mathcal{P} -automaton \mathcal{C} that recognizes Büchi_0

Initialization: the state space of \mathcal{C} is a subset of $(P \times \mathbb{N}) \cup \{f\}$, where (p, i) is denoted by p^i and f is the unique final state. For all $\gamma \in \Gamma$, $f \xrightarrow{\gamma} f$ in \mathcal{C} . By convention p^0 is f for all $p \in F$.

$i := 0$.

repeat

$i := i + 1$ (consider now the p^i 's as initial states.)

Add an ε -transition from p^i to p^{i-1} for each $p \in F$ (only)

Add new transitions to \mathcal{C} according to Algorithm 3.1.2 (see inner loop of Algorithm 3.4.4). Remove the ε -transitions.

Replace each transition $p^i \xrightarrow{\gamma} S$ by $p^i \xrightarrow{\gamma} \pi^i(S)$

until $i > 1$ and the outgoing transitions from the p^i 's are “the same” as from the p^{i-1} 's:

$$p^i \xrightarrow{\gamma} S \Leftrightarrow p^{i-1} \xrightarrow{\gamma} \phi(S) .$$

Applying the algorithm to the simple example above, we see that the ε -transitions and the a -transitions from p^2 to p^1 and from p^3 to p^2 are deleted (by the projection), and that only three generations are created. See Figure 3.5.

Theorem 3.4.9 *The automaton constructed by Algorithm 3.4.8 recognizes $\text{Büchi}_0(R)$, the winning region of the Büchi game given by the PDGS \mathcal{P} and regular goal set R .*

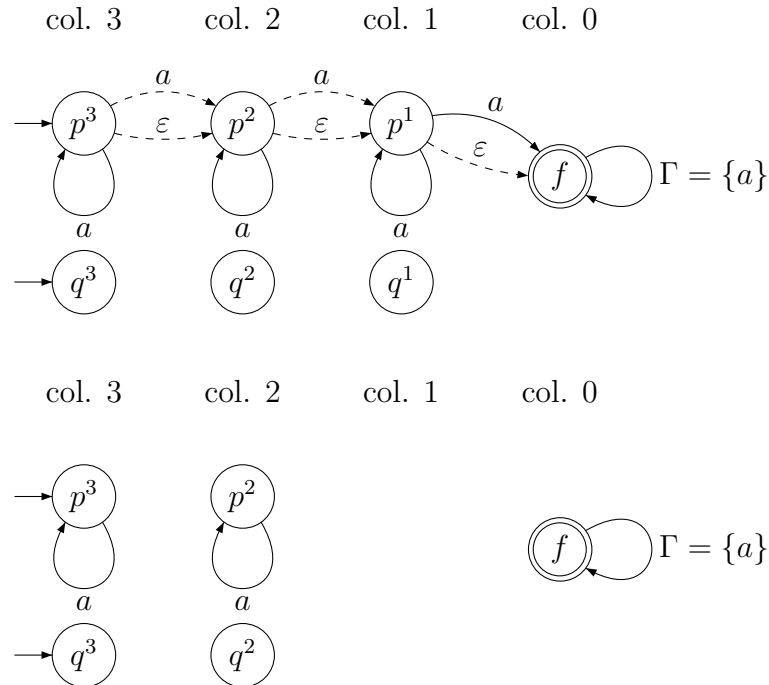


Figure 3.5: Automaton from Algorithm 3.4.8 and Example 3.4.5, the dashed arrows are removed

Before giving the proof we make some remarks and state the next proposition. The theorem implies that $\text{Büchi}_0(R)$ is regular for regular R . Similarly to Section 3.1 each execution of the inner loop (saturation procedure) is done in time $|\Delta| 2^{\mathcal{O}(|Q|^2)}$, where $|Q| = 2|P| + 1$. At each step of the outer loop, we “lose” at least one transition. Since there are at most $|\Gamma| |P| 2^{|Q|}$ of them, the global time complexity of the algorithm is $\mathcal{O}(|\Gamma| |\Delta| 2^{c|P|^2})$.

Note that in Algorithm 3.4.8 we can erase the p^{i-1} 's and their transitions as soon as the generation i is done. Similarly to the first algorithm, we have the following property.

Proposition 3.4.10 *In Algorithm 3.4.8, for all $u \in \Gamma^*$, $p \in P$, $i \geq 1$,*

$$p^{i+1} \xrightarrow{u} S \Rightarrow p^i \xrightarrow{u} \phi(S) .$$

Remark that because of the projection π , the transitions $p^i \xrightarrow{u} S$ verify $S \subseteq \{f\} \cup P \times \{i\}$.

Proof: Proposition 3.4.10

The proof is similar to that of the first algorithm. We proceed by induction on i . It is a direct consequence of the same property over the transitions (by induction on the length of the word).

- For $i = 1$, we use another induction on the number of transitions starting from p^2 added by the saturation procedure. Which means in this case, if a transition $p^2 \xrightarrow{\gamma} S$ is added, then its projection verifies the property, that is

$$p^1 \xrightarrow{\gamma} \phi(\pi^2(S)) ,$$

because at the end of generation 2, we will have $p^2 \xrightarrow{\gamma} \pi^2(S)$.

- At the beginning of the second iteration, one has no other transitions from the q^2 's than the $q^2 \xrightarrow{\varepsilon} \{q^2\}$, and for all $q \in F$, $q^2 \xrightarrow{\varepsilon} \{q^1\}$. We have to check that $q^1 \xrightarrow{\varepsilon} \phi(\pi^2(q^1)) = \phi(q^2) = q^1$, which is clear.

And from the q^1 's there may be paths $q^1 \xrightarrow{w_*} S$ such that $S \subseteq P \times \{1\} \cup \{f\}$.

Hence from q^2 (only if $q \in F$), we have

$$q^2 \xrightarrow{\varepsilon} q^1 \xrightarrow{w_*} S \subseteq P \times \{1\} \cup \{f\} \text{ with}$$

$$\pi^2(S) \subseteq P \times \{2\} \cup \{f\}, \phi(\pi^2(S)) \subseteq P \times \{1\} \cup \{f\}, \text{ and } \phi(\pi^2(S)) = S.$$

So we have $q^1 \xrightarrow{w_*} \phi(\pi^2(S))$.

- *During* the saturation procedure, as a new transition $p^2 \xrightarrow{\gamma} S$ is added, it is through an existing path $q^2 \xrightarrow{w_*} S$, by induction hypothesis one has also $q^2 \xrightarrow{w_*} \pi^2(S) \Rightarrow q^1 \xrightarrow{w_*} \phi(\pi^2(S))$ so the transition $p^1 \xrightarrow{\gamma} \phi(\pi^2(S))$ exists since the projection of the first generation.

- Induction hypothesis: $\forall S, p^i \xrightarrow{a} S \Rightarrow p^{i-1} \xrightarrow{a} \phi(S)$.
- The proof for $i + 1$ is here exactly the same as the case $i = 1$.

■

Proof: Theorem 3.4.9

Proof of termination

Thanks to the projections π^i 's, the number of possible transitions from each row of p^i 's is bounded. And thanks to Proposition 3.4.10, there are less and less transitions until the algorithm reaches a fixed point.

Proof of correctness

We note Z_i the language recognized by \mathcal{C} from the initial states p^i 's. We denote $n + 1$ the last generation of the algorithm, which is such that $Z_n = Z_{n+1}$, but we

still consider $Z_i = Z_n$ for all $i \geq n$. One has to show that $Z_n = \text{Büchi}_0$.

First part: $Z_n \subseteq \text{Büchi}_0$.

We first prove by induction on the integers that for all i , $Z_i \subseteq \text{Büchi}_0^i$.

- By construction, $Z_1 = \text{Büchi}_0^1$.
- Induction hypothesis: $Z_i \subseteq \text{Büchi}_0^i$.
- The algorithm first determines the attractor⁺ of the language. By monotonicity,

$$\tilde{Z}_{i+1} = \text{Attr}_0^+(Z_i \cap R) \subseteq \text{Attr}_0^+(\text{Büchi}_0^i \cap R) = \text{Büchi}_0^{i+1} .$$

After the projection of the transitions, the obtained language Z_{i+1} is contained in \tilde{Z}_{i+1} : Proposition 3.4.10 shows that an accepting path from a state p_{i+1} was possible before the projection (through the q^i 's). So $Z_{i+1} \subseteq \tilde{Z}_{i+1} \subseteq \text{Büchi}_0^{i+1}$.

This proves that $\forall i, Z_n = Z_{n+1} \subseteq \text{Büchi}_0^i$, and so $Z_n \subseteq \text{Büchi}_0^\omega$. Now by transfinite induction we prove that for all ordinal $\alpha \geq \omega$, $Z_n \subseteq \text{Büchi}_0^\alpha$.

- We have $Z_n \subseteq \text{Büchi}_0^\omega$.
- Induction hypothesis: $Z_n \subseteq \text{Büchi}_0^\alpha$.
- For a successor ordinal,

$$Z_n = Z_{n+1} \subseteq \tilde{Z}_{n+1} = \text{Attr}_0^+(Z_n \cap R) \subseteq \text{Attr}_0^+(\text{Büchi}_0^\alpha \cap R) = \text{Büchi}_0^{\alpha+1} .$$

- For a limit ordinal λ , we have $\forall \alpha < \lambda : Z_n \subseteq \text{Büchi}_0^\alpha$ and so

$$Z_n \subseteq \text{Büchi}_0^\lambda = \bigcap_{\alpha < \lambda} \text{Büchi}_0^\alpha .$$

We conclude that $Z_n \subseteq \text{Büchi}_0$.

Second part: $\text{Büchi}_0 \subseteq Z_n$.

We prove by induction on i that for all i , $\text{Büchi}_0 \subseteq Z_i$.

- By construction, $\text{Büchi}_0 \subseteq \text{Büchi}_0^1 = Z_1$.
- Induction hypothesis: $\text{Büchi}_0 \subseteq Z_i$.

- Before the projection we have

$$\text{Büchi}_0 = \text{Attr}_0^+(\text{Büchi}_0 \cap R) \subseteq \text{Attr}_0^+(Z_i \cap R) = \tilde{Z}_{i+1} .$$

We proceed by induction on the number of transitions that are changed by the projection, *i.e.*, we consider the successive automata $\mathcal{A}_0, \dots, \mathcal{A}_m$, where $L(\mathcal{A}_0) = \tilde{Z}_{i+1}$, $L(\mathcal{A}_m) = Z_{i+1}$, and \mathcal{A}_{j+1} is obtained from \mathcal{A}_j by “projecting” one transition. We have to prove by induction on m that $\text{Büchi}_0 \subseteq L(\mathcal{A}_m)$.

- If $m = 0$, $\text{Büchi}_0 \subseteq L(\mathcal{A}_0) = \tilde{Z}_{i+1} = Z_{i+1}$.

- Induction hypothesis: $\text{Büchi}_0 \subseteq L(\mathcal{A}_m)$.

- We suppose by absurd that there is a configuration $pw \in L(\mathcal{A}_m) \setminus L(\mathcal{A}_{m+1})$ such that $pw \in \text{Büchi}_0$. We choose such a word pw of minimal length $|pw|$. For each accepting path in \mathcal{A}_m for pw , there is a decomposition $p^{i+1} \xrightarrow{u}_m^* S \xrightarrow{v}_m^* \{f\}$ such that $w = uv$, with $q^i \in S$, and in \mathcal{A}_{m+1} : $\neg(q^{i+1} \xrightarrow{v}_{m+1}^* \{f\})$ (the transition that is projected was “leading” to q^i in \mathcal{A}_m and is now “leading” to q^{i+1}).

This means that $qv \in Z_i$ and $qv \notin L(\mathcal{A}_{m+1})$.

If $qv \notin L(\mathcal{A}_m)$, then $qv \notin \text{Büchi}_0$ (Induction hypothesis).

If $qv \in L(\mathcal{A}_m) \setminus L(\mathcal{A}_{m+1})$, then qv cannot be in Büchi_0 by hypothesis: $u \neq \varepsilon$ and $|qv| < |pw|$.

In both cases, pw should not be in Büchi_0 (see Section 3.1), hence the contradiction.

We conclude that $\text{Büchi}_0 \subseteq L(\mathcal{A}_{m+1})$.

■

With an extension of the arguments of the previous section we obtain corresponding winning strategies:

Theorem 3.4.11 *For a Büchi game given by a PDGS \mathcal{P} and goal set R , one can compute from the automaton constructed by Algorithm 3.4.8 a min-rank (positional) winning strategy and a pushdown winning strategy, uniformly for the winning region of Player 0.*

Proof: By definition of Büchi_0 and by Algorithm 3.4.8, $\text{Attr}_0^+(\text{Büchi}_0 \cap R) = \text{Büchi}_0$, and Player 0 can use the strategy associated to that attractor. ■

Similarly to Section 3.2.5 we can note that it requires more computational effort to determine whether Player 0 can visit R *at least* j times (for a large j) than to determine whether Player 0 can win the Büchi game. This is the difference

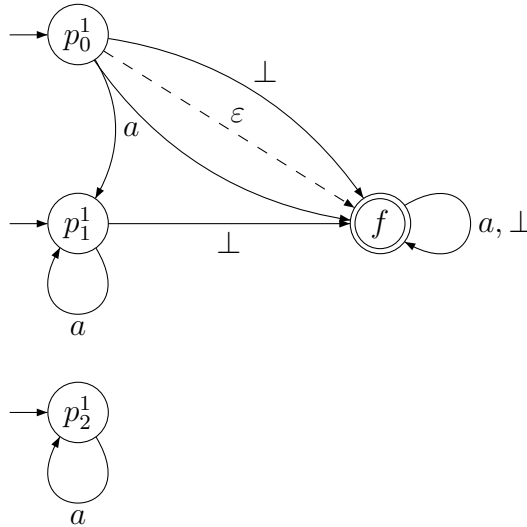


Figure 3.6: Automaton from Algorithm 3.4.8 and Section 3.4.3, first generation, the dashed arrow is removed.

between the computation of Büchi₀^j from Algorithm 3.4.4 and that of Büchi₀ from Algorithm 3.4.8.

In Algorithm 3.4.8 projection π is needed to have convergence but it seems difficult to give an intuition of the effect of the projection on the recognized set of configurations. Even in a set theoretical way we do not know how to express this effect without referring to the automaton. It is also difficult to describe the intermediate stages of computation: the set of configurations recognized at successive generation, whereas it was possible in the non-terminating version. Finally we neither found a better upper bound on the number of generations nor a lower bound.

3.4.3 Simple Example

The example of parity game of Section 4.3 is in fact a Büchi game because there is only two colors: Player 0 wins if and only if priority 0 is seen infinitely often. One can solve it here as a Büchi game. Let

$$\begin{aligned} \Gamma &= \{a, \perp\}, P_0 = \{p_1\}, P_1 = \{p_0, p_2\}, \\ \Delta &= \{p_0 \perp \hookrightarrow p_0 a \perp, p_0 a \hookrightarrow p_0 a a, p_0 a \hookrightarrow p_1 a, p_1 a \hookrightarrow p_1 \varepsilon, \\ &\quad p_1 \perp \hookrightarrow p_0 \perp, p_1 \perp \hookrightarrow p_2 \perp, p_2 \perp \hookrightarrow p_2 \perp, p_2 a \hookrightarrow p_2 \varepsilon\}, \\ F &= \{p_0\}, R = p_0 \Gamma^*. \end{aligned}$$

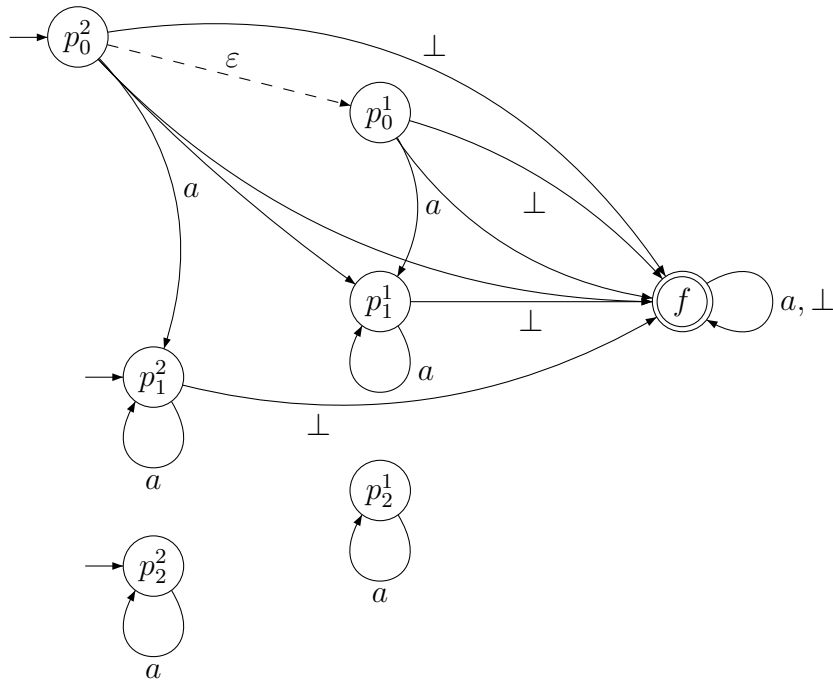


Figure 3.7: Automaton from Algorithm 3.4.8 and Section 3.4.3, second generation before the projection, the dashed arrow is removed.

The game graph is depicted in Figure 4.1. The only relevant choice of Player 0 is to go from $p_1 \perp$ to $p_0 \perp$ to ensure win. Applying Algorithm 3.4.8 the automata of Figures 3.6 and 3.7 are constructed. After the projection, the “branch” from p_0^2 to p_1^1 in Figure 3.7 disappears, and the resulting automaton is similar to that of Figure 3.6 (renaming p_0^1 in p_0^2, \dots). For that reason the algorithm terminates after two generations. The winning region of Player 0 is $\{p_0, p_1\}a^* \perp$ (restricting to configurations of the form $Qa^* \perp$).

For a given configuration there is at most one accepting run, so the winning strategy is uniquely defined.

3.4.4 Example of convergence to the fixed-point

The following example was suggested by Damian Niwiński. The aim is to show that in the inductive definition of $\text{Büchi}_0(R)$ an induction beyond ω is necessary. Let

$$\begin{aligned} \Gamma &= \{a\}, P_0 = \{p, q\}, P_1 = \emptyset, \\ \Delta &= \{qa \hookrightarrow qaa, qa \hookrightarrow p, pa \hookrightarrow p\}, \\ F &= \{p\}, R = p\Gamma^* = pa^*. \end{aligned}$$

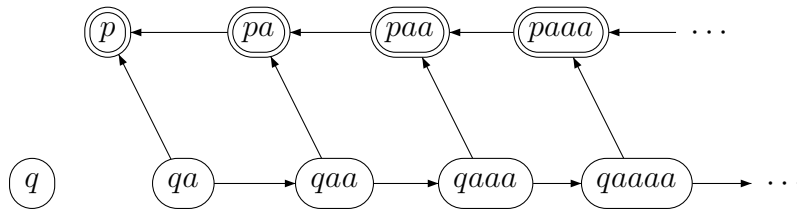


Figure 3.8: Example of a Büchi game (Section 3.4.4)

The game graph is shown in Figure 3.8. At every vertex Player 0 has to move. The vertex p is a deadlock, losing for Player 0. Player 0 can visit R as many times as he wants, but not infinitely often, so the winning region of Player 0 is empty.

Applying the inductive definitions of $Attr_0^+$ and $Büchi_0$, we have:

$$\begin{aligned}
 Büchi_0^0(R) &= V = pa^* \cup qa^* , \\
 Büchi_0^1(R) &= Attr_0^+(V \cap R) = Attr_0^+(pa^*) = pa^+ \cup qa^+ , \\
 Büchi_0^2(R) &= Attr_0^+((pa^+ \cup qa^+) \cap R) = Attr_0^+(pa^+) = paa^+ \cup qa^+ , \\
 &\dots \\
 Büchi_0^i(R) &= pa^i a^* \cup qa^+ \quad \text{for all } i \geq 0, \text{ and then} \\
 Büchi_0^\omega(R) &= \bigcap_{i < \omega} Büchi_0^i(R) = qa^+ ,
 \end{aligned}$$

which is not empty, and is not yet the fixed point:

$$\begin{aligned}
 Büchi_0^{\omega+1}(R) &= Attr_0^+(Büchi_0^\omega(R) \cap R) = Attr_0^+(\emptyset) = \emptyset , \\
 Büchi_0^\alpha(R) &= \emptyset \quad \text{for all } \alpha > \omega .
 \end{aligned}$$

This reflects the fact that from vertex pa^i Player 0 can visit R only i times.

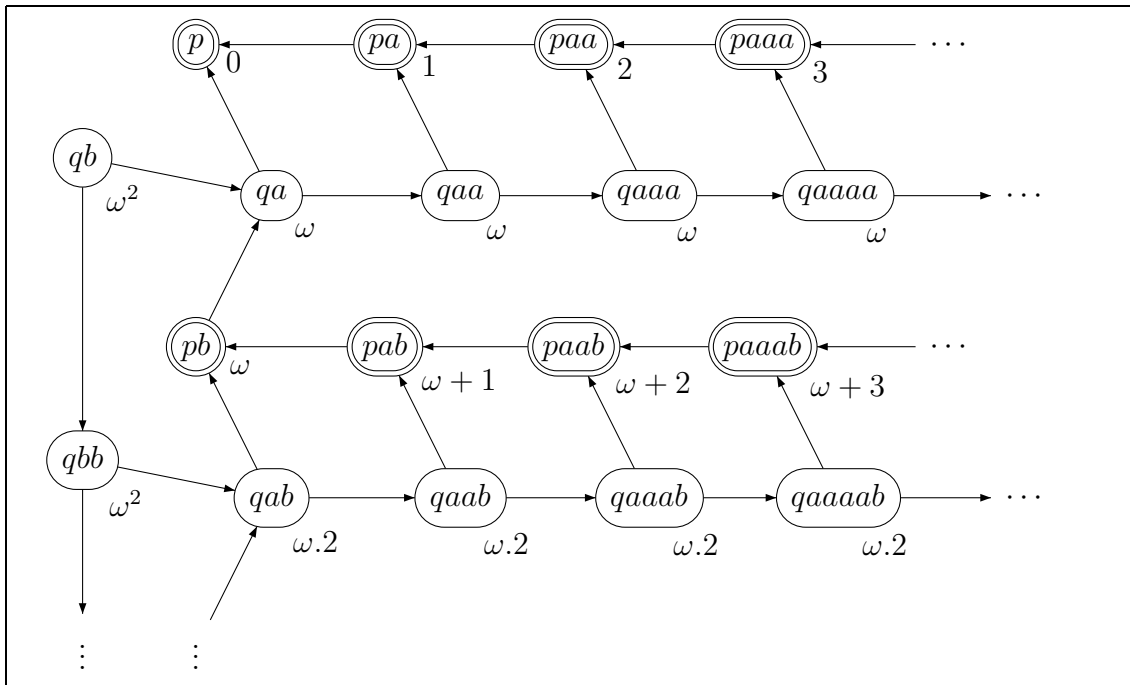


Figure 3.9: Some ordinals (Section 3.4.4)

This example can be extended, using new control states and new stack letters. In Figure 3.9 we have added external vertex labels that are ordinals. A vertex is

labeled by an ordinal α if and only if α is the greatest ordinal such that this vertex is in $\text{Büchi}_0^\alpha(R)$. So the fixed point is first reached with $\text{Büchi}_0^{\omega^2+1}(R)$. The formal definition of the PDGS is:

$$\begin{aligned}\Gamma &= \{a, b\}, P_0 = \{p, q\}, P_1 = \emptyset, \\ \Delta &= \{qb \hookrightarrow qbb, qb \hookrightarrow qa, qa \hookrightarrow qaa, qa \hookrightarrow p, pa \hookrightarrow p, pb \hookrightarrow qa\}, \\ F &= \{p\}, R = p\Gamma^* = p\{a, b\}^*.\end{aligned}$$

Applying Algorithm 3.4.8, we obtain successively the automata of Figure 3.10. In some sense four generation are sufficient to reach ω^2 on this particular example. This can be again extended, using new control states or new stack letters, in such a way that the fixed point is first reached at $\omega^k + 1$ for any $k > 0$.

3.5 Safety Game and Co-Büchi Game

In Sections 3.1 and 3.2 we have considered reachability games for Player 0, where the task is to reach a set R . This corresponds to *safety* games for Player 1: he has to stay in $V \setminus R$:

Player 1 wins a play if it is infinite and never reaches R ,
or ends in a deadlock for Player 0.

We can compute the winning region of Player 1: it is $V \setminus \text{Attr}_0(R)$, the complement of the winning region of Player 0, because these games are determined. And using Algorithm 3.1.5 one can determine if a configuration is *not* in the winning region of Player 0, without complementing the automaton. The question is now to compute a winning strategy for Player 1, assuming that we are in his winning region. Recalling Remark 3.1.1, the strategy of Player 1 is easy to formulate: he just has to stay outside of $\text{Attr}_0(R)$.

Given a configuration $u \in V_1 \setminus \text{Attr}_0(R)$, one can use Algorithm 3.1.5 to find a successor of u that is not in $\text{Attr}_0(R)$. One can also store the computation of this algorithm in a pushdown stack and define a pushdown strategy.

In the case of Büchi games it is more complicated. In the previous section we have considered Büchi games for Player 0, such that he wins a play if and only if the goal set R is seen infinitely often. Dually for Player 1 it is a *co-Büchi game*:

Player 1 wins a play if it meets only finitely often the goal set R ,
or ends in a deadlock for Player 0.

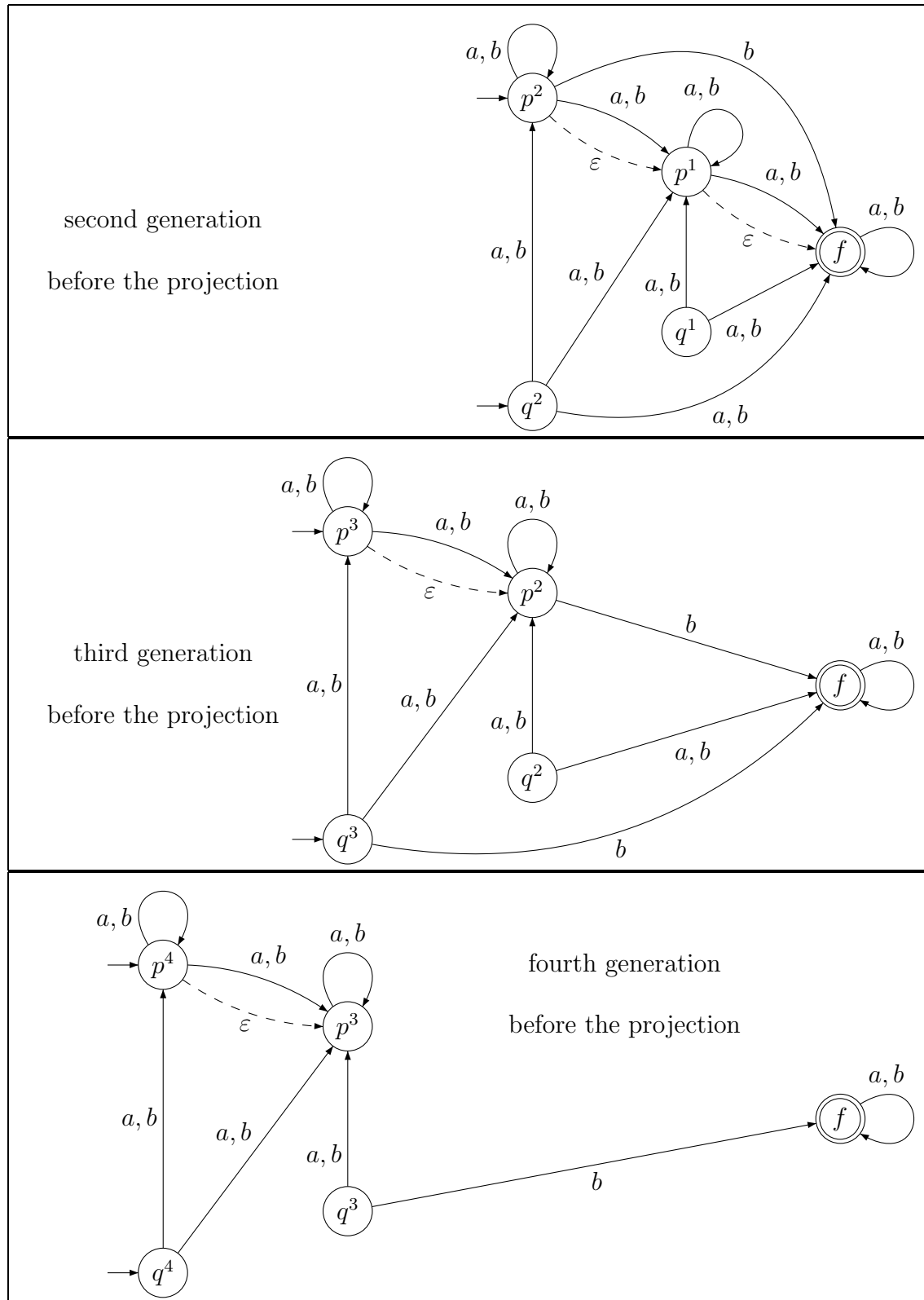


Figure 3.10: Automata from Algorithm 3.4.8 and Section 3.4.4, the dashed arrows are removed.

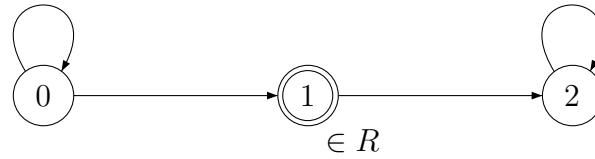


Figure 3.11: Example (3.5.1) of a simple co-Büchi game

We can again compute the winning region of Player 1: it is the complement of the winning region of Player 0. And using Algorithm 3.1.5 one can determine if a configuration is *not* in the winning region of Player 0, without complementing the automaton. The question is to compute a winning strategy for Player 1, assuming that we are in his winning region. Despite what intuition might tell, it is not always the case that Player 1 can force the play to reach a position from which R is no more met. This fact is illustrated in the game of Example 3.5.1 and Figure 3.11.

Example 3.5.1 *In the game of Figure 3.11, we have $V = V_0 = \{0, 1, 2\}$, so Player 0 makes every decision, E is as depicted and $R = \{1\}$. Player 0 is playing the Büchi game associated to R , and Player 1 the co-Büchi. It is clear that from vertex 0, Player 0 can see R at most one time, and loses the game. But he can also stay in vertex 0 as long as he wants, and have the possibility to visit R at some time.*

Looking carefully at the set-theoretic definitions of $Attr_0^+$ and $Büchi_0$, it is possible to complement them, and when written in a suitable way, one can deduce the following positional winning strategy for Player 1.

- Player 1 wins from an initial configuration $u \in V$ if and only if $u \notin Büchi_0(R)$.
- From a configuration $u \in V_1 \setminus Büchi_0(R)$, determine the ordinal α such that $u \in Büchi_0^\alpha(R) \setminus Büchi_0^{\alpha+1}(R)$, meaning that from u Player 0 can force the game to visit R at most “ α times”. Then go to a vertex that is also in $Büchi_0^\alpha(R) \setminus Büchi_0^{\alpha+1}(R)$ and not in R , or to a vertex that is in R and in $Büchi_0^{\alpha-1}(R) \setminus Büchi_0^\alpha(R)$.

We conjecture that one can use the algorithm of the previous section to transform this description of the strategy into an effective one. In the case that α is an integer, it is clear:

- First compute the set $Büchi_0(R)$ using Algorithm 3.4.8.

- If $u \notin \text{Büchi}_0(R)$, use Algorithm 3.4.4 to compute successively the alternating automata representing the sets $\text{Büchi}_0^i(R)$ for $i = 0, 1, 2, \dots$ until $u \notin \text{Büchi}_0^i(R)$.
- Test for the successors of u whether the conditions above are fulfilled

If α is greater or equal ω , we conjecture that one can use the projections π^i of Algorithm 3.4.8 only a restricted number of times to determine α and play to the right vertex.

We see here that with this method the computational effort is very much bigger than for the Büchi game, because the automata \mathcal{B}_i for $\text{Büchi}_0^i(R)$ are bigger and bigger. It is not clear whether one can implement this strategy as a pushdown strategy like before, because we consider different alternating automata.

3.6 A Σ_3 Winning Condition

After having considered reachability and Büchi winning condition, that can be defined as well on finite graphs, we study infinite two-player games over pushdown graphs with a winning condition that refers explicitly to the infinity of the game graph: a play is won by player 0 if some vertex is visited infinity often during the play. It can be shown that the set of winning plays is a proper Σ_3 -set in the Borel hierarchy, thus transcending the Boolean closure of Σ_2 -sets which arises with the standard automata theoretic winning conditions (such as the Muller, Rabin, or parity condition). We will show that this Σ_3 -game over pushdown graphs can be solved effectively (by a computation of the winning region of player 0 and his memoryless winning strategy). This seems to be a first example of an effectively solvable game beyond the second level of the Borel hierarchy.

3.6.1 Motivation

The Muller and parity winning conditions (as well as related ones like Rabin and Streett conditions) define sets of plays which are located at a very low level of the Borel hierarchy, namely in $\mathcal{B}(\Sigma_2)$, the Boolean closure of the Borel class Σ_2 . This restriction to winning conditions of low set theoretical complexity is justified by two reasons: First, most winning conditions which are motivated by practical applications (safety, liveness, assume-guarantee properties, fairness, etc.), and Boolean combinations thereof, all define sets in $\mathcal{B}(\Sigma_2)$. Secondly, by Büchi's and

McNaughton’s results on the transformation of monadic second-order logic formulas into deterministic Muller automata, any winning condition which is formalizable in linear time temporal logic or in monadic second-order logic (S1S) over infinite strings defines a $\mathcal{B}(\Sigma_2)$ -set. (One transforms a logical formula φ into an equivalent deterministic Muller automaton, say with transition graph G_φ , and proceeds from a game graph G and a winning condition defined by φ to $G \times G_\varphi$ as game graph equipped with the Muller winning condition applied to the second components of vertices.) In this connection, Büchi claims in [Büc83, p. 1173] as a general thesis that any set of ω -sequences with an “honestly finite presentation” (by some form of “finite-state recursion”) belongs to $\mathcal{B}(\Sigma_2)$.

Recently, the Büchi-Landweber Theorem was extended to infinite game graphs, and in particular to the transition graphs of pushdown automata [KV00, Sei96, Wal96b]. For example, it was shown by Walukiewicz [Wal96b] that parity games over pushdown graphs can be solved effectively. But the restriction to the parity condition is now only justifiable by pragmatic aspects, and it is well conceivable that higher levels of the Borel hierarchy are reachable by natural winning conditions exploiting the infinity of pushdown transition graphs.

In the rest of this chapter we propose such a winning condition, by the requirement that (in a winning play) there should be one vertex occurring infinitely often. Syntactically, this is formulated as a condition on a play ρ using a Σ_3 -prefix of unbounded quantifiers:

“there is a vertex v such that for all time instances t there is $t' > t$ such that v is visited at t' in the play ρ under consideration”

In [CDT02] it was shown that for a suitable deterministic pushdown automaton the corresponding set of winning plays forms indeed a Σ_3 -complete set in the Borel hierarchy. The completeness proof needs some prerequisites of set theory, in particular on continuous reductions and the Wadge game [Wad84], and is outside the scope of this thesis. Another motivation comes from the context of verification and was pointed out by Olivier Serre [BSW03]. The stack models the (recursive) procedure invocations and in this game Player 1 wants to make the stack “explode”, *i.e.*, show that the program needs infinite resources.

In Section 3.6.3 we show, following [CDT02], that the Σ_3 -winning condition does not prohibit an algorithmic solution of the corresponding games. Building on the approach of previous sections for Büchi games, we present an algorithm to decide whether a given vertex of a pushdown transition graph is in the winning region of player 0; and from this, also a memoryless winning strategy can be extracted.

This result may be considered as a first tiny step in a far-reaching proposal of Büchi ([Büc83, p.1171-72]). He considers constructive game presentations by “state-recursions”, as they arise in automata theoretic games, and he asks to extend the construction of winning strategies in the form of “recursions” (i.e., algorithmic procedures) from the case of $\mathcal{B}(\Sigma_2)$ -games to appropriate games on arbitrary levels of the Borel hierarchy.

3.6.2 Outline of the Solution

The winning condition is now the following: the play is won by Player 0 if and only if

$$\text{there is a configuration from } V \text{ that appears infinitely often in the play,} \quad (3.7)$$

equivalently, if and only if for some length n a configuration of length n is visited infinitely often. Our aim is to compute the set W_0 of winning positions of Player 0: the positions from which he can win whatever Player 1 does.

For technical reasons in this section we modify slightly the definition of $Attr_0^+$: it is convenient to allow deadlocks by the empty stack in the game graph and to declare here Player 1 as the winner of any play terminating with empty stack.

$$\begin{aligned} X_0(T) &= \emptyset, \\ X_{i+1}(T) &= X_i(T) \cup \{u \in V_0 \mid \exists v, (u, v) \in E, v \in T \cup X_i(T)\} \\ &\quad \cup \{u \in V_1 \mid |u| > 1, \forall v, (u, v) \in E \Rightarrow v \in T \cup X_i(T)\}, \\ Attr_0^+(T) &= \bigcup_{i \geq 0} X_i(T). \end{aligned}$$

We are now able to define $Büchi_0(T)$, the set of those configurations from which Player 0 can force to *really* reach T infinitely many times (to win the “Büchi game for T ”, without deadlock): it is the greatest fixed point of the function

$$S \mapsto Attr_0^+(S \cap T),$$

and can also be obtained by induction on the ordinals

$$\begin{aligned} Büchi_0^0(T) &= V, \\ Büchi_0^{\alpha+1}(T) &= Attr_0^+(Büchi_0^\alpha(T) \cap T) \quad \text{for any ordinal } \alpha, \\ Büchi_0^\lambda(T) &= \bigcap_{\alpha < \lambda} Büchi_0^\alpha(T) \quad \text{for a limit ordinal } \lambda. \end{aligned}$$

There is some α such that $Büchi_0^{\alpha+1}(T) = Büchi_0^\alpha(T) = Büchi_0(T)$. We note $\Gamma^{\leq M}$ the language $\{\varepsilon\} \cup \Gamma^1 \cup \dots \cup \Gamma^M$. The effective solution of pushdown games with winning condition (3.7) is based on the following straightforward representation of the winning region W_0 of player 0:

Proposition 3.6.1 *Over a game graph induced by a pushdown game system, the winning region W_0 of Player 0 w.r.t. winning condition (3.7) is*

$$W_0 = \bigcup_{M>0} \text{Büchi}_0(P\Gamma^{\leq M}) .$$

Note that this leads also to the (non-effective) existence of a positional winning strategy for Player 0 on his winning region:

- Given a vertex $u \in W_0$, find the smallest M such that $u \in \text{Büchi}_0(P\Gamma^{\leq M})$.
- Play according to the Büchi game of $P\Gamma^{\leq M}$, see corresponding Remark 3.1.1 and the beginning of Section 3.4.

Although intuitively Player 1 also has a *positional* winning strategy on his winning region, it is not clear how to prove this result or how to define a positional strategy from the previous sets $\text{Büchi}_0(P\Gamma^{\leq M})$. A configuration $u \in V$ is in the winning region of Player 1 if and only if for every $M > 0$ it is *not* in $\text{Büchi}_0(P\Gamma^{\leq M})$. Similarly to the case of co-Büchi, it follows that for each M , Player 1 has a strategy to visit $P\Gamma^{\leq M}$ only a finite number of times. But it seems difficult to arrange these different strategies. For that reason until now we cannot apply the results of Section 2.3.4 to compute (using MSO) the winning region and a winning strategy. In that sense the results of this section should be considered as a new decidability result.

Let us refine this proposition into an algorithmic description of W_0 . In the previous sections we have shown that if the set T is regular, then one can compute a finite automaton recognizing $\text{Attr}_0(T)$, respectively $\text{Attr}_0^+(T)$, which hence are again regular. Using the regularity of $\text{Attr}_0^+(T)$ one can compute a finite automaton recognizing $\text{Büchi}_0(T)$. Of course $\Gamma^{\leq M}$ is regular for $M \geq 0$, so $\text{Büchi}_0(P\Gamma^{\leq M})$ can be computed. To compute the set W_0 of Proposition 3.6.1, we finally have to overcome the problem that W_0 is an infinite union. We shall prove that

$$W_0 = \text{Attr}_0(\text{Büchi}_0(P\Gamma^{\leq N}) \bullet \Gamma^*)$$

where $N = 1 + |\Gamma| |P| \max\{|\nu| - 1 \mid (p, \gamma, q, \nu) \in \Delta\}$, and the set $\text{Büchi}_0(P\Gamma^{\leq N}) \bullet \Gamma^*$ will be defined later (it contains $\text{Büchi}_0(P\Gamma^{\leq N}) \cdot \Gamma^*$, but it is not so easy to define). Roughly speaking, the idea is that under certain conditions on the transitions of the alternating automaton, if Player 1 can make the stack increase by more than N letters, then he can make it increase indefinitely (without returning to previous stack contents an unbounded number of times) and thus wins.

In Proposition 3.6.1 it is essential that the degree of the game graph is finite, as shown by the example of Figure 3.12. Here the game graph is a prefix-recognizable

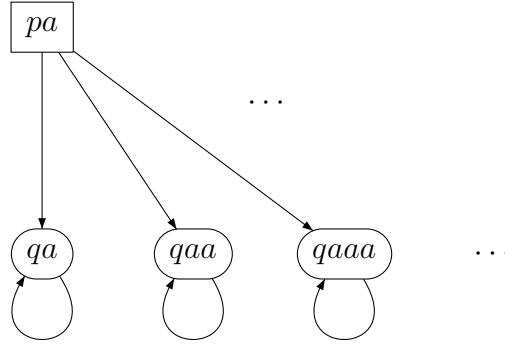


Figure 3.12: Example of a game on a Prefix-recognizable graph

one. In state pa Player 1 has to choose a successor among infinitely many, then the game will stay forever in the new vertex. So each vertex is in W_0 . Note that the set $\text{Büchi}_0(P\Gamma^{\leq M})$ do not contain pa for any $M > 0$, because from pa Player 1 can move to qa^{M+1} .

3.6.3 Details

Let us expose the transformation of a \mathcal{P} -automaton \mathcal{A} recognizing T into a \mathcal{P} -automaton recognizing $\text{Büchi}_0(T)$. We consider the case $T = P\Gamma^{\leq M}$ for a given number M and set

$$\begin{aligned} Y_0^M &= P\Gamma^{\leq M}, \\ Y_{\alpha+1}^M &= \text{Attr}_0^+(Y_\alpha^M) \cap P\Gamma^{\leq M} \quad \text{for any ordinal } \alpha, \text{ and } , \\ Y_\lambda^M &= \bigcap_{\alpha \leq \lambda} Y_\alpha^M \quad \text{for a limit ordinal } \lambda . \end{aligned}$$

We note Y_∞^M the fixed point of this computation: $Y_\infty^M = \text{Attr}_0^+(Y_\infty^M) \cap P\Gamma^{\leq M}$. Then $\text{Büchi}_0(P\Gamma^{\leq M}) = \text{Attr}_0(Y_\infty^M)$.

Consider a PDGS $\mathcal{P} = (P_0, P_1, \Gamma, \Delta)$ with $P = P_0 \cup P_1$. The construction of the automaton recognizing Y_∞^M starts with a \mathcal{P} -automaton \mathcal{B}_0 which recognizes $P\Gamma^{\leq M}$: its state set is $P \cup \{f_0, \dots, f_M\}$, with transitions $f_i \xrightarrow{\Gamma} f_{i+1}$ for $i < M$, each f_i being a final state, and the states of $P \cup \{f_0\}$ are merged into a unique state named f_0 , *i.e.*, f_0 is initial.

Like in Section 3.4, in stages or “generations” $i = 1, 2, 3, \dots$ new copies of P are added. We write (q, i) or short q^i for the copy of a node $q \in P$ added in stage i .

So the state space will be a subset of $(P \times \mathbb{N}) \cup \{f_0, \dots, f_M\}$ (where $q^0 = f_0$ for all $q \in P$). We write P^i for the set $P \times \{i\}$. The auxiliary operations ϕ and π from Section 3.4 are also needed.

Algorithm 3.6.2 *To compute an automaton recognizing Y_∞^M*

Input: PDGS $\mathcal{P} = (P_0, P_1, \Gamma, \Delta)$ and $M > 0$

Output: a \mathcal{P} -automaton \mathcal{C} that recognizes Y_∞^M

Initialization: Set $\mathcal{C} := \mathcal{B}_0$ recognizing $P\Gamma^{\leq M} = Z_0$, with states q^0 (for $q \in P$) and f_0, \dots, f_M , where for all $q \in P$, q^0 is set to be f_0 . (Recall that for all $\gamma \in \Gamma, f_i \xrightarrow{\gamma} f_{i+1}$, and the f_i 's are the final states.)

$i := 0$.

repeat

$i := i + 1$ (i is number of the current generation)

Add the states q^i , for each $q \in P$, using them as initial states.

Add an ε -transition from q^i to q^{i-1} for each $q \in P$

{ obtain an automaton still recognizing Z_{i-1} }

Add new transitions to \mathcal{C} by the saturation procedure of Algorithm 3.1.2:

repeat

(Player 0) if $p \in P_0, p\gamma \hookrightarrow q\mu \in \Delta$ and $q^i \xrightarrow{\mu} S$ in the current automaton, then add a new transition $p^i \xrightarrow{\gamma} S$.

(Player 1) if $p \in P_1, \{p\gamma \hookrightarrow q_1\mu_1, \dots, p\gamma \hookrightarrow q_n\mu_n\}$ are all the Δ -rules (game moves) starting from $p\gamma$ and $\forall k, q_k^i \xrightarrow{\mu_k} S_k$ in the current automaton, then add a new transition $p^i \xrightarrow{\gamma} \bigcup_k S_k$.

until no new transition can be added

{ the obtained automaton recognizes $\text{Attr}_0(Z_{i-1})$ }

remove the ε -transitions.

{ obtain $\mathcal{B}_i^!$ recognizing $\text{Attr}_0^+(Z_{i-1}) = Z_i^!$ }

replace each transition $q^i \xrightarrow{\gamma} S$ by $q^i \xrightarrow{\gamma} \pi^i(S)$.

{ obtain \mathcal{B}_i'' recognizing $Z_i'' \subseteq Z_i^!$ }

replace each transition $q^i \xrightarrow{\gamma} S$ by $q^i \xrightarrow{\gamma} S \cup \{f_0\}$

{ obtain \mathcal{B}_i recognizing $Z_i'' \cap P\Gamma^{\leq M} = Z_i$, we have $\bigcap_{i \geq 0} Y_i^M \subseteq Z_i$ }

set $\mathcal{C} := \mathcal{B}_i$, finishing generation number i

until $i > 1$ and $\forall p, \gamma : p^i \xrightarrow{\gamma} S \iff p^{i-1} \xrightarrow{\gamma} \phi(S)$.

Note that we can erase the q^{i-1} 's and their transitions as soon as the generation i is done. To compare successive generations we have the following property.

Proposition 3.6.3 *In Algorithm 3.6.2, for all $\nu \in \Gamma^*$, $q \in P$, $i \geq 1$ we have*

$$q^{i+1} \xrightarrow{\nu}_* S \Rightarrow q^i \xrightarrow{\nu}_* \phi(S) .$$

The proofs of this proposition and of the following theorem are similar to the corresponding claims in the previous sections. Note that because of the projection π , the transitions $q^i \xrightarrow{\nu}_* S$ verify $S \subseteq (P \times \{i\}) \cup \{f_0, \dots, f_M\}$. Note also that no new transition from the states f_0, \dots, f_M is added.

Proof: We proceed by induction on i . The proposition is a direct consequence of the same property over the transitions (by induction on the length of the word).

- For $i = 1$, we use another induction on the number of transitions starting from p^2 added by the saturation procedure: *during* the saturation procedure, for each new transition $p^2 \xrightarrow{\gamma} S$, we want to check that $p^1 \xrightarrow{\gamma} \phi(\pi^2(S) \cup f_0)$ (because at the end of generation 2, we will have $p^2 \xrightarrow{\gamma} \pi^2(S) \cup f_0$).

As a preliminary remark, we observe that once the first generation is done, each path $q^1 \xrightarrow{\mu}_* S$ from a state q^1 is such that $S \subseteq P \times \{1\} \cup \{f_0, \dots, f_M\}$, hence $\pi^2(S) \subseteq P \times \{2\} \cup \{f_0, \dots, f_M\}$, $\phi(\pi^2(S)) \subseteq P \times \{1\} \cup \{f_0, \dots, f_M\}$, and $\phi(\pi^2(S)) = S$. We also know that $f_0 \in S$.

- At the beginning of the second iteration, one has no other transitions from the states q^2 than $q^2 \xrightarrow{\varepsilon} \{q^2\}$ and $q^2 \xrightarrow{\varepsilon} \{q^1\}$, which are temporary.

- *During* the saturation procedure, as a new transition $p^2 \xrightarrow{\gamma} S$ is added, it is through an existing path $q^2 \xrightarrow{\mu}_* S$ (see algorithm).

If this path uses just ε -transitions ($\mu = \varepsilon$, $S \subseteq P \times [1, 2]$), then a similar path from q^1 existed during the first iteration, generating a transition $p^1 \xrightarrow{\gamma} \phi(\pi^2(S))$ (where we choose to stay in $P \times \{1\}$). Hence at the end of first generation, one gets $p^1 \xrightarrow{\gamma} \pi^1(\phi(\pi^2(S))) \cup f_0$, and $\pi^1(\phi(\pi^2(S))) \cup f_0 = \phi(\pi^2(S)) \cup f_0 = \phi(\pi^2(S) \cup f_0)$. Whereas at the end of the second generation, the transition generated is actually $p^2 \xrightarrow{\gamma} \pi^2(S) \cup f_0$.

If the first segment of this path $q^2 \xrightarrow{\mu}_* S$ is a “real” transition $q^2 \xrightarrow{\alpha} T$, $\alpha \in \Gamma$, then by induction hypothesis one has also $q^1 \xrightarrow{\mu}_* \phi(\pi^2(S) \cup f_0)$, and the corresponding transition from p^1 was added.

If the first segment is $q^2 \xrightarrow{\varepsilon} q^1 \xrightarrow{\alpha} T$, then similarly, because of $q^1 \xrightarrow{\alpha} T$ already existing, the corresponding transition from p^1 was added.

- Induction hypothesis: $\forall S, p^i \xrightarrow{\alpha} S \Rightarrow p^{i-1} \xrightarrow{\alpha} \phi(S)$.
- The proof for $i + 1$ follows the pattern of case $i = 1$.

■

Lemma 3.6.4 *The automaton \mathcal{C} constructed in Algorithm 3.6.2 recognizes Y_∞^M .*

Proof:

Proof of termination

Thanks to the projections π^i 's, there is only bounded number of possible transitions from each row of p^i 's. And thanks to Proposition 3.6.3, there is less and less transitions until the algorithm reaches a fixed point.

Proof of correctness

We note Z_i the language recognized by \mathcal{C} from the initial states p^i 's. We denote $n+1$ the last generation of the algorithm, which is such that $Z_n = Z_{n+1}$, by convention we still consider $Z_i = Z_n$ for all $i \geq n$. One has to show that $Z_n = Y_\infty^M$.

We consider the intermediate results (stages) of Algorithm 3.6.2: near the end of the i -th generation, just after one removes the ε -transitions, one gets the automaton \mathcal{B}'_i , recognizing Z'_i . Then, just after the projection, one gets \mathcal{B}''_i , recognizing Z''_i . Finally one gets \mathcal{B}_i , recognizing Z_i , by replacing each transition $p^i \xrightarrow{\gamma} S$ by $p^i \xrightarrow{\gamma} S \cup \{f_0\}$.

First part: $Z_n \subseteq Y_\infty^M$.

We prove by induction on i that for all i , $Z_i \subseteq Y_i$.

- Remembering that for all $p \in P$, p^0 is set to be f_0 , $Z_0 = Y_0 = P\Gamma^{\leq M}$.
- Induction hypothesis: $Z_i \subseteq Y_i$.
- The algorithm first determines the Attractor^+ of the language. By monotonicity,

$$Z'_{i+1} = \text{Attr}_0^+(Z_i) \subseteq \text{Attr}_0^+(Y_i) = Y'_{i+1} .$$

After the projection of the transitions, the obtained language Z''_{i+1} is contained in Z'_{i+1} : Proposition 3.6.3 shows that an accepting path from a state p^{i+1} was possible before the projection (through the states p^i). So $Z''_{i+1} \subseteq Z'_{i+1} \subseteq Y'_{i+1}$ (for $i = 0$ the projection does not change any transition). The next operation of the algorithm “computes” the intersection with $P\Gamma^{\leq M}$. Here we just need an inclusion. In the resulting automaton \mathcal{B}_{i+1} , each transition from a p^{i+1} has at least a “branch” that goes to $\{f_0, \dots, f_M\}$, so it is clear that $Z_{i+1} \subseteq P\Gamma^{\leq M}$. Also $Z_{i+1} \subseteq Z''_{i+1}$, and

$$Z_{i+1} \subseteq Z''_{i+1} \cap P\Gamma^{\leq M} \subseteq Y'_{i+1} \cap P\Gamma^{\leq M} = Y_{i+1} .$$

We have that $\forall i, Z_n = Z_{n+1} \subseteq Y_i$, and so $Z_n \subseteq Y_\omega^M$. For ordinals greater than ω , the argument is the same as in the proof of Theorem 3.4.9, based on the fact that $Z_n = Z_{n+1} \subseteq \text{Attr}_0^+(Z_n) \cap P\Gamma^{\leq M}$. We conclude $Z_n \subseteq Y_\infty^M$.

Second part: $Y_\infty^M \subseteq Z_n$.

We prove by induction on i that for all i , $Y_\infty^M \subseteq Z_i$.

- By construction, $Y_\infty^M \subseteq Y_0 = Z_0$.
- Induction hypothesis: $Y_\infty^M \subseteq Z_i$.
- Before the projection we have:

$$\tilde{Y}^\infty = \text{Attr}_0^+(Y_\infty^M) \subseteq \text{Attr}_0^+(Z_i) = Z'_{i+1} .$$

To go to Z''_{i+1} , we proceed by induction on the number of transitions that are changed by the projection, *i.e.*, we consider the successive automata $\mathcal{A}_0, \dots, \mathcal{A}_m$, where $L(\mathcal{A}_0) = Z'_{i+1}$, $L(\mathcal{A}_m) = Z''_{i+1}$, and \mathcal{A}_{j+1} is obtained from \mathcal{A}_j by “projecting” one transition. We have to prove by induction on m that $\tilde{Y}^\infty \subseteq L(\mathcal{A}_m)$.

- If $m = 0$, $\tilde{Y}^\infty \subseteq L(\mathcal{A}_0) = Z'_{i+1} = Z''_{i+1}$.

- Induction hypothesis: $\tilde{Y}^\infty \subseteq L(\mathcal{A}_m)$.

- We suppose by absurd that there is a configuration $p\mu \in L(\mathcal{A}_m) \setminus L(\mathcal{A}_{m+1})$ such that $p\mu \in \tilde{Y}^\infty$. We choose that of minimal length $|p\mu|$. For each accepting path labeled by $p\mu$ in \mathcal{A}_m , there is a decomposition $p^{i+1} \xrightarrow{\nu} S \xrightarrow{\xi} F \subseteq \{f_0, \dots, f_M\}$ such that $\mu = \nu\xi$, with $q^i \in S$, and in \mathcal{A}_{m+1} : $\neg \exists (q^{i+1} \xrightarrow{\xi} F' \subseteq \{f_0, \dots, f_M\})$ (the transition that is projected was “leading” to q^i in \mathcal{A}_m and is now “leading” to q^{i+1}).

This means that $q\xi \in Z_i$ and $q\xi \notin L(\mathcal{A}_{m+1})$.

If $q\xi \notin L(\mathcal{A}_m)$, then $q\xi \notin \tilde{Y}^\infty$ (Ind. hyp.), and $p\mu$ should not stay in \tilde{Y}^∞ (see the previous sections), hence the contradiction.

If $q\xi \in L(\mathcal{A}_m) \setminus L(\mathcal{A}_{m+1})$, then $q\xi$ cannot be in \tilde{Y}^∞ by hypothesis: $\nu \neq \varepsilon$ and $|q\xi| < |p\mu|$. If $q\xi$ is not in \tilde{Y}^∞ , then $p\mu$ should not be in \tilde{Y}^∞ (see the previous sections), hence the contradiction.

We conclude that $\tilde{Y}^\infty \subseteq L(\mathcal{A}_{m+1})$.

We have now $\text{Attr}_0^+(Y_\infty^M) \subseteq Z''_{i+1}$, hence

$$Y_\infty^M = \text{Attr}_0^+(Y_\infty^M) \cap P\Gamma^{\leq M} \subseteq Z''_{i+1} \cap P\Gamma^{\leq M} .$$

It is now clear that $Z''_{i+1} \cap P\Gamma^{\leq M} \subseteq Z_{i+1}$. (note that the inclusion from right to left was proved above.)

■

It remains to eliminate the quantification on M implicit in $\bigcup_{M>0} \text{Büchi}_0(P\Gamma^{\leq M})$, by choosing a sufficiently large bound for M . We introduce an ordering relation which permits to compare transitions.

Definition 3.6.5 For any $S, S' \subseteq P^i \cup \{f_0, \dots, f_M\}$,

$$S \sqsubseteq S' \Leftrightarrow \begin{cases} S \cap P^i \subseteq S' \cap P^i \text{ and} \\ \max(\{j \mid f_j \in S\} \cup \{-1\}) \leq \max(\{j \mid f_j \in S'\} \cup \{-1\}) \end{cases}$$

The idea is that in case $S \sqsubseteq S'$, one recognizes “more” after a transition $q^i \rightarrow S$ than after a transition $q^i \rightarrow S'$. To compare transitions $q^i \rightarrow S$ and $q^j \rightarrow S'$, with $i < j$, one considers $\pi_j(S)$ and S' with respect to \sqsubseteq . The index j of $f_j \in S$ measures the possibility for Player 1 to increase the length of the stack, and possibly win.

Proposition 3.6.6 In the automaton \mathcal{C} constructed by Algorithm 3.6.2, assume that for a transition $q^i \xrightarrow{\gamma} S$ we have $S \sqsubseteq S'$ for each transition $q^i \xrightarrow{\gamma} S'$ from the same state. If $\ell = \max\{j \mid f_j \in S\} \geq 0$, then from the configuration $q\gamma$, Player 1 has a strategy to reach a configuration where the length of the stack is at least ℓ and such that this ℓ letters will never be “popped”.

Proof: Induction on the number of transitions constructed by the algorithm. Note that the projection π^i does not change the value of ℓ . If $\ell = 0$, the property is trivially true.

In the generation number 1, there is an ε -transition from each q^i to f_0 . If for some q, γ and ℓ we have the hypothesis of the proposition, then it follows that

- either $q \in P_0$ and every transition $q\gamma \hookrightarrow q'\mu \in \Delta$ add at least ℓ letters to the stack, moreover Player 0 has no possibility to decrease the stack length from $q'\mu$ (otherwise we would have a path $q^i \xrightarrow{\mu} S$, and another transition added),
- or $q \in P_1$ and there is a transition $q\gamma \hookrightarrow q'\mu \in \Delta$ that adds at least ℓ letters to the stack, moreover Player 0 has no possibility to decrease the stack length from $q'\mu$ (otherwise we would have a path $q^i \xrightarrow{\mu} S$, and another transition added).

For the next generations we have the same argument, relying on the preceding generations. If $q \in P_1$ then there is a transition $q\gamma \hookrightarrow q'\gamma'\nu \in \Delta$ such that from $q'\gamma'$

Player 1 can reach a configuration with ℓ_1 letters (applying the proposition to the previous generation) and $\ell = \ell_1 + |\nu|$. And similarly for $q \in P_0$. ■

We consider now $N = 1 + |\Gamma||P| \max\{|\mu| - 1 \mid \exists p\gamma \hookrightarrow q\mu \in \Delta\}$. The rightmost factor is the maximal number of letters that can be added to the stack in one move.

Proposition 3.6.7 *In the automaton \mathcal{C} constructed by Algorithm 3.6.2, assume again that for a transition $q^i \xrightarrow{\gamma} S$ we have $S \sqsubseteq S'$ for each transition $q^i \xrightarrow{\gamma} S'$ from the same state. If $\ell = \max\{j \mid f_j \in S\} \geq N$, then from configuration $q\gamma$, Player 1 can win the game by increasing the stack indefinitely.*

Proof: According to the previous proposition, Player 1 can ensure the stack increases by at least ℓ letters, that will *never* be popped. Using an argument similar to the classical pumping argument (see e.g. [HU69]), there exists $(q, \alpha) \in P \times \Gamma$ such that, during this process, two different configurations $q\alpha\nu$ and $q\alpha\xi\nu$ are met ($\nu \in \Gamma^*, \xi \in \Gamma^+$), and the letters of ν and ξ are not scanned (nor changed) any more in the stack after these configurations. This proves that continuing from $q\alpha\xi\nu$, Player 1 can force the stack to increase indefinitely. This shows that a configuration in $q\gamma\Gamma^*$ cannot be in the winning region W_0 of Player 0. ■

It follows from the proposition that in \mathcal{C} we can eliminate transitions $q^i \xrightarrow{\gamma} S$, such that $f_j \in S, j > N$. So intuitively the computation of Y_∞^N is sufficient to determine Y_∞^M for all $M \geq N$. We have clearly $Y_\infty^N \subseteq W_0$, and also $Y_\infty^N \cdot \Gamma^* \subseteq W_0$ because in the computation of Y_∞^N we have assumed that if the stack is empty, Player 0 loses. Hence $\text{Attr}_0(Y_\infty^N \Gamma^*) \subseteq W_0$, but the equality does not hold in general.

In the automaton obtained from Algorithm 3.6.2 for a given $M > 0$ the next step is to merge the states f_k to a unique final state f , and to add a transition $f \xrightarrow{\gamma} f$ for all $\gamma \in \Gamma$, to obtain an automaton \mathcal{C}' . We will note $Y_\infty^M \bullet \Gamma^*$ the set of configurations recognized by \mathcal{C}' . Nevertheless the set $Y_\infty^M \bullet \Gamma^*$ is defined only by the algorithm, and until now there is no language theoretical definition of $Y_\infty^M \bullet \Gamma^*$ from Y_∞^M . We have $Y_\infty^M \cdot \Gamma^* \subseteq Y_\infty^M \bullet \Gamma^*$ but the equality does not hold in general, because the automaton is alternating. Now we can prove the following.

Corollary 3.6.8 *For all $M \geq N$, $Y_\infty^N \bullet \Gamma^* = Y_\infty^M \bullet \Gamma^*$.*

The intuition behind this “bound” N and this result will be discussed later.

Proof: The inclusion from left to right is clear by monotonicity. For the other inclusion, the automaton recognizing $Y_\infty^M \bullet \Gamma^*$ “contains” that of $Y_\infty^N \bullet \Gamma^*$. It has

possibly some other transitions $q^i \longrightarrow S$, with $f_j \in S, j > N$, which verify the hypothesis of Proposition 3.6.7. Those transitions do not permit to accept a configuration in W_0 , *i.e.*, no winning play from such a configuration is possible. But clearly $Y_\infty^M \bullet \Gamma^* \subseteq \bigcup_{M>0} \text{Büchi}_0(P\Gamma^{\leq M}) \subseteq W_0$ (a play from Y_∞^M is also possible from $Y_\infty^M \bullet \Gamma^*$. See Proposition 3.6.1). ■

Theorem 3.6.9 *Given a pushdown game system, one can compute a finite automaton recognizing the winning region*

$$W_0 = \text{Attr}_0(Y_\infty^N \bullet \Gamma^*)$$

of Player 0 with respect to the Σ_3 -winning condition (3.7).

Proof: Clearly $\text{Attr}_0(Y_\infty^N \bullet \Gamma^*) \subseteq W_0$. Proposition 3.6.1 states that

$$W_0 = \bigcup_{M>0} \text{Büchi}_0(P\Gamma^{\leq M}),$$

which is, by the preceding proposition,

$$\bigcup_{M>0} \text{Attr}_0(Y_\infty^M) \subseteq \bigcup_{M>0} \text{Attr}_0(Y_\infty^M \bullet \Gamma^*) \subseteq \text{Attr}_0(Y_\infty^N \bullet \Gamma^*).$$

■

The construction of an automaton recognizing $W_0 = \text{Attr}_0(Y_\infty^N \bullet \Gamma^*)$ works as follows: one uses Algorithm 3.6.2 with $M = N$. The resulting automaton \mathcal{C} recognizes Y_∞^N . Now one merges the states f_k to a unique final state f , and one adds a transition $f \xrightarrow{\gamma} f$ for all $\gamma \in \Gamma$, in order to obtain an automaton \mathcal{C}' which recognizes $Y_\infty^N \bullet \Gamma^*$. To recognize $\text{Attr}_0(Y_\infty^N \bullet \Gamma^*)$ we just need another application of the saturation procedure as it appears in Algorithm 3.1.2, which finally results in an (alternating) automaton \mathcal{C}'' which recognizes W_0 .

In the definition of Attr_0 , the “usual” convention about deadlocks (with empty stack) is implicitly assumed. So finally at the end of the computation of $\text{Attr}_0(Y_\infty^N \bullet \Gamma^*)$, the states p^{i+1} with $p \in P_1$ has to be marked as final. They correspond to the possibility for Player 0 to win by reaching a configuration with empty stack when Player 1 is on. We did not assumed this convention in the definition of Attr_0^+ because it would have compromised the computation of $\text{Büchi}_0(P\Gamma^{\leq N}) \bullet \Gamma^*$, where we add letters at the end of the stack.

Following the arguments of the previous sections, it is easy to extract a positional winning strategy or a pushdown strategy for player 0 on the set W_0 . The choice of an appropriate transition from a game graph vertex $qw \in W_0$ is done by analyzing an accepting run of the automaton \mathcal{C}' on the input qw .

3.7 Discussion, Examples

The key-point for the termination of our algorithm is the bound N of Proposition 3.6.7 and Corollary 3.6.8, but this bound can be misleading. In a direct sense it is a bound on the size of the automaton that we need to construct, but it seems difficult to interpret this bound directly in the pushdown game that we consider, in terms of configurations and number of letters. Moreover, because we always use alternation, it is difficult to give an intuition for this bound, and for the language $Y_\infty^M \bullet \Gamma^*$ obtained by merging the final states. Recall also that an alternating automaton with n states can have 2^n transitions, and the minimal finite automaton needed to recognize the same language can have 2^n states.

In the case of a usual finite automaton \mathcal{A} recognizing a language L over Γ , we can make the following observation: if there is no transition from final states to non-final states, then merging all final states and adding a loop labeled by the whole alphabet Γ on the new final state results in the recognized language becoming $L \cdot \Gamma^*$. But if we consider alternating automaton it is not the case, as shown in the example of Figure 3.13.

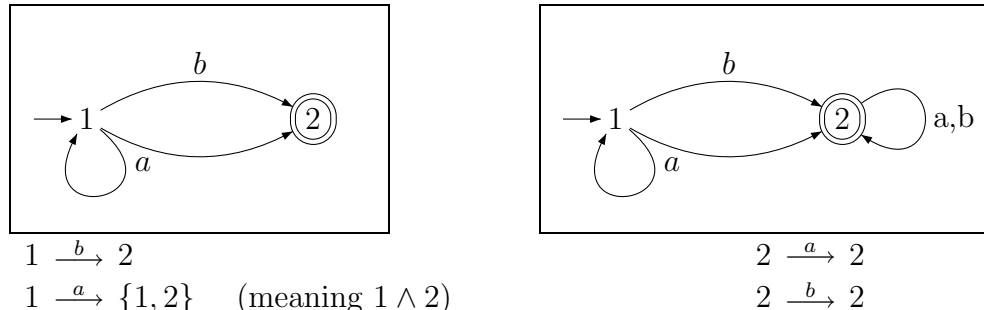


Figure 3.13: A simple alternating automaton, and the same with a loop on the final state

In the first version (left) \mathcal{A} recognizes the finite language $\{b\}$. After the transformation (right), \mathcal{A} not only accepts the words of $b\Sigma^*$, but also those of $a^*b\Sigma^*$. The recognized language is then $\Sigma^*b\Sigma^*$. This simple fact has a consequence in the next example of pushdown game, which is due originally to Olivier Serre, yet simplified here for the exposition.

3.7.1 First Example

Example 3.7.1 We consider the following PDGS:

$$\begin{aligned} \mathcal{P} &= (P_0, P_1, \Gamma, \Delta) \text{ where} \\ P &= P_0 = \{p, q\}, \quad P_1 = \emptyset \\ \Gamma &= \{a, b, c\} \\ \Delta &= \{pb \hookrightarrow p, pa \hookrightarrow pba, pc \hookrightarrow qac, qa \hookrightarrow qaa\} \end{aligned}$$

and the winning condition (3.7) for Player 0.

This game is not very exciting, because the players has no decision to make: the play is completely determined from the initial configuration. One can see that starting with control state q and top letter a , the stack will “explode”, and Player 0 will lose. In control state p , the possible top b 's are first removed, then if an a will lose. In control state p , the possible top b 's are first removed, then if an a appears there is a loop winning for Player 0, but if a c appears it goes to qa and Player 1 wins. See a part of the game graph in Figure 3.14.

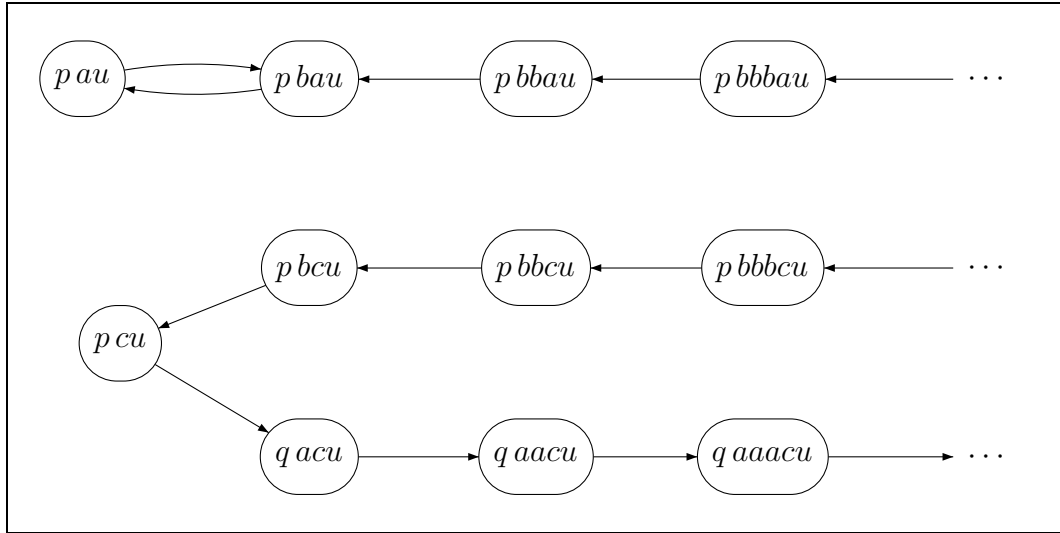


Figure 3.14: Pushdown game of Example 3.7.1, where $u \in \Sigma^*$

We have here $N = 1 + 3 \cdot 2 \cdot 1 = 7$. Applying Algorithm 3.6.2 we get in the first generation the automaton \mathcal{B}'_1 of Figure 3.15 (after the saturation procedure). Then the automaton \mathcal{B}_1 is depicted in Figure 3.16, where there is an AND-transition $p^1 \xrightarrow{b} \{p^1, f_0\}$.

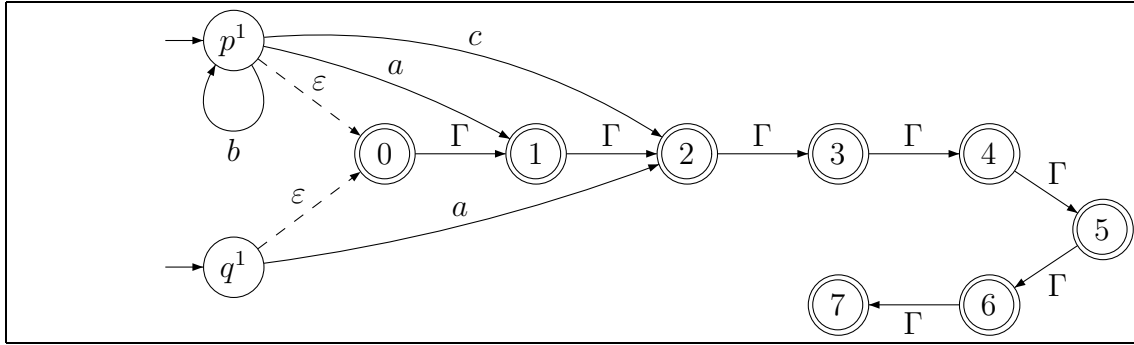


Figure 3.15: Automaton from Algorithm 3.6.2, the dashed arrows are removed to obtain \mathcal{B}'_1 . Unnecessary transitions have been avoided in the picture

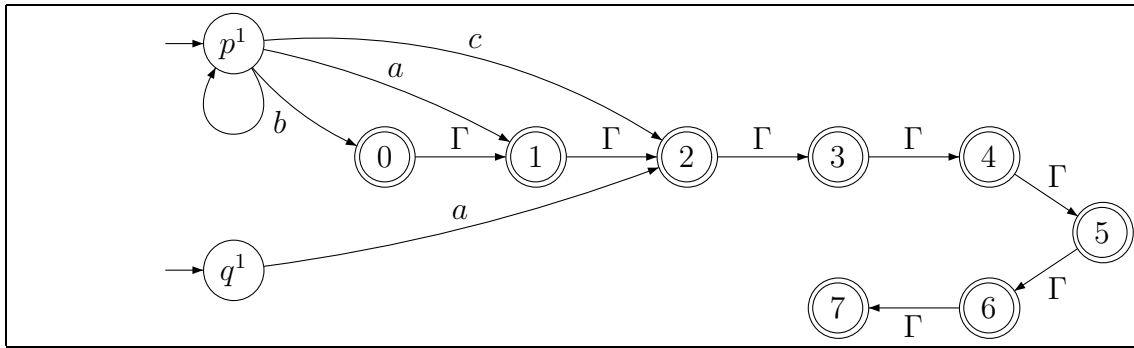


Figure 3.16: Automaton \mathcal{B}_1 (simplified)

In the second generation one gets the automaton \mathcal{B}'_2 of Figure 3.17. After elimination of ε -transition and simplification, the automaton \mathcal{B}_2 is as depicted in Figure 3.18. It is easy to infer (by induction) what are the next generations. In the automaton \mathcal{B}_i we will have following transitions:

$$\begin{aligned}
 p^i &\xrightarrow{a} f_1, \\
 p^i &\xrightarrow{b} \{p^i, f_0\}, \\
 p^i &\xrightarrow{c} f_{i+1} \quad \text{if } i+1 \leq 7, \\
 q^i &\xrightarrow{a} f_{i+1} \quad \text{if } i+1 \leq 7.
 \end{aligned}$$

So the two last transitions will disappear when $i > 6$, and the algorithm will terminate, because \mathcal{B}_7 and \mathcal{B}_8 are identical up to renaming of p^7 in p^8 and q^7 in q^8 .

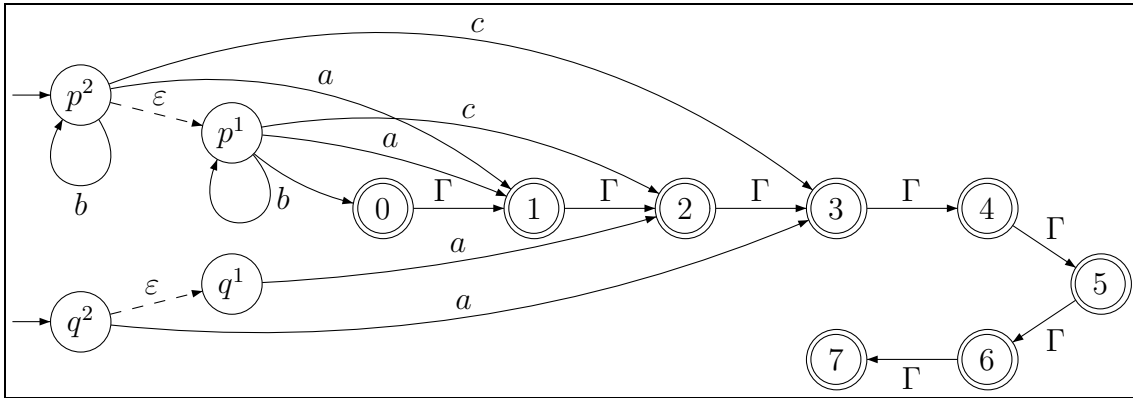


Figure 3.17: Automaton from Algorithm 3.6.2, the dashed arrows are removed to obtain \mathcal{B}'_2 . Unnecessary transitions have been avoided in the picture

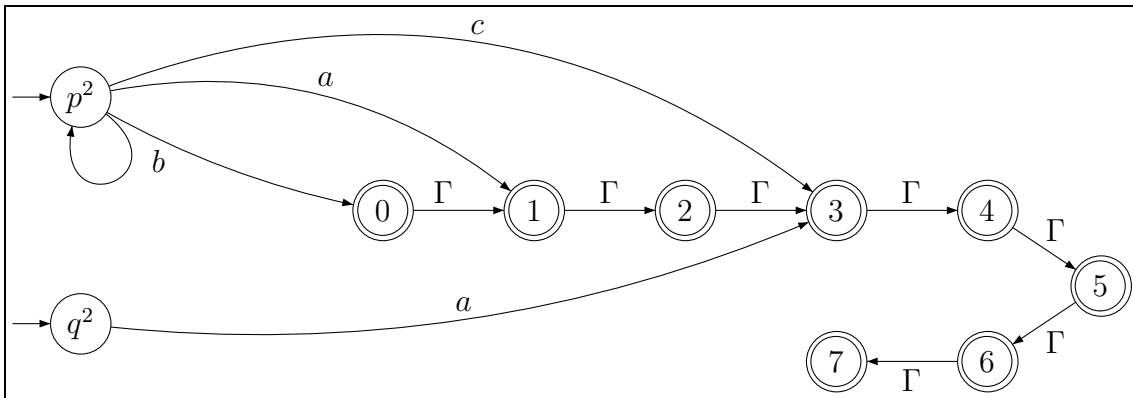


Figure 3.18: Automaton \mathcal{B}_2 (simplified)

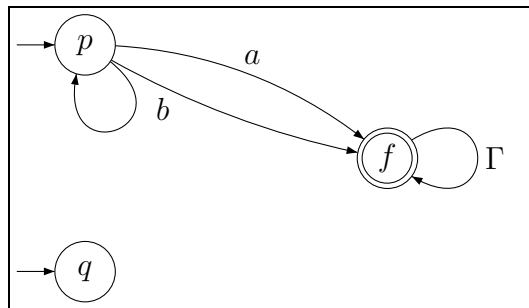


Figure 3.19: Automaton \mathcal{C}' recognizing $Y_\infty^N \bullet \Gamma^*$

After merging the final states, the automaton \mathcal{C}' of Figure 3.19 recognizes $Y_\infty^N \bullet \Gamma^* = b^*a\Gamma^*$. It is exactly the automaton of Figure 3.13 (right), if we consider p as the initial state. If we compute again the attractor of this set, the resulting automaton would be slightly different (with a transition $p \xrightarrow{b} p$), but recognizes the same set. Finally the winning region of Player 0 is

$$W_0 = Attr_0(Y_\infty^N \bullet \Gamma^*) = pb^*a\Gamma^* .$$

What we want here to point out is that this N is *not* a bound on the number of letters that the automaton has to look at (at the top of the store) to determine if a given configuration is in the winning region. The next example is due to Igor Walukiewicz and allow to precise the meaning of Proposition 3.6.7.

3.7.2 Second Example

We want to define a pushdown game with $\mathcal{O}(n)$ control states and the winning condition (3.7) for Player 0, such that Player 0 has to put 2^n letters on the stack in order to win. It means that in this game, if Player 0 pushes down less than 2^n letters on the stack, then the stack will later “explode”. The construction is based on binary counters and was suggested by Igor Walukiewicz.

Definition 3.7.2 *Given $n > 0$, a counter of length n is a word*

$$\sigma_0\sigma_1 \cdots \sigma_{n-1} \in \{0, 1\}^n ,$$

it represents the number $\sum_{i=0}^{n-1} \sigma_i 2^i$.

So we use counters of n bits, and the parameter n is now fixed for the rest of this section without further mentioning. Let $\Gamma = \{0, 1, \#\}$ be the stack alphabet, where $\#$ is a separator. The idea of the game is that Player 1 can force Player 0 to write 2^n counters representing the successive numbers between 0 and $2^n - 1$. Namely for $n = 3$, Player 0 will have to reach the configuration

$$p \#\#\#111\#011\#101\#001\#110\#010\#100\#000\#\# . \quad (3.8)$$

After that Player 1 will check that the counters are correct, and Player 1 wins if and only if he can find a mistake. There is only one control state for Player 0: $P_0 = \{p\}$, with transitions

$$\begin{aligned} p\gamma &\hookrightarrow p\gamma'\gamma, \quad \forall \gamma, \gamma' \in \Gamma, \text{ and} \\ p\# &\hookrightarrow q , \end{aligned}$$

meaning that Player 0 can push whatever he wants before giving control to Player 1 ($q \in P_1$). The play starts with the configuration $p\#\#$. Obviously Player 0 loses if he never stop pushing, see (3.7). We will need two states q_{win} and q_{lost} , respectively winning and losing for Player 0, that can be easily implemented according to the winning condition:

$$\begin{aligned} q_{win}\gamma &\hookrightarrow q_{win}\gamma, \forall \gamma \in \Gamma, \\ q_{lost}\gamma &\hookrightarrow q_{lost}0\gamma, \forall \gamma \in \Gamma. \end{aligned}$$

From control state q , Player 1 has the opportunity to check that the top counter consists only of 1's. This can be implemented with a single control state. Depending on the result, the play will go to q_{win} or q_{lost} .

$$\begin{aligned} q\# &\hookrightarrow \bar{q}, & \bar{q}1 &\hookrightarrow \bar{q}, \\ \bar{q}0 &\hookrightarrow q_{lost}, & \bar{q}\# &\hookrightarrow q_{win}. \end{aligned}$$

Otherwise Player 1 can pop an arbitrary number of letters to find a counter where he thinks that the number of bits is not legal (more than n or less that n).

$$\begin{aligned} q\# &\hookrightarrow q'\#, \\ q'\gamma &\hookrightarrow q', \forall \gamma \in \Gamma, & q'\# &\hookrightarrow q^0, \\ q^i\sigma &\hookrightarrow q^{i+1}, \forall \sigma \in \{0, 1\}, i < n, & q^n\# &\hookrightarrow q_{win}, \\ q^n\sigma &\hookrightarrow q_{lost}, \forall \sigma \in \{0, 1\}, & q^i\# &\hookrightarrow q_{lost}, \forall i < n. \end{aligned}$$

To be complete we also need transitions $q1 \hookrightarrow q_{lost}1$ and $q0 \hookrightarrow q_{lost}0$ to ensure that Player 0 has put a separator $\#$. They will be ignored in the following. We can now assume that every counter has the right length (otherwise Player 1 has a winning strategy). From control state q' with top letter $\#$, Player 1 has also the opportunity to check that the two current top counters represent successive numbers. He choses a position where he thinks that Player 0 was cheating, then exactly $n + 1$ letters are popped to go to the corresponding position in the second counter. Depending on whether a 1 was already seen in the first counter, we know if the bit in the second counter has to be different or equal to the one in the first counter.

$$\begin{aligned} q'\# &\hookrightarrow r, \\ r0 &\hookrightarrow r, & r0 &\hookrightarrow r_1^n, & r_0^i\gamma &\hookrightarrow r_0^{i-1}, \forall \gamma \in \Gamma, i > 0, \\ r1 &\hookrightarrow \bar{r}, & r1 &\hookrightarrow r_0^n, & r_1^i\gamma &\hookrightarrow r_1^{i-1}, \forall \gamma \in \Gamma, i > 0, \\ \bar{r}0 &\hookrightarrow \bar{r}, & \bar{r}0 &\hookrightarrow r_0^n, & r_0^0 &\hookrightarrow q_{win}, & r_0^0 &\hookrightarrow q_{lost}, \\ \bar{r}1 &\hookrightarrow \bar{r}, & \bar{r}1 &\hookrightarrow r_1^n, & r_1^0 &\hookrightarrow q_{win}, & r_1^0 &\hookrightarrow q_{lost}. \end{aligned}$$

- In generation number i , if $i < N$, then we will have a transition $q_{lost} \xrightarrow{\Gamma} i+1$, and if $i \geq N$, no transition from q_{lost} . This is because of the rule $q_{lost}\gamma \leftrightarrow q_{lost}0\gamma$ and represents the fact that Player 0 is losing from q_{lost} .
- From the states q, q', \bar{q}, \dots we will have in each generation exactly the same transitions in \mathcal{B}_i , because every transition of the PDGS from these control states pops a letter from the stack (except from q). From control state q , the PDGS was already like an alternating automaton reading words. Seeing q as an initial state of \mathcal{B}_i , q_{win} as accepting state and q_{lost} as rejecting state, \mathcal{B}_i is just an alternating (universal) automaton that accepts exactly one word: the stack content that Player 0 has to reach (3.8). More formally this is true if we add a loop labeled by Γ on the state q_{win} and declare it as accepting state, in such a way that once a run has reached q_{win} , we don't need to check the rest of the word.

Here it is important to note that this alternating automaton has a linear number of states in n and the configuration that is accepted is of length exponential in n .

- From p the situation is more complicated. If we consider only the rule $p\gamma \leftrightarrow p\gamma'\gamma$ of the PDGS to generate transitions in \mathcal{B}_i , then we will have (like from q_{lost}) a transition $p \xrightarrow{\Gamma} i+1$ in generation i , and then no transition at all for $i \geq N$. Again this is related to the fact that Player 0 loses if he always pushes new letters.

Considering the rule $p\# \leftrightarrow q$, in each generation a transition $p \xrightarrow{\#} q$ will be generated. At first sight this is because Player 0 is winning from the configuration (3.8). But after this transition is created, using the rule $p\gamma \leftrightarrow p\gamma'\gamma$, the algorithm will step by step add new transitions going to the next states $(\bar{q}, r, q^0, q', q^1, q^2, \bar{r}, r_1^n, \dots)$. Namely there will be successively following transitions:

$$\begin{aligned}
p &\xrightarrow{\#} q \\
p &\xrightarrow{\#} \{\bar{q}, r, q^0, q'\} \\
p &\xrightarrow{1} \{\bar{q}, \bar{r}, r_0^n, q^1, q'\} \\
p &\xrightarrow{1} \{\bar{q}, \bar{r}, r_1^n, r_0^{n-1}, q^2, q'\} \\
p &\xrightarrow{1} \{\bar{q}, \bar{r}, r_1^n, r_1^{n-1}, r_0^{n-2}, q^3, q'\}
\end{aligned}$$

and so on. And just because the alternating automaton from initial state q accepts one word, somehow a “path” will be found in this automaton, where

finally q_{win} will be reached in each branch of the path. Because the automaton is alternating, an accepting run from q is best represented by a tree, or a Directed Acyclic Graph, and some branches of the tree will reach q_{win} earlier than some others. The problem is that the accepted word (3.8) has exponential length in n , but the bound N is linear in n , so it will not be possible to go through all the word. We will see later how this problem is solved.

For now we just have to note that these transitions will be the same in each generation and will be present in the final output \mathcal{C} of the algorithm.

After N iterations, the algorithm will terminate and the automaton \mathcal{C} will look like Figure 3.20 yet without transitions $p \xrightarrow{\gamma} 2$ and $q_{lost} \xrightarrow{\gamma} 2$. The configuration $p\#\#$ is not accepted by this automaton for the reason mentioned just above, but the next step of the construction is to merge the states f_k to a unique final state f , and add a transition $f \xrightarrow{\gamma} f$ for all $\gamma \in \Gamma$. Then the saturation procedure is again applied. This time every branch reaching q_{win} is sure to be “accepting”. At the end a transition $p \xrightarrow{\gamma} f$ appears, for all $\gamma \in \Gamma$. Actually in our setting Player 0 can win from *any* configuration: he just have to first push $\#\#$ and then the right counters. The resulting automaton is depicted in Figure 3.21

3.8 Extensions

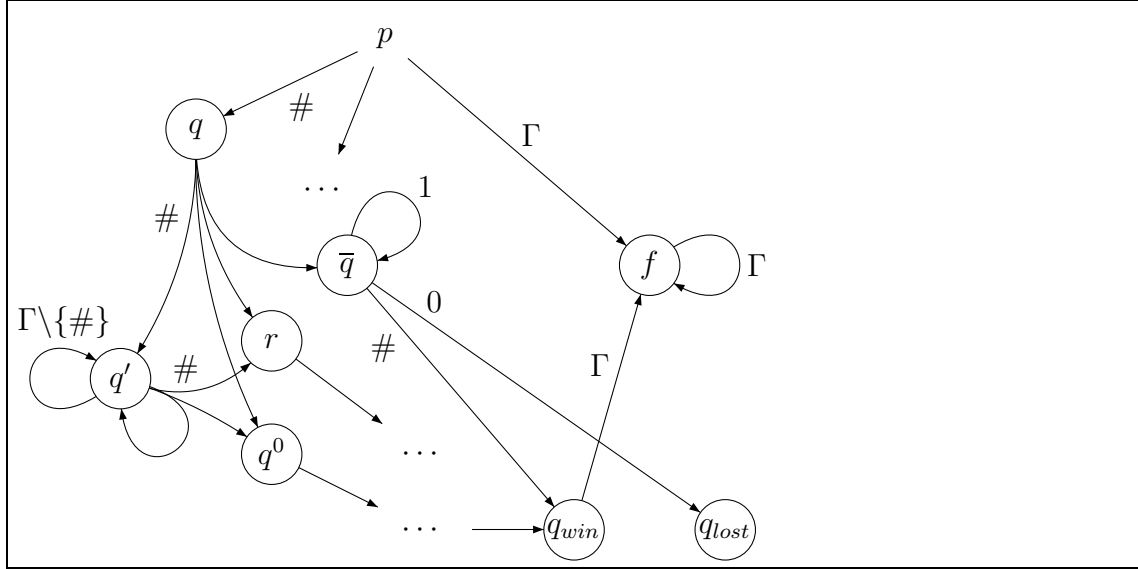
We discuss two extensions of the previous results, the first one concerns the winning condition, the second one concerns the class of graphs treated.

3.8.1 Modifying the Winning Condition

One can refine a little the winning condition considered in the previous section and look at the following: given a subset $F \subseteq P$ of “good” control states, the play is won by Player 0 if and only if

$$\text{there is a configuration from } F\Gamma^* \text{ that appears infinitely often in the play.} \quad (3.9)$$

It is possible to adapt the algorithm to this new winning condition just by adding an ε -transition from q^i to q^{i-1} only if $q \in F$, similarly to the case of the Büchi condition in Algorithm 3.4.8. According to [CDT02] this condition is still a Σ_3 condition. Other conditions has been proposed to reach higher levels of the Borel hierarchy, but it seem not easy to solve the corresponding games using the symbolic approach developed here.



Depicted transitions:

$$\begin{array}{lll}
 p \xrightarrow{\gamma} f, \forall \gamma \in \Gamma & q' \xrightarrow{\gamma} \{q'\}, \forall \gamma \in \Gamma \setminus \{\#\} & \\
 p \xrightarrow{\#} q & \bar{q} \xrightarrow{1} \bar{q} & \\
 q \xrightarrow{\#} \{\bar{q}, r, q^0, q'\} & \bar{q} \xrightarrow{0} q_{lost} & q_{win} \xrightarrow{\gamma} f, \forall \gamma \in \Gamma \\
 q' \xrightarrow{\#} \{r, q^0, q'\} & \bar{q} \xrightarrow{\#} q_{win} &
 \end{array}$$

Figure 3.21: Rough view of Automaton \mathcal{C}'' recognizing W_0 . Now each state is final except p

3.8.2 Prefix-recognizable Graphs

It is possible to extend the symbolic approach — for reachability, Büchi and Σ_3 games — to the case of prefix-recognizable graphs. This costs much more computational effort. We outline how to proceed. We use the same definition as in Section 4.5 (see [Cau96]). Given a finite alphabet Γ , a graph, or set of edges $G \subseteq \Gamma^* \times \Gamma^*$ is a prefix-recognizable graph, or PRG, if and only if

$$G = \{uw \leftrightarrow vw \mid u \in U_i, v \in V_i, w \in W_i, 1 \leq i \leq N\},$$

where for all i , $1 \leq i \leq N$, the U_i, V_i, W_i are regular sets over Γ .

Games over PRG are defined in a natural way. In a configuration $x \in \Gamma^+$, the first letter determines the player whose turn it is. We have $\Gamma = \Gamma_0 \uplus \Gamma_1$, $V_0 = \Gamma_0 \Gamma^*$, and $V_1 = \Gamma_1 \Gamma^*$, similarly to the PDGS. A game starting from $\pi_0 \in \Gamma^+$ is defined in the usual way.

Of course we can represent sets of configurations, and thus winning conditions, by finite automata, but the saturation procedure from Algorithm 3.1.2 needs to be adapted. The first step is to transform the PRG into another PRG which is closer to a PDGS. First we note using Proposition 3.4.3, that one can transform the alphabet Γ and the transition rules so that we know at any time whether the configuration is in a set W_i . Secondly the idea is that a transition $uw \hookrightarrow vw$ is decomposed into a sequence of transitions.

- First the player whose turn it is chooses an $i \leq N$ such that he claims that he can apply a transition of the form $uw \hookrightarrow vw$ for $u \in U_i, v \in V_i, w \in W_i$. This can be implemented by adding a kind of “control state” p_i in front of uw .
- Then letters from u are popped one by one and the control states simulate the finite automaton that recognizes U_i , reading u .
- When the control state corresponds to a final state, *i.e.*, the word u read so far is in U_i , and the rest of the stack is in W_i , then the player is allowed to push any word $v \in V_i$ in one move. This achieves the transition $uw \hookrightarrow vw$.

This idea is close to that of Section 4.5. The transitions that “pop” one letter are just in the form of PDGS-transition and do not pose any problem. For transitions allowing to push any word from a regular language, Algorithm 3.1.2 must be adapted. The new saturation procedure is as follows, using notations of Algorithm 3.1.2. Typically there is a transition rule $p'_i \hookrightarrow V_i$.

- If this is a rule for Player 0 and *there exists* a $v \in V_i$ and a run $q \xrightarrow{v}^* S$ in $\mathcal{A}_{Att(R)}$, then add a new transition $p'_i \xrightarrow{\varepsilon} S$.
- If this is a rule for Player 1 and there exists $S \subseteq Q$ such that *every* word $v \in V_i$ has a run $q \xrightarrow{v}^* S'$ in $\mathcal{A}_{Att(R)}$ for some $S' \subseteq S$, then add a new transition $p'_i \xrightarrow{\varepsilon} S$.

The computation of such an S needs at each step much effort, that’s why we do not want to follow this track any longer. The Büchi game can also be solved using this transformation and Algorithm 3.4.8. For a Σ_3 -game one has to be careful because in the decomposition of a transition of the PRG new configurations appear that might be visited infinitely many times even if in the original game the Σ_3 condition is not satisfied. Using the refined winning condition 3.9 above it is possible to solve this problem.

We do not formalize the transformation of the PRG presented above, this is done in detail in Section 4.5 for the more general framework of parity games. We just want to point out that the considered Σ_3 -games over PRG can be solved using the symbolic approach.

Chapter 4

Game-Reduction and Parity Games over Pushdown Graphs

This chapter and the following are based on the notion of game simulation, that allow to solve a game by reducing it to a simpler game that we know. It can be either a simpler graph with a more complex winning condition, or a more complex graph with a simpler winning condition. This technique has been used for a long time in mathematics, where the second game might be called “auxiliary game”, but it seems that there is no generic name for the reduction.

The fundamental result in this direction is due to Walukiewicz in [Wal96b]. He gave a solution of parity games on pushdown graphs using a reduction to a *finite* graph and a refined winning condition involving claims for one player (see also [Cac02c]). He proved the existence of pushdown strategies and determined the winner with an EXPTIME procedure. He proved also that the solvability of reachability games over pushdown graphs is EXPTIME-complete.

In Section 4.2 we give a new presentation and proof of the results of [Wal96b] stressing upon effectivity. Section 4.3 presents an example of pushdown game. Then in Section 4.4 we extend these results to compute uniformly the winning region of the game (the set of configurations from which Player 0 can win). It is proved to be effectively regular, and a corresponding winning pushdown strategy is also uniformly defined. The result of Section 4.4 has been found independently by Olivier Serre in [Ser03]. In Section 4.5 we consider parity games on prefix-recognizable graphs, which are an extension of pushdown graphs, where the degree of a vertex can be infinite [Cau96]. We show that any prefix-recognizable game can be “game-simulated” by a pushdown game, in the sense that under a certain correspondence of game positions, the winner of one game is the same player as the winner of the other game. An

example is also provided. Applying the uniform solution of Section 4.4, we get a uniform solution and an effective winning strategy also over prefix-recognizable graphs.

4.1 Definition of the Game Simulation

A parity game structure (V_0, V_1, E, Ω) is *game-simulated* by another parity game structure $(V'_0, V'_1, E', \Omega')$ from initial vertices $\pi_0 \in V$ and $\pi'_0 \in V'$ if

- Player 0 wins the parity game $(V'_0, V'_1, E', \Omega')$ from π'_0 if and only if Player 0 wins the parity game (V_0, V_1, E, Ω) from π_0 ,
- from a winning strategy of Player 0 in $(V'_0, V'_1, E', \Omega')$ one can compute a winning strategy of Player 0 in (V_0, V_1, E, Ω) .

In this thesis we use game-simulation only for parity games, but this definition can be extended to other winning conditions. For example in [BJW02] parity games on finite graphs are reduced to safety games (on a larger finite graph). It is also known how to reduce a Muller winning condition to a parity winning condition, see [GTW02, Chap. 1].

Note that this definition is very general and can be applied to different contexts. In some sense as soon as one can solve a game (V_0, V_1, E, Ω) , that is determine the winner and a winning strategy from π_0 , one can say that it is game-simulated by any game $(V'_0, V'_1, E', \Omega')$ (if we know the winner).

4.2 Pushdown Games and Walukiewicz's Results

Given a PDGS $\mathcal{P} = (P_0, P_1, \Gamma, \Delta)$ defining the game graph (V_0, V_1, E) (see Section 2.2.3), we consider a parity winning condition. For that we have a *priority function* $\Omega : P \rightarrow [max]$, extended to the set V of configurations by $\Omega(p\nu) = \Omega(p)$. According to the definition of Section 2.1.3, the parity game structure is (V_0, V_1, E, Ω) .

Sections 3 and 4 of [Wal96b] are not stated in an effective (*i.e.*, algorithmic) framework, and their results “become” effective only with the help of Section 5 of [Wal96b]. We prefer to give first a new presentation of the construction of Section 5 of [Wal96b]. Then the most important results can be deduced, including all algorithmic claims.

The idea of [Wal96b] is to reduce the pushdown game to a parity game on a finite graph. This allows to determine the winner, and also the winning strategy.

We assume the positional (“memoryless”) determinacy of parity games over finite graphs, see [EJ91].

To distinguish the pushdown game and the game on a finite graph, we will call here *Finite State Parity game* (FSP) a parity game structure (S_0, S_1, E, λ) such that the set $S_0 \uplus S_1$ of vertices is finite. The term “state” is an analogy to the case of finite automaton. From now on we use the infix notation \rightarrow for the edge relation: $\forall s, s' \in S, (s, s') \in E \Leftrightarrow s \rightarrow s'$. We restrict here $\Delta \subseteq P \times \Gamma \times P \times \Gamma^{\leq 2}$, where $\Gamma^{\leq 2} = \varepsilon \cup \Gamma \cup \Gamma^2$. In this section we consider a particular initial configuration $\pi_0 = p_0 \perp$, where $\perp \in \Gamma, p_0 \in P$.

We recall the definition of pushdown strategy from Section 3.2.4. A *pushdown strategy* for Player 0 in its general form is a deterministic pushdown automaton with input and output. It “reads” the moves of Player 1 (elements of Δ) and outputs the moves (choices) of Player 0, like a pushdown transducer.

Definition 4.2.1 *Given a PDGS $(P_0, P_1, \Gamma, \Delta)$, where Δ_σ is the set of transition rules in Δ departing from Player σ configurations, a pushdown strategy for Player 0 in this game is a deterministic pushdown automaton $\mathcal{S} = (Q, A, \Pi)$, with a set Q of control states, some stack alphabet A , and a finite transition relation $\Pi \subseteq ((Q \times A \times \Delta_1) \times (Q \times A^*)) \cup ((Q \times A) \times (Q \times A^* \times \Delta_0))$.*

Theorem 4.2.2 [Wal96b] *Given a Pushdown Game System \mathcal{G} with a parity winning condition, one can construct a Finite State Parity game \mathcal{G}' such that: \mathcal{G} is game simulated by \mathcal{G}' . More precisely:*

1. *the winner of the parity game over \mathcal{G} from the initial configuration $p_0 \perp$ is the winner of \mathcal{G}' from a certain initial vertex denoted $Check(p_0, \perp, B, \Omega(p_0))$, where $B \in (\mathcal{P}(P))^{max}$,*
2. *a winning pushdown strategy for Player 0 in the parity game over \mathcal{G} from the initial configuration $p_0 \perp$ can be constructed from a winning strategy for Player 0 in \mathcal{G}' from the initial vertex mentioned above.*

In the sequel we present informally how a play on the PDGS is “simulated” in the FSP. We will see later that a configuration $p\gamma\nu$ of the PDGS, where $p \in P, \gamma \in \Gamma$, and $\nu \in \Gamma^*$, is represented in the FSP by a vertex $Check(p, \gamma, B, m)$, where $m \in [max]$ is a priority, and $B \in (\mathcal{P}(P))^{max}$ “summarizes” information about ν (the number m is the smallest priority seen in a certain part of the game, and B represents the set of control states q such that Player 0 can win the game from $q\nu$, under certain conditions depending on m). To begin with, consider the initial configuration $(p_0 \perp)$, where the symbol $\perp \in \Gamma$ cannot be erased according to the

rules of Δ . The corresponding vertex of the FSP is $Check(p_0, \perp, B, m)$, where in this particular case B and m are not relevant.

From a configuration $p\gamma\nu$, simulated by $Check(p, \gamma, B, m)$, the player whose turn it is is determined by p , in the PDGS as well as in the FSP: either $p \in P_0$ or $p \in P_1$. Let $\sigma \in \{0, 1\}$ such that $p \in P_\sigma$. Different types of moves are possible in the PDGS.

If Player σ chooses a transition (p, γ, p', γ') , *i.e.*, if the stack length remains constant, then the FSP proceeds to the vertex $Check(p', \gamma', B, \min(m, \Omega(p')))$. This means that B remains the same, m is updated for later use and represents the minimal priority seen since last initialization of m (see below). The priority of this vertex is $\Omega(p')$ in the FSP, as well as the corresponding configuration in the PDGS, and the play goes on like that until some “push” or “pop” operation occurs.

The key point is the treatment of the push operation, because one cannot store in the FSP the whole information contained in the stack. If Player σ chooses a transition $(p, \gamma, p', \gamma'\eta) \in \Delta$, *i.e.*, “pushes” one more symbol onto the stack, then in the FSP the corresponding new vertex is $Push(B, m, p', \gamma'\eta)$. This is an intermediate vertex where Player 0 (always) has to make a decision. He has to guess what can happen later, and what he can guarantee. Player 0 chooses a tuple $C \in (\mathcal{P}(P))^{max}$ such that he claims/guesses that whenever the symbol γ' currently at the top of the stack is “popped”, then after this pop operation, the PDGS will be in a control-state $q \in C_\ell$, such that ℓ is the smallest priority seen between the “push” and the “pop” of this γ' . This part of the game is a “subgame” in [Wal96b], and this notion is not so far from the idea of “detour” in [Var98]. More precisely, γ' can be replaced later by another letter, but the condition on C must hold when the length of the stack decreases and symbol η comes at the top of the stack.

So Player 0 goes to the vertex $Claim(B, m, p', \gamma'\eta, C)$, which is a vertex of Player 1. In particular, if $C = (\emptyset, \dots, \emptyset)$, then Player 0 is claiming that the stack will never become again shorter. And Player 0 can claim that the smallest priority that can be seen in the subgame is ℓ by choosing C as $(\emptyset, \dots, \emptyset, C_\ell, \dots, C_{max-1})$. Player 1 has to answer the claim of Player 0: either he thinks that Player 0 is bluffing, and he challenges the claim, or he believes that Player 0 can achieve his claim, and he wants to see what happens after the subgame.

The second case is simple: Player 1 goes to vertex $Jump(q, \eta, B, m, \ell)$ such that $q \in C_\ell$. This is an intermediate vertex which, as a shortcut, simulates one of the above mentioned subgames: among the propositions of Player 0, Player 1 chooses that the smallest priority seen in this subgame was ℓ , and when η appears again at the top of the stack, the new control state is q . The priority of this $Jump(\dots)$ vertex is ℓ in the FSP. Then the play goes on to $Check(q, \eta, B, \min(\ell, m, \Omega(q)))$ without

any alternative.

In the first case, when Player 1 challenges the claim, he goes to vertex $Check(p', \gamma', C, \Omega(p'))$. This means that the last component is reset to $\Omega(p')$, and will remember the minimal priority seen in the subgame we just entered. The tuple C is stored, and whenever a “pop” operation occurs later, it is possible to check if the claim of Player 0 is achieved. If it is, this means immediate win for Player 0. If it is not, this means immediate win for Player 1 (see the proof for details, and above for the update of m). But the play can also stay forever in the $Check()$ vertices, *i.e.*, without “pop”. In this case the winner is determined by the parity condition. In fact the claim of Player 0 after a push operation means also that if no pop occurs later, then he has to win the subgame just with the parity condition.

We restrict ourselves here to the following form of pushdown strategy. We consider a strategy automaton (Q, A, Π) where $Q = P = P_0 \uplus P_1$, $A = \Gamma \times \Sigma$, Σ is any alphabet, and $\Pi \subseteq ((P_1 \times A \times \Delta_1) \times (P \times A^*)) \cup ((P_0 \times A) \times (P \times A^* \times \Delta_0))$. Moreover we have the condition that whenever the game is in a configuration $p\gamma_0 \cdots \gamma_n$, the strategy automaton should be in a configuration $p(\gamma_0, \sigma_0) \cdots (\gamma_n, \sigma_n)$, which means that the strategy has more information in its stack, represented by $\sigma_0 \cdots \sigma_n$, but follows the play. If $p \in P_0$, then $p(\gamma_0, \sigma_0)$ determines the move of Player 0 w.r.t. Π , and the strategy updates its stack. If $p \in P_1$, then for any move of Player 1, *i.e.*, for any transition in Δ_1 , the strategy should update its stack. At the beginning of the play, the strategy has to be initialized properly, according to the initial configuration of the game. Then for each move of the play, the strategy executes a transition.

In our particular form of strategy, there is a redundancy in the transition relation: suppose $p(\gamma_0, \sigma_0)$ is the top of the stack, if $p \in P_0$, then a unique transition is possible in the strategy, and the output of the move $\delta_0 \in \Delta_0$ can be deduced from the update of the stack. If $p \in P_1$, then a unique transition can follow the choice of Player 1 and update the stack accordingly, so the input of $\delta_1 \in \Delta_1$ is redundant. From now on we consider $\Pi \subseteq (P \times A) \times (P \times A^{\leq 2})$.

Formally,

- if $p \in P_0$, then $\forall a \in A \exists!(p, a, p', w) \in \Pi$. Moreover if $(p, a, p', w) \in \Pi$ and $a = (\gamma, \sigma)$, $w = (\gamma_1, \sigma_1) \cdots (\gamma_k, \sigma_k)$ ($2 \geq k \geq 0$), then $(p, \gamma, p', \gamma_1 \cdots \gamma_k) \in \Delta$, that is to say the hint of the strategy is valid.
- If $p \in P_1$, then $\forall (p, \gamma, p', \gamma_1 \cdots \gamma_k) \in \Delta$ there exists a unique $(p, a, p', w) \in \Pi$ such that $a = (\gamma, \sigma)$ and $w = (\gamma_1, \sigma_1) \cdots (\gamma_k, \sigma_k)$ ($2 \geq k \geq 0$).

Proof: (Theorem 4.2.2)

Definition of the FSP

The PDGS is given by $\mathcal{G} = (P_0, P_1, \Gamma, \Delta)$, $P = P_0 \uplus P_1$, and Ω .

For every $p, p', q \in P$; $\gamma, \gamma', \eta \in \Gamma$; $m, \ell \in [\max]$; $B, C \in (\mathcal{P}(P))^{\max}$, the FSP has the following vertices:

$$\begin{aligned} & Check(p, \gamma, B, m), \quad Push(B, m, p', \gamma'\eta), \\ & Claim(B, m, p', \gamma'\eta, C), \quad Jump(p, \gamma, B, m, \ell), \quad Win_0(p), \quad Win_1(p), \end{aligned}$$

where Win_σ means immediate win for Player σ , and the following transitions:

$$\begin{aligned} Check(p, \gamma, B, m) &\rightarrow Check(p', \gamma', B, \min(m, \Omega(p'))) && \text{if } (p, \gamma, p', \gamma') \in \Delta, \\ Check(p, \gamma, B, m) &\rightarrow Win_0(p') && \text{if } (p, \gamma, p', \varepsilon) \in \Delta \text{ and } p' \in B_m, \\ Check(p, \gamma, B, m) &\rightarrow Win_1(p') && \text{if } (p, \gamma, p', \varepsilon) \in \Delta \text{ and } p' \notin B_m, \\ Check(p, \gamma, B, m) &\rightarrow Push(B, m, p', \gamma'\eta) && \text{if } (p, \gamma, p', \gamma'\eta) \in \Delta, \\ Push(B, m, p', \gamma'\eta) &\rightarrow Claim(B, m, p', \gamma'\eta, C), \\ Claim(B, m, p', \gamma'\eta, C) &\rightarrow Check(p', \gamma', C, \Omega(p')), \\ Claim(B, m, p', \gamma'\eta, C) &\rightarrow Jump(q, \eta, B, m, \ell) && \text{if } q \in C_\ell, \text{ and} \\ Jump(q, \eta, B, m, \ell) &\rightarrow Check(q, \eta, B, \min(\ell, m, \Omega(q))). \end{aligned}$$

One defines in the FSP the player whose turn is it: $Check(p, \gamma, B, m) \in S_0 \Leftrightarrow p \in P_0$, but $Push(B, m, p', \gamma'\eta) \in S_0$ and $Claim(B, m, p', \gamma'\eta, C) \in S_1$. From other vertices, the players have no alternative: there is a unique successor. One has the following priorities:

$$\begin{aligned} \lambda(Check(p, \gamma, B, m)) &= \Omega(p), \\ \lambda(Jump(q, \gamma, B, m, \ell)) &= \ell, \\ \lambda(Push(B, m, p', \gamma'\eta)) &= \lambda(Claim(B, m, p', \gamma'\eta, C)) = \max - 1. \end{aligned}$$

because the latter are intermediate vertices which should not interfere with the “real”parity condition.

It remains to clarify the situation concerning deadlocks. If the first letter of the stack and the control state do not permit to execute a transition, there is a deadlock in the PDGS as in the corresponding vertex of the FSP. We leave to the reader to choose the convention concerning which player wins in that case.

If one needs a bottom stack symbol (\perp), that cannot be erased and cannot be pushed, one has to care for this explicitly in Γ and Δ . Otherwise when the stack is empty, no transition is possible in our framework of PDGS. We have again to choose a convention for this type of deadlock. It concerns the choice of B in the initial

vertex $Check(p_0, \perp, B, \Omega(p_0))$ of the FSP.

Equivalence between the games: from FSP to PDGS

Suppose that Player 0 has a winning strategy in the FSP from vertex $Check(p_0, \perp, B, \Omega(p_0))$. Since the game graph is finite, and the strategy can be taken positional [EJ91], it is effectively given as a subset of the set of transitions, and denoted $\xrightarrow{str} \subseteq \rightarrow$. We define from it a winning pushdown strategy in the PDGS. This construction is effective.

The strategy automaton is (P_0, P_1, A, Π) , with $A = \Gamma \times \Sigma$. We fix

$$\Sigma = (\mathcal{P}(P))^{max} \times [max].$$

For notational convenience, an element $(\gamma, (B, m))$ of A will be written γBm , and a transition $((p, \gamma Bm), (p', \gamma' B' m')) \in \Pi$ will be written as a prefix rewriting rule $p \gamma Bm \xrightarrow{str} p' \gamma' B' m'$. Similarly $p \gamma Bm \xrightarrow{str} p' \varepsilon$, and $p \gamma Bm \xrightarrow{str} p' \gamma' B' m' \gamma'' B'' m''$.

The initial configuration of the PDGS is $p_0 \perp$, and the one of the FSP is $Check(p_0, \perp, B, \Omega(p_0))$, where B is chosen according to the convention about empty stack (see above). The initial configuration of the strategy is $p_0 \perp B \Omega(p_0)$. From a configuration $p \gamma Bm w$ of the strategy automaton, where $w \in A^*$, the transition in Π is defined as follows:

If $p \in P_0$, then we know that in the FSP Player 0 chooses the next vertex from $Check(p, \gamma, B, m)$ according to \xrightarrow{str} .

- If $Check(p, \gamma, B, m) \xrightarrow{str} Check(p', \gamma', B, \min(m, \Omega(p')))$, then use the transition $p \gamma Bm \xrightarrow{str} p' \gamma' B \min(m, \Omega(p'))$.
- If $Check(p, \gamma, B, m) \xrightarrow{str} Win_0(p')$, then apply $p \gamma Bm \xrightarrow{str} p' \varepsilon$. Of course in the PDGS there is no immediate win, but the game goes on (cf jump move). Moreover it is necessary, in the new top letter $\gamma' B' m'$ of the stack to update m' according to m and $\Omega(p')$, as follows (details are left to the reader): $p \gamma Bm \gamma' B' m' \xrightarrow{str} (p', m) \gamma' B' m' \xrightarrow{str} p' \gamma' B' \min(m, m', \Omega(p'))$.
- If $Check(p, \gamma, B, m) \xrightarrow{str} Push(B, m, p', \gamma' \eta) \xrightarrow{str} Claim(B, m, p', \gamma' \eta, C)$, then apply $p \gamma Bm \xrightarrow{str} p' \gamma' C \Omega(p') \eta Bm$. Of course in the PDGS Player 1 has no opportunity to jump, he must enter the subgame.

If $p \in P_1$, then Player 1 chooses any possible transition in the PDGS, and the Strategy automaton updates its stack according to the winning strategy \xrightarrow{str} of the FSP. More precisely,

- if Player 1 chooses $(p, \gamma, p', \gamma') \in \Delta$, then the strategy executes $p\gamma Bm \xrightarrow{str} p'\gamma'B \min(m, \Omega(p'))$.
- If he chooses $(p, \gamma, p', \varepsilon) \in \Delta$, then the strategy do $p\gamma Bm \xrightarrow{str} p'\varepsilon$, followed by an update of m' .
- If he chooses $(p, \gamma, p', \gamma'\eta) \in \Delta$, then we have to follow \xrightarrow{str} in the FSP, and find C such that $Push(B, m, p', \gamma'\eta) \xrightarrow{str} Claim(B, m, p', \gamma'\eta, C)$. Then $p\gamma Bm \xrightarrow{str} p'\gamma'C\Omega(p')\eta Bm$ is applied.

Because \xrightarrow{str} is winning in the FSP, \xrightarrow{str} is also winning in the PDGS. Moreover using known algorithms to solve the FSP, we have constructed a pushdown strategy which is winning in the PDGS.

From PDGS to FSP

Given a winning strategy in the PDGS, we will define a winning strategy in the FSP. Here a strategy in the PDGS from initial configuration $p_0 \perp = \pi_0$ is a function Str which associates to the prefix $\pi_0 \cdots \pi_n$ of a play a “next move”, *i.e.*, a transition in Δ . We consider a strategy for Player 0, so it is defined if $\pi_n \in V_0$. This function is not necessarily computable, so this part is not effective.

As above, a vertex $Check(p, \gamma, B, m)$ corresponds to a configuration $p\gamma\nu$ of the PDGS. If $p \in P_1$, the PDGS has to follow the move of the FSP in the usual way, whereas if $p \in P_0$, the strategy Str determines the “good” move of the FSP. The only difficult point is the push operation: from $Push(B, m, p', \gamma'\eta)$ Player 0 has to guess a tuple $C \in (\mathcal{P}(P))^{max}$ of sets of possible control states after the next pop. This is well defined if function Str is well defined, although this is a second reason why this part is not effective (even if Str is effective). ■

Corollary 4.2.3 *If there is a winning strategy for Player 0 in the parity game over the pushdown graph, then one can compute a winning pushdown strategy.*

The results in [KV00, Var98] (see Chapter 5) are in some sense stronger. One can deduce from them the winner, and a winning strategy defined by a finite automaton with output. It reads the current configuration and outputs the “next move”. This strategy is positional and can also be executed by a pushdown automaton which store the run of the current configuration on the finite automaton and update it (the automaton reads the configuration from bottom to top).

Here there is no deep reason that the strategy is not positional (it is due to the update of the priority m). It is open whether it is possible to solve the FSP and

find a normal form for the stack of the strategy in such a way that the resulting pushdown strategy is positional.

In the above construction, the FSP has the same number max of priorities as the PDGS, and the number of vertices is exponential in $|P|$ (more precisely, it is in $\mathcal{O}(|\Delta|2^{2 \cdot max \cdot |P|})$). So far the best known algorithms to solve finite state parity game are polynomial in the number of vertices and exponential in the number of priorities. Applied here, we get a solution for parity games over pushdown systems $(P_0, P_1, \Gamma, \Delta)$ which is exponential in $max \cdot |P|$.

Note that the reduction from PDGS to FSP is not symmetric: only Player 0 has to perform claims. For this reason the reduction does not allow to compute directly also the strategy of Player 1 if he wins. Of course another reduction exchanging the roles of Player 0 and Player 1 is possible.

4.3 Example

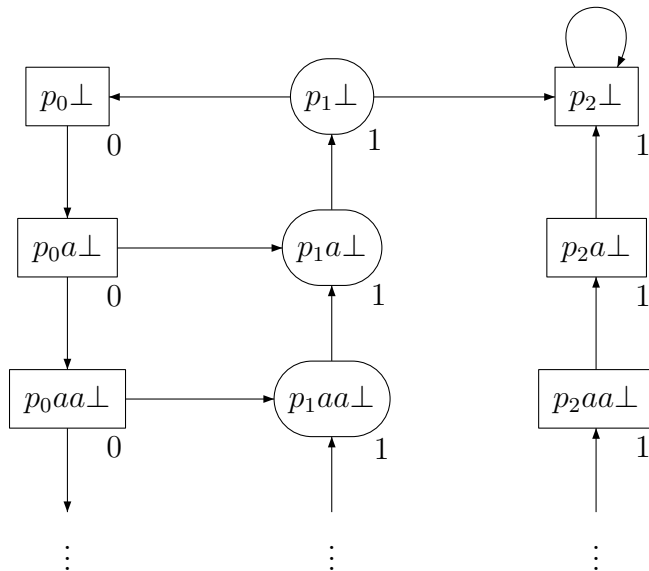


Figure 4.1: Example of a simple pushdown game (Section 4.3)

We present here a simple example of pushdown game to illustrate the previous

section. Let

$$\begin{aligned} \Gamma &= \{a, \perp\}, P_0 = \{p_1\}, P_1 = \{p_0, p_2\}, \\ \Delta &= \{(p_0, \perp, p_0, a\perp), (p_0, a, p_0, aa), (p_0, a, p_1, a), (p_1, a, p_1, \varepsilon), \\ &\quad (p_1, \perp, p_0, \perp), (p_1, \perp, p_2, \perp), (p_2, \perp, p_2, \perp), (p_2, a, p_2, \varepsilon)\}, \\ \Omega(p_0) &= 0, \Omega(p_1) = \Omega(p_2) = 1, \max = 2. \end{aligned}$$

The game graph looks like Figure 4.1. We consider the initial configuration $p_1\perp$. We represent in Figure 4.2 the part of the corresponding FSP that is relevant for Player 0. Namely the solid-arrows define a winning strategy for Player 0, other arrows are dashed. We write B_0B_1 for a tuple (B_0, B_1) in $(\mathcal{P}(P))^2$. The symbol \perp cannot be removed from the stack, so the initial value of the tuple $B \in (\mathcal{P}(P))^2$ is not relevant. We set it to $\emptyset\emptyset$ in the initial vertex $Check(p_1, \perp, \emptyset\emptyset, 0)$. We see

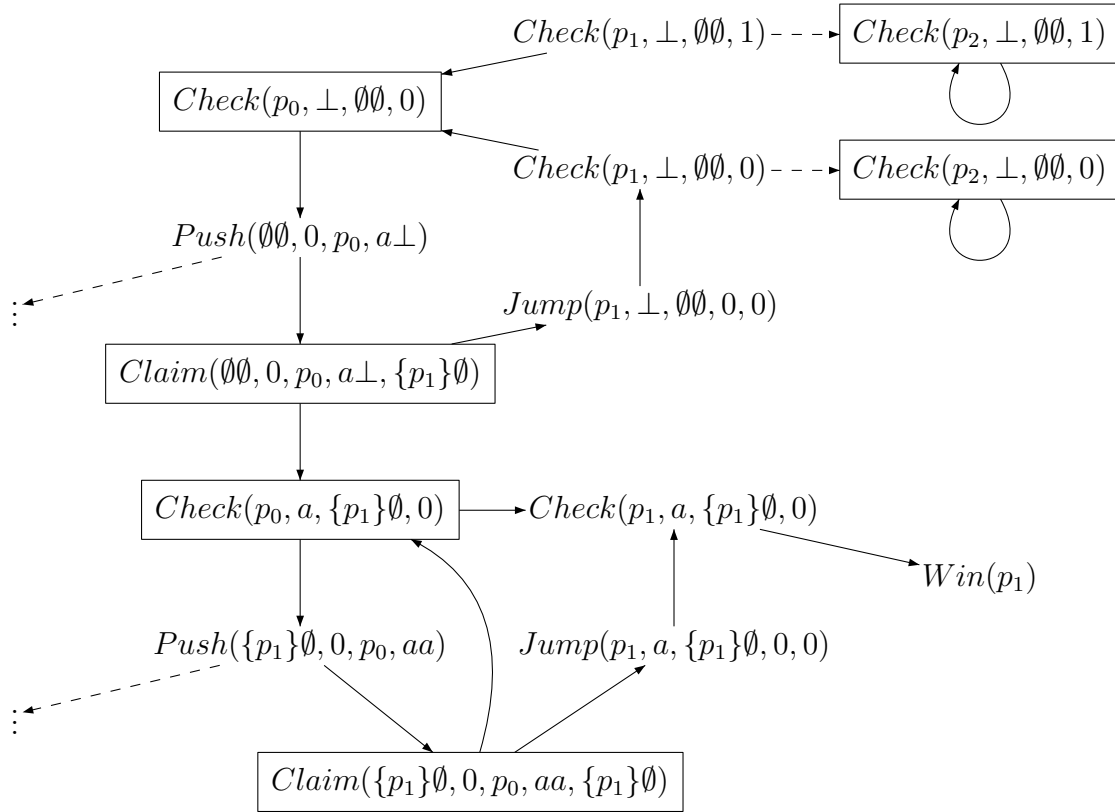


Figure 4.2: FSP corresponding to the PDGS (Section 4.3)

that Player 0 has a winning strategy from $Check(p_1, \perp, \emptyset\emptyset, 0)$, by choosing always

$(\{p_1\}, \emptyset)$ after a *Push* node, and, of course, going from $p_1 \perp$ to $p_0 \perp$. This example is also solved in Section 3.4.3.

4.4 Extension to a Uniform Solution

A deficit of the result of [Wal96b] is that the winner is determined only from the initial position $p_0 \perp$. We give here an algorithm which determines the winner from any position. Moreover we get a global or “symbolic” representation of the whole winning region, which will be proved to be regular (configurations are words over the alphabet $P \cup \Gamma$). One needs a pre-computation to solve the FSP below, e.g. with the algorithm of [VJ00].

We have seen that a configuration $p\gamma\nu$ of the PDGS is represented in the FSP by $Check(p, \gamma, B, m)$ where B “summarizes” information about ν . This can be used more systematically if we know from which configurations $q\nu$ Player 0 can win. For all $B \subseteq P$, we write $[B]^{max} = (B, \dots, B) \in (\mathcal{P}(P))^{max}$.

Algorithm 4.4.1 (uniform solution for parity game on PDGS)

Input: a PDGS $(P_0, P_1, \Gamma, \Delta)$, $P = P_0 \uplus P_1$, and a priority function $\Omega : P \rightarrow [max]$, a configuration $\pi_0 = p\gamma_0\gamma_1 \dots \gamma_n \in P\Gamma^*$

Output: a winning strategy from π_0 , or the answer “ π_0 is not in the winning region of Player 0”

Solve first the FSP corresponding to the PDGS (see proof of Theorem 4.2.2). Determine the winning region W_0 : the set of vertices from which Player 0 has a winning strategy in the FSP, and compute a positional (and uniform) winning strategy on W_0 .

$D_{n+1} := \emptyset$

for $i := n$ **downto** 0 **do**

$D_i := \{q \in P \mid Check(q, \gamma_i, [D_{i+1}]^{max}, \Omega(q)) \in W_0\}$

end for

if $p \notin D_0$ **then**

answer “ π_0 is not in the winning region of Player 0”

else

answer “there is a winning pushdown strategy with initial configuration

$p \gamma_0 [D_1]^{max} \Omega(p) \gamma_1 [D_2]^{max} 0 \dots \gamma_n [D_{n+1}]^{max} 0$, and transitions like in the proof of Theorem 4.2.2.”

end if

In the initialization of D_{n+1} , \emptyset should be replaced by some $D_{n+1} \subseteq P$ if there is another convention about empty stack. More formally, this iterative computation can be transformed into an alternating automaton reading the word $p\gamma_0\gamma_1 \cdots \gamma_n$, where the transitions are defined depending on which vertices are winning in the FSP. This proves that the winning region of the PDGS is regular.

Theorem 4.4.2 *Given a pushdown game with a parity winning condition, one can compute uniformly the winning region of Player 0, which is regular, and a winning pushdown strategy with Algorithm 4.4.1.*

For the proof we observe that the winning condition concerns only the priorities seen infinitely often, and the result of a play does not depend on a finite prefix of it. Once the FSP is solved uniformly, the initialization of the strategy, as well as determining the winner, is in linear time in the length of the configuration, and the computation of the “next step” is in constant time.

Example

We consider the same example as in section 4.3. If we solve completely the FSP, we see that the following nodes are in the winning region of Player 0:

$$\begin{aligned} \text{Check}(p_0, \perp, B_0B_1, 0) & \text{ for all } (B_0, B_1) \in (\mathcal{P}(P))^2, \\ \text{Check}(p_1, \perp, B_0B_1, 1) & \text{ for all } (B_0, B_1) \in (\mathcal{P}(P))^2, \\ \text{Check}(p_0, a, B_0B_1, 0) & \text{ if } p_1 \in B_0, \\ \text{Check}(p_1, a, B_0B_1, 1) & \text{ if } p_1 \in B_1, \\ \text{Check}(p_2, a, B_0B_1, 1) & \text{ if } p_2 \in B_1. \end{aligned}$$

Applying the algorithm to a configuration $\pi_0 = pa \cdots a\perp$, we get $D_{n+1} := \emptyset$ and for all $i \in \{0, n\}$, $D_i = \{p_0, p_1\}$. Then the winning region of Player 0 in the PDGS is $\{p_0, p_1\}a^*\perp$.

4.5 Parity Games on Prefix-Recognizable Graphs

We extend the previous results to a class of graphs where the degree can be unbounded or infinite. Among several equivalent definitions of Prefix-Recognizable Graph (class REC_{RAT} in [Cau96]) we choose the following. Given a finite alphabet Γ , a graph, or set of edges $G \subseteq \Gamma^* \times \Gamma^*$ is a *Prefix-Recognizable Graph* (or *PRG*) if

and only if

$$G = \{uw \leftrightarrow vw \mid u \in U_i, v \in V_i, w \in W_i, 1 \leq i \leq N\},$$

where for all i , $1 \leq i \leq N$, the U_i, V_i, W_i are regular sets over Γ .

Games over PRG are defined in a natural way. In a configuration $x \in \Gamma^+$, the first letter determines the priority and the player whose turn it is. The priority function is $\Omega : \Gamma \rightarrow [\max]$, extended to Γ^+ by $\Omega(ax) = \Omega(a), \forall a \in \Gamma, x \in \Gamma^*$. And we have $\Gamma = \Gamma_0 \uplus \Gamma_1$, $V_0 = \Gamma_0 \Gamma^*$, and $V_1 = \Gamma_1 \Gamma^*$, similarly to the PDGS. A game starting from $\pi_0 \in \Gamma^+$ is defined in the usual way. Again we consider min-parity: Player 0 wins $\pi_0 \pi_1 \dots$ if and only if $\liminf_{k \rightarrow \infty} \Omega(\pi_k)$ is even.

4.5.1 Reduction to Parity Game on Pushdown Graph

We will define a PDGS $(P_0, P_1, \Gamma', \Delta)$ which is equivalent to the PRG in the sense that Player 0 wins the PDGS if and only if he wins the PRG, and a winning strategy in one game can be effectively constructed from a winning strategy in the other game. So this is a stronger condition than needed in our definition of game simulation. The idea of this simulation was also presented in Section 3.8.2 in a more informal way.

Let $\Gamma' = \Gamma \uplus \{\perp\}$. A vertex $ax \in \Gamma^+$ of the PRG ($a \in \Gamma$) is represented by the configuration $t_k^0 ax \perp$, if $k = \Omega(a)$ and $a \in \Gamma_0$, respectively by $t_k^1 ax \perp$, if $k = \Omega(a)$ and $a \in \Gamma_1$ ($t_k^0, t_k^1 \in P$). The idea of the reduction is to decompose the transition $uw \leftrightarrow vw$ of the PRG letter by letter, using intermediate configurations in the PDGS.

Theorem 4.5.1 *Given a PRG \mathcal{G} with parity condition, one can construct in linear time a PDGS \mathcal{G}' such that \mathcal{G} is simulated by \mathcal{G}' , with the stronger condition here that a play over the PRG is mapped to a play over the PDGS preserving the winning condition. Consequently:*

1. *the winner of the parity-PRG from a given configuration is the winner of the parity-PDGS from the corresponding configuration,*
2. *a winning strategy in the parity-PRG can be calculated from a winning strategy in the parity-PDGS.*

Proof: Each regular set is recognized by a (say deterministic, complete) finite automaton: \mathcal{B}_i for U_i , \mathcal{C}_i for \tilde{V}_i , \mathcal{D}_i for W_i . Here \tilde{V}_i is the mirror language of V_i , i.e., \mathcal{C}_i is reading from right to left. We note p_{ij} the states of \mathcal{B}_i , q_{ij} and r_{ij} those of \mathcal{C}_i and \mathcal{D}_i respectively. The corresponding initial states are p_{i0} , q_{i0} , and r_{i0} . We note $p_{ij} \xrightarrow{\mathcal{B}_i} p_{ij'}$ a transition in \mathcal{B}_i labeled by the letter a . In the following j is always

an integer in the *finite* range $[0, NS]$ where NS is the maximal number of states of the automata \mathcal{B}_i , \mathcal{C}_i , and \mathcal{D}_i .

We define now the PDGS that simulates the PRG. The control-states of the PDGS have the same names as the states of the automata \mathcal{B}_i , \mathcal{C}_i , and \mathcal{D}_i , with additional superscript 0 or 1 (i is ranging over $[1, N]$):

$$\begin{aligned} P_0 &= \{p_{ij}^0 \mid 0 \leq j \leq NS\} \cup \{t_k^0 \mid k < \max\} \cup \{s_i^0\}, \\ P_1 &= \{p_{ij}^1 \mid 0 \leq j \leq NS\} \cup \{t_k^1 \mid k < \max\} \cup \{s_i^1\}. \end{aligned}$$

Additional control states t_k^0 and t_k^1 are used to mark the configurations of the PDGS that correspond to vertices of the PRG. States s_i^0, s_i^1 are added for technical reasons.

The transitions rules of the PDGS are the following: for all $a, b \in \Gamma, c \in \Gamma'$,

$$\begin{aligned} t_k^0 c \hookrightarrow p_{i0}^0 c & \quad (\text{Player 0 chooses to use in the PRG a transition of type } i: \\ & \quad u_i w_i \hookrightarrow v_i w_i, u_i \in U_i, v_i \in V_i, w_i \in W_i), \\ t_k^1 c \hookrightarrow p_{i0}^1 c & \quad (\text{similarly for Player 1}). \end{aligned}$$

Then for all $\sigma \in \{0, 1\}$, the opponent of σ is denoted $\bar{\sigma}$, and we have

$$\begin{aligned} p_{ij}^\sigma a \hookrightarrow p_{ij'}^\sigma & \text{ if } p_{ij} \xrightarrow{\mathcal{B}_i} p_{ij'} & (\text{“reading” of } u_i), \\ p_{ij}^\sigma c \hookrightarrow s_i^{\bar{\sigma}} c & \text{ if } p_{ij} \text{ is a final state of } \mathcal{B}_i & (\text{Player } \sigma \text{ decides that the word } u_i \\ & \text{ends here, and asks the opponent for agreement}). \end{aligned}$$

$$s_i^{\bar{\sigma}} c \hookrightarrow r_{i0}^\sigma c \quad (\text{the opponent wants to verify that the rest of the stack is really in } W_i, \text{ because he thinks that this is not the case})$$

$$r_{ij}^\sigma a \hookrightarrow r_{ij'}^\sigma \text{ if } r_{ij} \xrightarrow{\mathcal{D}_i} r_{ij'} \quad (\text{“reading” of } w_i, \text{ then:})$$

$r_{ij}^\sigma \perp$ is immediate lost for σ if r_{ij}^σ is not final in \mathcal{D}_i ,

and immediate win for σ if r_{ij}^σ is final in \mathcal{D}_i ,

$$s_i^{\bar{\sigma}} c \hookrightarrow q_{i0}^\sigma c \quad (\text{otherwise, the opponent is trusting Player } \sigma, \text{ and lets him continue),}$$

$$q_{ij}^\sigma c \hookrightarrow q_{ij'}^\sigma bc \text{ if } q_{ij} \xrightarrow{\mathcal{C}_i} q_{ij'} \quad (\text{“writing” of } v_i, \text{ chosen by Player } \sigma),$$

$$q_{ij}^\sigma a \hookrightarrow t_k^0 a \text{ if } q_{ij} \text{ is a final state of } \mathcal{C}_i, a \in \Gamma_0 \text{ and } k = \Omega(a)$$

(Player σ chooses that v_i ends here),

$$q_{ij}^\sigma a \hookrightarrow t_k^1 a \text{ if } q_{ij} \text{ is a final state of } \mathcal{C}_i, a \in \Gamma_1 \text{ and } k = \Omega(a) \quad (\text{similarly}).$$

Note that the control state t_k^0 is redundant with the first letter.

Of course the priority of t_k^σ is $\Omega(t_k^\sigma) = k$. Given $x \in \Gamma^+$, it is clear that: $x \hookrightarrow y$ in the PRG ($y \in \Gamma^*$) if and only if from the corresponding configuration $t_k^\sigma x \perp$, Player σ can reach the configuration $t_{k'}^{\sigma'} y \perp$ corresponding to y or wins immediately (if the opponent $\bar{\sigma}$ thinks that σ wants to violate the transition rule).

Now the unique deficit of this construction is that Player σ can stay forever in the intermediate states q_{ij}^σ , pushing infinitely many new letters onto the stack. To avoid this unfair behavior, which does not correspond to a real transition of the PRG, we assign to these intermediate states a priority that is losing for Player σ . Suppose that the priority function Ω ranges from 0 to $2c$, $c \geq 0$, then we define the following priorities:

$$\begin{aligned}\Omega(p_{ij}^0) &= \Omega(q_{ij}^0) = \Omega(r_{ij}^0) = \Omega(s_i^0) = 2c + 1, \\ \Omega(p_{ij}^1) &= \Omega(q_{ij}^1) = \Omega(r_{ij}^1) = \Omega(s_i^1) = 2c.\end{aligned}$$

These priorities are greater than the normal priorities, so they have no influence on the winning condition of the “real” game. One takes $2c + 1$ for Player 0, while Player 0 wants an even number; so he can’t win by staying in those intermediate states. Similarly for Player 1. And dually if the maximal priority of Ω is $2c + 1$, then the new priorities are $2c + 1$ and $2c + 2$. ■

The reduction presented here will be extended in Section 5.3 to other classes of graphs, and is not so far from the result of [Cau03b] (Proposition 4.2), that the prefix-recognizable graphs are obtained from the pushdown graphs by ε -closure.

Note that it remains open how to extend the techniques of [KV00, Var98], based on tree automata, also to a uniform solution.

Example 4.5.2 *Let $\Gamma = \{a, b\}$, we consider the following PRG:*

$$G = (a \leftrightarrow \varepsilon)a^+ \cup (a \leftrightarrow b)\varepsilon \cup (b \leftrightarrow a^+)\varepsilon,$$

written here in the form $(U_1 \leftrightarrow V_1)W_1 \cup (U_2 \leftrightarrow V_2)W_2 \cup (U_3 \leftrightarrow V_3)W_3$, with $N = 3$. Let $\max = 2$, $\Omega(a) = 1$, $\Omega(b) = 0$, $\Gamma_0 = \{a\}$, $\Gamma_1 = \{b\}$. The game graph is pictured in Figure 4.3.

According to the above construction, a vertex a^i is represented in the PDGS by $t_1^0 a^i \perp$, and b is represented by $t_0^1 b \perp$. The automaton recognizing U_i, V_i and W_i are very simple, we do not define them explicitly. Figure 4.4 represents a part of the graph of the corresponding PDGS.

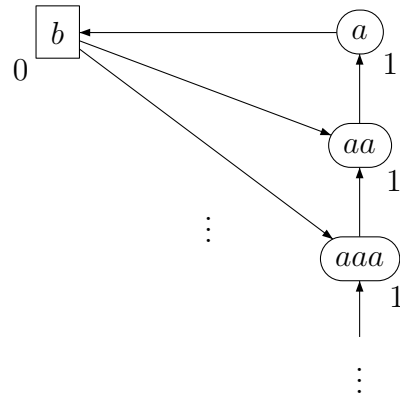


Figure 4.3: Example (4.5.2) of a game on a Prefix-recognizable graph

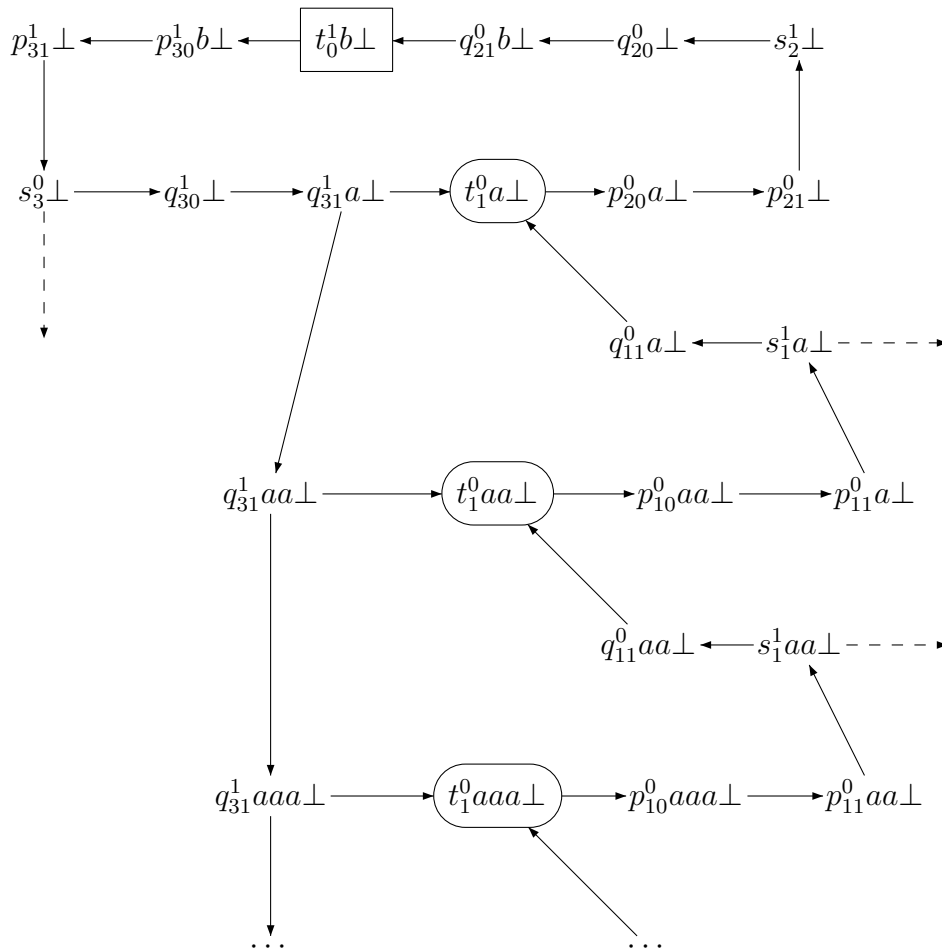


Figure 4.4: Graph of the corresponding PDGS (Example 4.5.2)

Chapter 5

Parity Games over Caucal Graphs

Until now we have considered pushdown graphs, and a simple extension of it: prefix-recognizable graphs. In the present chapter we consider a wider class of graphs: transition graphs of higher order pushdown automata. We restrict ourselves to parity games, for which the technique of game simulation is suited. As usual we introduce higher order pushdown automata here as game graphs, where the player and the priority of a configuration are determined by the control state. We consider also the infinite hierarchy of graphs defined recently by Caucal [Cau02] from the finite trees using inverse mapping and unfolding. To solve these new games, we have two main results: via game-simulation we show how to translate a game on a higher order pushdown automaton to a kind of model-checking game on a Caucal graph; then we reduce such a game to a game on a graph from a lower level of the hierarchy and finally to a parity game on a finite graph, which gives an effective solution. It is then possible to reconstruct a winning strategy for the original game. As far as we know this is the first result in this direction. So far only the decidability of MSO-properties of these graphs was known [Cau02, KNU02].

In the next section we define the different models of graphs and introduce automata theoretic and logical prerequisites: higher order pushdown automata, the hierarchy of graphs of [Cau02], the graph automaton of [KV00] and the μ -calculus. Then we present in terms of game-simulation the reduction from higher order pushdown automata to the Caucal graphs and vice versa. In Section 5.3 we show that a game on a Caucal graph can be reduced to an equivalent game on a graph of lower level. For this we use a generalization of ideas from [Var98] to trees of infinite degree: the construction of an alternating one-way tree automaton equivalent to a given two-way alternating automaton. The main result that we use without proof is the positional (memoryless) determinacy of parity games of [EJ91]: from

any configuration one of the players has a positional winning strategy.

5.1 The Models

5.1.1 Higher Order Pushdown Game System

The pushdown automata have good properties but they are quite restricted. To model more general processes, other classes of machines have been proposed in the literature, like higher order pushdown automata. Note that when used as acceptors, they can recognize non context-free languages. They use control states in a similar way to pushdown automata, but their stack is replaced by a more sophisticated store, allowing a kind of copy of the content, to be defined next.

We recall the definition from [KNU02] (which is equivalent to the one from [Eng83]), where we slightly change the terminology. A *level 1 store* (or *1-store*) over an alphabet Γ is an arbitrary sequence $[a_1, \dots, a_\ell]$ of elements of Γ , with $\ell \geq 0$. A *level n store* (or *n -store*), for $n \geq 2$, is a sequence $[s_1, \dots, s_\ell]$ of $(n-1)$ -stores, where $\ell \geq 0$. We allow a store to be empty. The following operations can be performed on 1-store:

$$\begin{aligned} \text{push}_1^a([a_1, \dots, a_{\ell-1}, a_\ell]) &:= [a_1, \dots, a_{\ell-1}, a_\ell, a] \text{ for all } a \in \Gamma, \\ \text{pop}_1([a_1, \dots, a_{\ell-1}, a_\ell]) &:= [a_1, \dots, a_{\ell-1}], \\ \text{top}([a_1, \dots, a_{\ell-1}, a_\ell]) &:= a_\ell. \end{aligned}$$

If $[s_1, \dots, s_\ell]$ is a store of level $n > 1$, the following operations are possible:

$$\begin{aligned} \text{push}_n([s_1, \dots, s_{\ell-1}, s_\ell]) &:= [s_1, \dots, s_\ell, s_\ell], \\ \text{push}_k([s_1, \dots, s_{\ell-1}, s_\ell]) &:= [s_1, \dots, \text{push}_k(s_\ell)] \text{ if } 2 \leq k < n, \\ \text{push}_1^a([s_1, \dots, s_{\ell-1}, s_\ell]) &:= [s_1, \dots, \text{push}_1^a(s_\ell)] \text{ for all } a \in \Gamma, \\ \text{pop}_n([s_1, \dots, s_{\ell-1}, s_\ell]) &:= [s_1, \dots, s_{\ell-1}], \\ \text{pop}_k([s_1, \dots, s_{\ell-1}, s_\ell]) &:= [s_1, \dots, s_{\ell-1}, \text{pop}_k(s_\ell)] \text{ if } 1 \leq k < n, \\ \text{top}([s_1, \dots, s_{\ell-1}, s_\ell]) &:= \text{top}(s_\ell). \end{aligned}$$

So an operation push_k , $k > 1$ realizes a copy of the top store of level k , and allows to have more expressive power than pushdown graphs. The operation pop_k is undefined on a store whose top store of level k is empty. Similarly top is undefined on a store whose top 1-store is empty. Given Γ and n , the set Op_n of operations (on a store) of level n consists of:

$$\text{push}_k \text{ for all } 2 \leq k \leq n, \text{ push}_1^a \text{ for all } a \in \Gamma, \text{ and } \text{pop}_k \text{ for all } 1 \leq k \leq n.$$

A *Higher Order Pushdown Game System* of level n (or n -HPDGS) is a tuple $H = (P_0, P_1, \Gamma, \Delta)$ where

- $P = P_0 \uplus P_1$ is the partitioned finite set of control locations, where P_i indicates the game positions of Player i ,
- Γ the finite store alphabet, and
- $\Delta \subseteq P \times \Gamma \times P \times Op_n$ the finite set of (unlabeled) transition rules.

A *configuration* of an n -HPDGS H is a pair (p, s) where $p \in P$ and s is an n -store. The set of n -stores is denoted \mathcal{S}_n . We do not consider HPDGS as accepting devices, hence there is no input alphabet. A HPDGS $H = (P_0, P_1, \Gamma, \Delta)$ defines a *transition graph* (V, E) , where $V = \{(p, s) : p \in P, s \in \mathcal{S}_n\}$ is the set of all configurations, and

$$(p, s)E(p', s') \iff \exists(p, a, p', \sigma) \in \Delta : top(s) = a \text{ and } s' = \sigma(s) .$$

Note that if the top 1-store is empty, no transition is possible. If necessary one can add a “bottom store symbol” $\perp \in \Gamma$ and define explicitly the corresponding transitions, such that it cannot be erased.

To define a parity game on the graph of a HPDGS, we assign a priority to each control state, and we consider an initial configuration: a *parity game structure on a HPDGS* H is a tuple $\mathcal{G} = (H, \Omega, s_0)$, where $\Omega : P \rightarrow [max]$ is a priority function, and $s_0 \in V$. This extends naturally to a priority function defined on the set of configurations: with the notations of Section 4.1, $V_0 = P_0 \times \mathcal{S}_n$, $V_1 = P_1 \times \mathcal{S}_n$, $\Omega((p, s)) = \Omega(p)$, and E is defined above.

5.1.2 Caucal Hierarchy

Another track for defining new graphs is to start with the fundamental result of Rabin [Rab69] that the MSO theory of the complete infinite binary tree is decidable and to try to extend it to more general classes of infinite graphs. In the setting presented here the nodes of the graphs are not concrete objects like words but form an abstract set, and operations are defined on this external representation.

We recall the definitions from [Cau02]. Let L be a countable set of symbols for labeling arcs. A graph is here simple, oriented and arc labeled in a finite subset of L . Formally, a *graph* G is a subset of $V \times L \times V$, where V is an arbitrary set and

such that its *label set*

$$\begin{aligned} L_G &:= \{a \in L \mid \exists s, t : (s, a, t) \in G\} && \text{is finite, but its vertex set} \\ V_G &:= \{s \mid \exists a, t : (s, a, t) \in G \vee (t, a, s) \in G\} && \text{is finite or countable.} \end{aligned}$$

We write also $t \xrightarrow[G]{a} s$ (or $t \xrightarrow{a} s$) for $(t, a, s) \in G$.

A *finite graph* is a graph whose vertex set is finite. A *tree* is a graph where each vertex has at most one predecessor, the unique root has no predecessor, and each vertex is accessible from the root. A *vertex labeled tree* is a tree, with a labeling function associating to each node a letter from a finite alphabet. The *unfolding* of a graph G is the following forest (set of trees):

$$Unf(G) := \{ws \xrightarrow{a} wsat : w \in (V_G \cdot L_G)^* \wedge s \xrightarrow[G]{a} t\} .$$

The unfolding $Unf(G, s)$ of a graph G from a vertex s is the restriction of $Unf(G)$ to the vertices accessible from s . Given a set of graphs \mathcal{H} , $Unf(\mathcal{H})$ is the set of graphs obtained by unfolding from the graphs of \mathcal{H} . Inverse arcs are introduced to move up and down in trees: we have a set $\bar{L} := \{\bar{a} \mid a \in L\}$ of fresh symbols in bijection with L . By definition, we have an arc (s, \bar{a}, t) if and only if (t, a, s) is an arc of G . Note that in a tree there is at most one inverse arc from a given node.

In the usual way $s \xrightarrow[G]{w}^* t$ means that there is a *path* from s to t labeled by the word w . A *substitution* is a relation $h \subseteq L \times (L \cup \bar{L})^*$. It has finite domain if $Dom(h) := \{a \mid h(a) \neq \emptyset\}$ is finite. In this case, the inverse mapping of any graph G by h is

$$h^{-1}(G) = \{s \xrightarrow{a} t \mid \exists w \in h(a) : s \xrightarrow[G]{w}^* t\} .$$

The mapping h is rational if $h(a)$ is rational for every $a \in L$. Given a set of graphs \mathcal{H} , $Rat^{-1}(\mathcal{H})$ is the set of graphs obtained by inverse rational mapping from the graphs of \mathcal{H} . Let $Tree_0$ be the set of finite trees. The *Caucal Hierarchy* is defined in the following way:

$$\begin{aligned} Graph_n &:= Rat^{-1}(Tree_n) , \\ Tree_{n+1} &:= Unf(Graph_n) . \end{aligned}$$

Here $Graph_0$ is the set of finite graphs, $Tree_1$ is the set of regular trees of finite degree, $Graph_1$ is the set of prefix-recognizable graphs [Cau96] and $Tree_2$ is the set of algebraic trees. The other levels are mostly unknown.

Theorem 5.1.1 ([Cau02]) $\bigcup_{n \geq 0} Graph_n$ is a family of graphs having a decidable monadic second order theory.

As a corollary, μ -calculus model-checking on these graphs is decidable, and one can determine the winner of a parity game. But this result of decidability in [Cau02] relies on the results from [Cau96, CW98, Wal96a] whereas for the restricted framework of games we give here a direct algorithmic construction for determining the winner and a winning strategy.

5.1.3 Graph Automaton

So far there is no natural definition of game on a Caucal graph without extra machinery. The reason is that it is difficult to give a partition of the set of nodes of a Caucal graph into nodes of Player 0 and Player 1, and to give a “semantic” to the nodes, like being in the goal set, having a priority . . . But it is natural to check properties of a Caucal graph using graph automata running on it, and this will allow to build a bridge between HPDGS and Caucal graphs.

An alternating parity graph automaton, or *graph automaton* for short, as defined in [KV00] is a tuple $\mathcal{A} = (Q, W, \delta, q_0, \Omega)$ where

- Q is a finite set of states,
- W is a *finite* set of edge labels,
- δ is the transition function to be defined below,
- $q_0 \in Q$ is the initial state,
- $\Omega : Q \rightarrow [max]$ is a priority function defining the acceptance condition: the minimal priority appearing infinitely often should be even.

Let $next(W) = \{\varepsilon\} \cup \bigcup_{a \in W} \{[a], \langle a \rangle\}$, and $\mathcal{B}^+(next(W) \times Q)$ be the set of positive Boolean formulas built from the atoms in $next(W) \times Q$. The transition function is of the form $\delta : Q \rightarrow \mathcal{B}^+(next(W) \times Q)$. In the case of graphs, we will consider $W \subseteq L$, whereas in the case of trees, we will allow $W \subseteq L \cup \bar{L}$.

A run of a graph automaton $\mathcal{A} = (Q, W, \delta, q_0, \Omega)$ over a graph $G \subseteq V \times L \times V$ from a vertex $s_0 \in V$ is a labeled tree $\langle T_r, r \rangle$ in which every node is labeled by an element of $V \times Q$. This tree is like the unfolding of the product of the automaton and the graph. A node in T_r , labeled by (s, q) , describes a “copy” of the automaton that is in state q and is situated at the vertex s of G . Note that many nodes of T_r can correspond to the same vertex of G , because the automaton can come back to a previously visited vertex and because of alternation. The label of a node and its successors have to satisfy the transition function. Formally, a run $\langle T_r, r \rangle$

is a Σ_r -vertex labeled tree, where $\Sigma_r := V \times Q$ and $\langle T_r, r \rangle$ satisfies the following conditions:

- $r(t_0) = (s_0, q_0)$ where t_0 is the root of T_r .
- Consider $y \in T_r$ with $r(y) = (s, q)$ and $\delta(q) = \theta$. Then there is a (possibly empty) set $Y \subseteq \text{next}(W) \times Q$, such that Y satisfies θ , and for all $\langle d, q' \rangle \in Y$, the following hold:
 - If $d = \varepsilon$ then there exists a successor y' of y in T_r such that $r(y') = (s, q')$.
 - If $d = \langle a \rangle$ then there exists a successor y' of y in T_r , and a vertex s' such that $s \xrightarrow{a}_G s'$ and $r(y') = (s', q')$.
 - If $d = [a]$ then for each vertex s' such that $s \xrightarrow{a}_G s'$, there exists a successor y' of y in T_r such that $r(y') = (s', q')$.

The priority of a node $y \in T_r$, with $r(y) = (s, q)$, is $\Omega(q)$. A run is accepting if it satisfies the parity condition: along each infinite branch of T_r , the minimal priority appearing infinitely often is even.

When G is a tree, \mathcal{A} is like an alternating two-way parity automaton of [KV00], because it can go up and down, but here the degree of the tree can be infinite. It is more general than the model of [GW99] which cannot distinguish between son and parent node. For the proofs we will also consider a tree automaton (defined as a graph automaton) that “reads” the labels of the vertices.

A graph $G \subseteq V \times L \times V$ with “initial” vertex $s_0 \in V$ and a graph automaton \mathcal{A} over G , where $W = L_G$, define a parity game denoted by (G, \mathcal{A}) . The configurations of the game are pairs $(s, q) \in V \times Q$, the initial configuration is (s_0, q_0) . A game is played as follows: from a configuration $(s, q) \in V \times Q$, Player 0 chooses a set $Y \subseteq \text{next}(W) \times Q$, such that Y satisfies $\delta(q)$. Then Player 1 chooses an atom $\langle d, q' \rangle \in Y$, and

- If $d = \varepsilon$ then the new configuration of the game is (s, q') .
- If $d = \langle a \rangle$ then Player 0 chooses a vertex s' such that $s \xrightarrow{a}_G s'$ and the new configuration of the game is (s', q') .
- If $d = [a]$ then Player 1 chooses a vertex s' such that $s \xrightarrow{a}_G s'$ and the new configuration of the game is (s', q') .

A play from s_0 is a finite or infinite sequence of configurations $(s_0, q_0), (s_1, q_1), (s_2, q_2), \dots$. If the play is finite because at some point there is no Y satisfying $\delta(q)$, or Player 0

cannot find a vertex s' , then he loses immediately. If the play is finite because at some point Y is empty (and satisfies $\delta(q)$), or Player 1 cannot find a vertex s' , then he loses immediately. If the play is infinite then we consider the parity condition:

Player 0 wins if and only if $\liminf_{i \rightarrow \infty} \Omega(q_i)$ is even.

That is, if the minimal priority appearing infinitely often is even.

Accordingly we will consider (G, \mathcal{A}) as a game structure. It is well known that a run is a strategy for Player 0, and an accepting run is a winning strategy for Player 0, see e.g. [GTW02, ch. 4].

5.1.4 The Modal μ -calculus

The μ -calculus is a modal logic augmented with least and greatest fixpoint operators, see [Koz83]. We present here the syntax and semantics following [KV00]. Given a finite set W of actions and a finite set Var of variables, the set L_μ of μ -calculus formulas (in positive normal form) over W and Var is defined inductively as the smallest set containing the following:

- the symbols \perp and \top ,
- the variable y for all $y \in \text{Var}$,
- $\varphi_1 \wedge \varphi_2$ and $\varphi_1 \vee \varphi_2$ if φ_1 and φ_2 are in L_μ ,
- $[a]\varphi$ and $\langle a \rangle \varphi$ for all $a \in W$ and $\varphi \in L_\mu$,
- $\mu y. \varphi$ and $\nu y. \varphi$ for all $y \in \text{Var}$ and $\varphi \in L_\mu$.

A variable y is bound if it is in the scope of a fixed-point operator μy or νy . A sentence is a formula that contains no free variables from Var . Like in Section 2.3 for MSO we define the semantics of the μ -calculus with respect to *transition systems* (directed graphs) with edge labels. A transition system G is composed of

- a vertex set V ,
- a finite alphabet T for labeling transitions, such that $\forall a \in T : R_a \subseteq V \times V$ is a transition relation.

We need also a valuation $\Lambda : \text{Var} \rightarrow \mathcal{P}(V)$ for the free variables. Each formula φ and valuation Λ define a set $\varphi^G(\Lambda)$ of states of G that satisfy the formula. For

a valuation Λ , a variable $y \in \text{Var}$ and a set $V' \subseteq V$, we denote by $\Lambda[y \leftarrow V']$ the valuation obtained from Λ by assigning V' to y . The mapping φ^G is defined inductively as follows:

- $\perp^G(\Lambda) = \emptyset$ and $\top^G(\Lambda) = V$,
- for all $y \in \text{Var}$ we have $y^G(\Lambda) = \Lambda(y)$,
- $(\varphi_1 \wedge \varphi_2)^G(\Lambda) = \varphi_1^G(\Lambda) \cap \varphi_2^G(\Lambda)$,
- $(\varphi_1 \vee \varphi_2)^G(\Lambda) = \varphi_1^G(\Lambda) \cup \varphi_2^G(\Lambda)$,
- $([a]\varphi)^G(\Lambda) = \{v \in V : \text{for all } v' \text{ such that } vR_a v', \text{ we have } v' \in \varphi^G(\Lambda)\}$,
- $(\langle a \rangle \varphi)^G(\Lambda) = \{v \in V : \text{there exists } v' \text{ such that } vR_a v' \text{ and } v' \in \varphi^G(\Lambda)\}$,
- $(\mu y.\varphi)^G(\Lambda) = \bigcap \{V' \subseteq V : \varphi^G(\Lambda[y \leftarrow V']) \subseteq V'\}$,
- $(\nu y.\varphi)^G(\Lambda) = \bigcup \{V' \subseteq V : V' \subseteq \varphi^G(\Lambda[y \leftarrow V'])\}$,

So $(\mu y.\varphi)^G(\Lambda)$ is the least fixed-point of the function $V \mapsto \varphi^G(\Lambda[y \leftarrow V])$, and $(\nu y.\varphi)^G(\Lambda)$ is the greatest fixed-point of this function. The standard translation from μ -calculus to MSO uses this property on set of vertices, see Section 2.3 and [GTW02, ch. 14]. By this definition $\varphi^G(\Lambda)$ depends only on the valuation of free variables in φ and no valuation is required for a sentence. To model-check a formula with respect to a transition system we will transform it into an equivalent graph automaton in the sense that the graph automaton has an accepting run from a given vertex if and only if the formula is true at this vertex [EJS93]. We follow the presentation of [Wil01]. According to [Wil01] this transformation is easy and natural but the proof is more difficult.

Given a formula φ we construct an (alternating) graph automaton \mathcal{A}_φ as follows. The state space of \mathcal{A}_φ is (isomorph to) the set of subformulas of φ and the formula φ itself is the initial state. We assume φ is a sentence. To avoid ambiguity we require that each variable y appearing in φ is quantified at most once and all occurrences of y are in the scope of this quantification. So a given variable y can be associated to its fixed-point operator, that is, to its subformula $\eta y.\psi$, where η denotes μ or ν . In this case we denote the formula $\eta y.\psi$ by φ_y . Given $\varphi \in L_\mu$ the graph automaton $\mathcal{A}_\varphi = (Q, W, \delta, q_0, \Omega)$ is defined by

- Q is the set which contains for each subformula ψ of φ (including φ itself) a state denoted $\widehat{\psi}$,

- the initial state is $\widehat{\varphi}$,
- the transition relation is defined by
 - $\delta(\widehat{\perp}) = \text{false}$, $\delta(\widehat{\top}) = \text{true}$,
 - $\delta(\widehat{y}) = \varphi_y$,
 - $\delta(\widehat{\varphi_1 \wedge \varphi_2}) = \widehat{\varphi_1} \wedge \widehat{\varphi_2}$, $\delta(\widehat{\varphi_1 \vee \varphi_2}) = \widehat{\varphi_1} \vee \widehat{\varphi_2}$,
 - $\delta(\widehat{[a]\psi}) = \langle [a], \widehat{\psi} \rangle$, $\delta(\widehat{\langle a \rangle \psi}) = \langle \langle a \rangle, \widehat{\psi} \rangle$,
 - $\delta(\widehat{\mu y.\psi}) = \widehat{\psi}$, $\delta(\widehat{\nu y.\psi}) = \widehat{\psi}$,
- and the priority function is defined as follows. It is relevant only in states \widehat{y} , so for other states we choose the greatest priority. We want to associate to each variable y a priority $\Omega(\widehat{y})$ in such a way that if y comes from a ν -fixpoint ($\nu y.\psi$) then the priority is even and if y comes from a μ -fixpoint ($\mu y.\psi$) then the priority is odd. Moreover it is required that if z occurs free in φ_y (meaning that φ_y is a subformula of φ_z) then $\Omega(\widehat{z}) \leq \Omega(\widehat{y})$.

Translating again to a parity game, one obtain a *model-checking game*: Player 0 wants to prove that the formula is true and Player 1 has to challenge this. Equivalently Player 0 wants to find an accepting run and Player 1 wants to refute it. Player 0 has a winning strategy if and only if the formula is true. The idea is that Player 0 is the *existential player* and makes the choices associated to disjunction (\vee) and diamond ($\langle a \rangle$). Player 1 is the *universal player* and makes the choices associated to conjunction (\wedge) and box ($[a]$). The fixed-point operators are like recursive procedures (see $\delta(\widehat{y}) = \varphi_y$) but Player 0 should not use this self loop infinitely often for a μ -fixed-point because of the priority.

5.2 Game-Simulation Between HPDGS and Caucal Graphs

In this section we show a game theoretical equivalence between graphs defined in terms of graph transformations, where the vertex set is “abstract” — the Caucal graphs — and a presentation based on rewriting of “concrete” nodes — the Higher Order Pushdown Game System.

During the writing of this thesis, this result has been improved by Arnaud Carayol and Stefan Wöhrle in [CW03]. They showed a tight relation between

HPDGS and Caucal graphs, in particular each Caucal graph is obtained from a HPDGS by ε -closure, and each transition graph of a HPDGS is a graph of the Caucal hierarchy of the same level.

5.2.1 From HPDGS to Caucal Graphs

Theorem 5.2.1 *Given a game structure \mathcal{G} on a HPDGS H of level n , one can construct a graph automaton \mathcal{A} and a tree $T \in Tree_n$ such that \mathcal{G} is game-simulated by (T, \mathcal{A}) .*

Proof: We describe the construction for $n = 1, 2$ and 3 before we give the generalization. Let $\mathcal{G} = (H, \Omega, s_0)$, $H = (P_0, P_1, \Gamma, \Delta)$ of level n .

Case $n = 1$. The idea here is similar to that of [KV00] and [Cau96].

Let T_1 be the complete Γ -tree. It is the unfolding of a finite graph with a unique vertex and so $T_1 \in Tree_1$. See an example in Figures 5.1 and 5.2 where $\Gamma = \{a, b\}$.

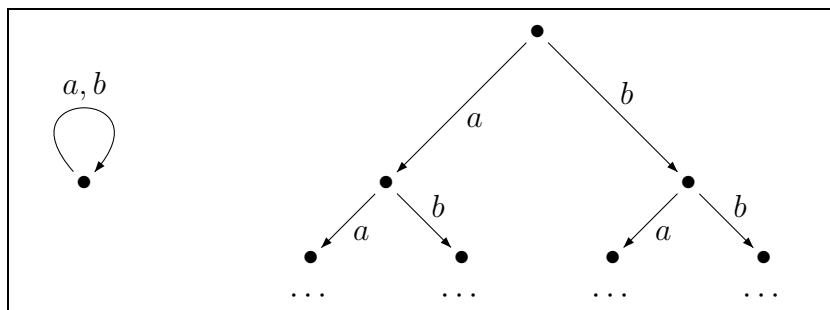


Figure 5.1: The complete $\{a, b\}$ -tree T_1 obtained by unfolding of a finite graph

This tree is isomorphic to Γ^* in the sense that each node is associated to the label of the path from the root to it (we write the store from bottom to top, so we consider suffix rewriting in the application of the rules). It is easy to simulate a 1-store inside this tree: each node corresponds to a word, which is a store content. Intuitively the effect of a transition $(p, a, p', push_1^b)$ on the store is simulated over T_1 by a path $\bar{a}ab$. Formally the state space of \mathcal{A} is $Q = P \times \Gamma^{\leq 2}$, where a state (p, ε) on a node $v \in T_1$ represents a configuration $(p, [v])$ of the HPDGS (by abuse v is associated to a word of Γ^*), whereas the states (p, x) , $x \neq \varepsilon$ are intermediate states that simulate the behavior of the store. From these intermediate states, the transition is somehow “deterministic”: $\forall a \in \Gamma, x \in \{\varepsilon\} \cup \Gamma$:

$$\delta((p, ax)) = \langle \langle a \rangle, (p, x) \rangle .$$

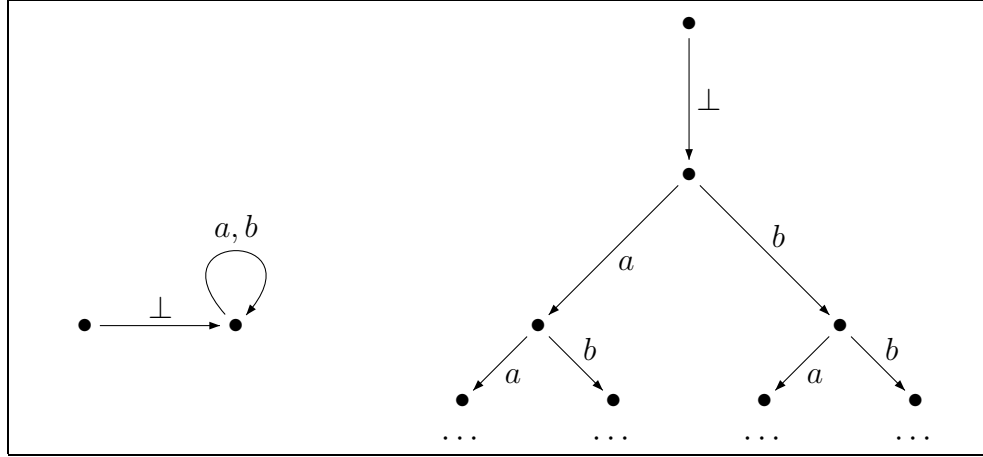


Figure 5.2: The same with a “bottom stack symbol”

Note that here (on T_1), if $a \in \Gamma \subseteq L$ the “actions” $\langle a \rangle$ and $[a]$ are equivalent because there is exactly one a -successor from each node. From the states (p, ε) the corresponding player has to choose the move:

$$\begin{aligned} \text{if } p \in P_0 \text{ then } \delta((p, \varepsilon)) &= \bigvee_{(p,a,p',pop_1) \in \Delta} \langle \bar{a} \rangle, (p', \varepsilon) \quad \vee \quad \bigvee_{(p,a,p',push_1^b) \in \Delta} \langle \bar{a} \rangle, (p', ab) \quad , \\ \text{if } p \in P_1 \text{ then } \delta((p, \varepsilon)) &= \bigwedge_{(p,a,p',pop_1) \in \Delta} [\bar{a}], (p', \varepsilon) \quad \wedge \quad \bigwedge_{(p,a,p',push_1^b) \in \Delta} [\bar{a}], (p', ab) \quad . \end{aligned}$$

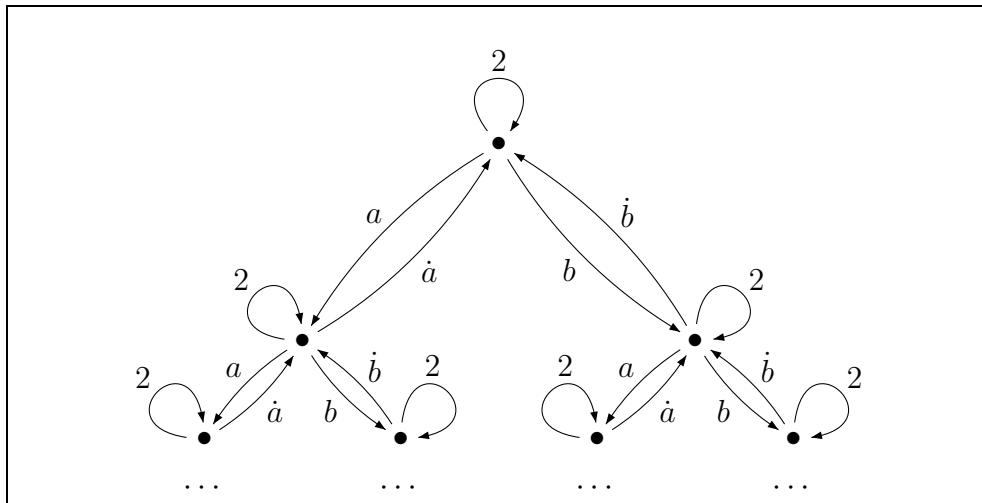
Here it is important to differentiate $\langle \bar{a} \rangle$ and $[\bar{a}]$, because one of the players has to find a transition. We see again that the convention is satisfied: when the play is in a deadlock, the player who should play loses immediately. To complete the construction it is necessary to make \mathcal{A} go to the node and state corresponding to s_0 at the beginning of his run. This is easy using additional states.

Case $n = 2$. For each letter $a \in \Gamma$, we assume that we have a fresh symbol \dot{a} in L . We define the graph $G_1 \in Graph_1$ from the tree T_1 :

$$G_1 = h_1^{-1}(T_1) ,$$

where the (finite) substitution h_1 is the following:

$$\begin{aligned} h_1(a) &= a \text{ for all } a \in \Gamma , & h_1(2) &= \varepsilon , \\ h_1(\dot{a}) &= \bar{a} \text{ for all } a \in \Gamma . \end{aligned}$$

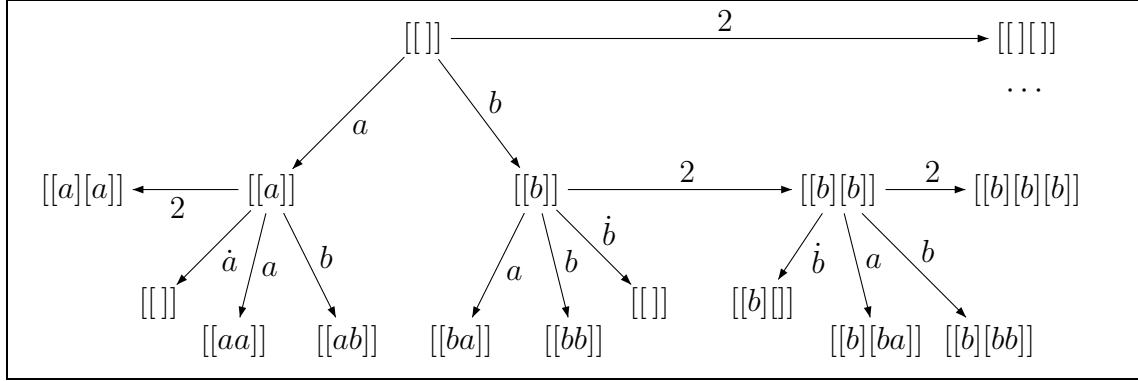
Figure 5.3: Graph G_1 for $\Gamma = \{a, b\}$

Hence we suppose that $2 \in L$ is a fresh symbol. A part of the graph G_1 is pictured in Figure 5.3. The loops labeled by 2 will be used to simulate the “copy” of the store content, *i.e.*, an operation $push_2$.

Then the tree $T_2 \in Tree_2$ is the unfolding of G_1 from the vertex that was the root of T_1 . In Figure 5.4 extra node-labels are added. They represent the corresponding 2-store. Note that several nodes can represent the same store content. The operations on 2-stores are simulated by paths in T_2 . More precisely, the effect of a transition is simulated in the following way if $\Gamma = \{a, b\}$:

$$\begin{aligned}
 (p, a, p', push_1^b) & \text{ corresponds to } \dot{a}ab, \\
 (p, a, p', pop_1) & \text{ corresponds to } \dot{a}, \\
 (p, a, p', push_2) & \text{ corresponds to } \dot{a}a2, \\
 (p, b, p', pop_2) & \text{ corresponds to } \dot{b}(\bar{a} + \bar{b} + \bar{a} + \bar{b})^* \bar{2}.
 \end{aligned}$$

Of course the expression $\dot{b}(\bar{a} + \bar{b} + \bar{a} + \bar{b})^* \bar{2}$ is regular, and one can move along such a path using three states of \mathcal{A} . Because we are on a tree, there is no infinite upward path. Following a 2-arc allows to copy the top 1-store because we stay exactly in the same position in G_1 . For popping the top 1-store, one has to find the last 2-arc that was used, and follow it in the reverse direction. Note that just after a $push_2$ (a 2-arc), we cannot move along an inverse arc \bar{a} (to simulate a pop_1), that’s why the arcs \dot{a} are necessary.


 Figure 5.4: An initial part of the tree T_2

Case $n = 3$. We go on with $G_2 \in Graph_2$, defined from T_2 by

$$G_2 = h_2^{-1}(T_2)$$

where the substitution h_2 is the following:

$$\begin{aligned} h_2(a) &= a \text{ for all } a \in \Gamma, & h_2(2) &= 2, \\ h_2(\dot{a}) &= \dot{a} \text{ for all } a \in \Gamma, & h_2(\dot{2}) &= \{\bar{a}, \bar{\dot{a}} \mid a \in \Gamma\}^* \bar{2}, \\ h_2(3) &= \varepsilon. \end{aligned}$$

Then $T_3 \in Tree_3$ is the unfolding of G_2 from the “root” (of T_2). On T_3 one can simulate a 3-store, almost the same way as a 2-store is simulated on T_2 (here $\Gamma = \{a, b\}$):

$$\begin{aligned} (p, a, p', push_1^b) &\text{ corresponds to } \dot{a}ab, \\ (p, a, p', pop_1) &\text{ corresponds to } \dot{a}, \\ (p, a, p', push_2) &\text{ corresponds to } \dot{a}a\bar{2}, \\ (p, a, p', pop_2) &\text{ corresponds to } \dot{a}\dot{\bar{2}}, \\ (p, a, p', push_3) &\text{ corresponds to } \dot{a}a\bar{3}, \\ (p, a, p', pop_3) &\text{ corresponds to } \dot{a}(\bar{2} + \bar{\dot{2}} + \bar{a} + \bar{b} + \bar{\dot{a}} + \bar{\dot{b}})^* \bar{3}. \end{aligned}$$

General case. It is easy to follow the construction: for $n \geq 3$, G_n is obtained from

T_n using following substitution h_n :

$$\begin{aligned} h_n(a) &= a \text{ for all } a \in \Gamma, & h_n(k) &= k \text{ for all } 2 \leq k \leq n, \\ h_n(\dot{a}) &= \dot{a} \text{ for all } a \in \Gamma, & h_n(\dot{k}) &= \dot{k} \text{ for all } 2 \leq k < n, \\ h_n(\dot{n}) &= \left\{ \bar{a}, \bar{\dot{a}}, \bar{k}, \bar{\dot{k}} \mid a \in \Gamma, 2 \leq k < n \right\}^* \bar{n}, \\ h_n(n+1) &= \varepsilon, \end{aligned}$$

and T_{n+1} is the unfolding of G_n from the “root”. The automaton \mathcal{A} has the same states as H plus auxiliary states for the regular expressions. It is clear that the winner of \mathcal{G} is the winner of (T, \mathcal{A}) , and a winning strategy in (T, \mathcal{A}) can be translated to a winning strategy in \mathcal{G} (the other direction holds also here). ■

Note that the tree $T \in Tree_n$ depends only on n and Γ . Note also that (as shown in Figure 5.4) several nodes can represent the same stack content. As we are just interested in game-simulation and not isomorphism this is not a problem. We consider only the behavior of the system. The game simulation can also be compared to the notion of weak (bi)simulation. This reduction from HPDGS to Caucal graph is sufficient to solve parity games on HPDGS, but we present also a game simulation in the other direction, to show that the model of Caucal graph with graph automaton is not more expressive than that of HPDGS, in terms of games.

5.2.2 From Caucal Graphs to HPDGS

Lemma 5.2.2 *Given a graph $G \in Graph_n$ and a graph automaton \mathcal{A} , one can construct a game structure \mathcal{G} on a HPDGS H of level n such that (G, \mathcal{A}) is game-simulated by \mathcal{G} .*

Proof: The result is clear for $n = 0$, because G and \mathcal{A} have a finite number of vertices and states.

Given $T_1 \in Tree_1$, $T_1 = Unf(G_0, s)$ for some $G_0 \in Graph_0$, we let $\Gamma = V_{G_0} \times L_{G_0}$. Letters from Γ will be pushed on a 1-store to remember the position in the unfolding, which is a path from s . Additionally the labels from L_{G_0} will allow to determine which inverse arc is possible from the current position. To simplify the notation we write $\langle a, q' \rangle \in \delta(q)$ if $\langle a, q' \rangle$ is an atom present in the formula $\delta(q)$. It is clear that the existential/universal choices in the formula can be expressed in the control

states of a HPDGS, so we skip this part and concentrate on the actual “moves”:

$$\begin{aligned} \langle a, q' \rangle \in \delta(q) &\text{ corresponds to } (q, (v, -), q', \text{push}_1^{(u,a)}) \text{ if } v \xrightarrow[G_0]{a} u, \\ \langle \bar{a}, q' \rangle \in \delta(q) &\text{ corresponds to } (q, (-, a), q', \text{pop}_1). \end{aligned}$$

A graph in $Graph_1$ can be simulated the same way using intermediate states for the rational substitutions.

Let $T_2 \in Tree_2$, $T_2 = Unf(G_1, s)$, T_2 can be simulated by a 2-store: each transition of G_1 is simulated on the top 1-store just like above, but the top 1-store has to be “copied” by a $push_2$ operation to keep track of the unfolding. It is also necessary to remember at each move the label of the arc of G_1 that was used. A solution is to use the following stack alphabet:

$$\Gamma = (V_{G_0} \times L_{G_0}) \uplus (\{2\} \times L_{G_1}).$$

An action $\langle a, q' \rangle \in \delta(q)$ is simulated by the following sequence of operations:

$$\begin{aligned} &\text{push}_2 \\ &< \text{simulation of an } a\text{-arc of } G_1 \text{ on the top 1-store } > \\ &\text{push}_1^{(2,a)}. \end{aligned}$$

And an action $\langle \bar{a}, q' \rangle \in \delta(q)$ in the following way:

$$\begin{aligned} &< \text{check that the top symbol is } (2, a) > \\ &\text{pop}_2. \end{aligned}$$

And so on for $n \geq 3$. This construction is more natural if we use the model of higher order pushdown automata from [Eng83], but both models are clearly equivalent [KNU02]. ■

5.3 Reducing the Hierarchy Level

In this section we present the main result of this chapter: an algorithmic solution of parity games on the graphs of the Caucal hierarchy, and hence on HPDGS. The proof is by induction on the definition of the hierarchy, using the next two lemmas to obtain graphs of lower levels.

Lemma 5.3.1 *Given $G \in Graph_n$ and a graph automaton \mathcal{A} , the game (G, \mathcal{A}) can be effectively simulated by a game (T, \mathcal{B}) , where $T \in Tree_n$, such that $G = h^{-1}(T)$, and \mathcal{B} is a graph automaton.*

The proof uses similar techniques as in Section 4.5.1 or [KV00] for the restricted case of prefix-recognizable graphs.

Proof: By definition $G = h^{-1}(T)$. The aim is to “simulate” an a -transition of \mathcal{A} along an arc of G by a path in T : a sequence of transitions of \mathcal{B} labeled by a word of $h(a)$. The automaton $\mathcal{B} = (Q_{\mathcal{B}}, W_{\mathcal{B}}, \delta_{\mathcal{B}}, q_0, \Omega_{\mathcal{B}})$ will have the same states as \mathcal{A} plus auxiliary states for this simulation. For each $a \in L$, $h(a)$ is regular. If $h(a) \neq \emptyset$, let

$$\mathcal{C}_a = (Q_a, W_a, \Delta_a, q_{0a}, F_a)$$

be a (non-deterministic) finite automaton on finite words recognizing $h(a)$. Here F_a is the set of final states. We consider \mathcal{C}_a as a finite graph, and note the transitions $q_a \xrightarrow[\mathcal{C}_a]{b} q'_a$ for $q_a, q'_a \in Q_a$. The new auxiliary states of \mathcal{B} are of the form $(q_a, [q])$ and $(q_a, \langle q \rangle)$ for $q \in Q_{\mathcal{A}}$, $q_a \in Q_a$. To obtain the transitions of \mathcal{B} from the transitions of \mathcal{A} ,

$$\begin{aligned} \text{each atom } \langle [a], q \rangle & \text{ is replaced by } \langle \varepsilon, (q_{0a}, [q]) \rangle , \\ \text{and each } \langle \langle a \rangle, q \rangle & \text{ is replaced by } \langle \varepsilon, (q_{0a}, \langle q \rangle) \rangle \end{aligned}$$

in the body of a transition $\delta_{\mathcal{A}}(q')$. Of course the atoms $\langle \varepsilon, q \rangle$ remain unchanged. Then the new transitions of \mathcal{B} are

$$\begin{aligned} \delta_{\mathcal{B}}((q_a, [q])) &= \bigwedge_{q_a \xrightarrow[\mathcal{C}_a]{b} q'_a} \langle [b], (q'_a, [q]) \rangle \wedge \bigwedge_{q_a \in F_a} \langle \varepsilon, q \rangle , \\ \delta_{\mathcal{B}}((q_a, \langle q \rangle)) &= \bigvee_{q_a \xrightarrow[\mathcal{C}_a]{b} q'_a} \langle \langle b \rangle, (q'_a, \langle q \rangle) \rangle \vee \bigvee_{q_a \in F_a} \langle \varepsilon, q \rangle , \end{aligned}$$

for each $a \in L$ such that $h(a) \neq \emptyset$.

To avoid the game to stay forever in the intermediate nodes of \mathcal{B} , we assign to these nodes a priority that is losing for the corresponding player. Suppose that the priority function $\Omega_{\mathcal{A}}$ of \mathcal{A} ranges from 0 to $2c$, $c \geq 0$, then we fix

$$\Omega_{\mathcal{B}}((q_a, [q])) = 2c , \quad \Omega_{\mathcal{B}}((q_a, \langle q \rangle)) = 2c + 1 .$$

And dually if the maximal priority of \mathcal{A} is $2c + 1$, then the new priorities are $2c + 1$ and $2c + 2$. They do not interfere with the “real” game (G, \mathcal{A}) . So one has one new priority and in the worst case the number of states of \mathcal{B} is $|Q_{\mathcal{B}}| = |Q_{\mathcal{A}}| \left(1 + \sum_{h(a) \neq \emptyset} |Q_a|\right)$. ■

Lemma 5.3.2 *Given $T \in \text{Tree}_{n+1}$ and a graph automaton \mathcal{A} , the game (T, \mathcal{A}) can be effectively simulated by a game (G, \mathcal{B}) , where $G \in \text{Graph}_n$, such that $T = \text{Unf}(G, s)$, and \mathcal{B} is a graph automaton.*

This result is related to the k -covering of [CW98], where k is the number of states of \mathcal{A} . The proof is based here on the construction of a one-way tree automaton that is equivalent to \mathcal{A} , because a one-way automaton cannot distinguish T and G . This construction was presented in [Var98] only in the case of (deterministic) trees of finite degree. It is extended here not only inside the Caucal hierarchy, but for any tree.

The idea is that if Player 0 has a winning strategy in (T, \mathcal{A}) , then he has also a positional winning strategy [EJ91]: choosing always the same transition from the same vertex. This strategy can be encoded as a labeling of T using a (big) finite alphabet. Then several conditions have to be checked to verify that this strategy is winning, but it can be done by a one-way automaton. Finely this strategy can be non-deterministically guessed by the automaton.

We give here a flavor of this proof, details are in Section 5.3.1. Formally a strategy for \mathcal{A} and a given tree T is a mapping

$$\tau : V_T \longrightarrow \mathcal{P}(Q \times \text{next}(W) \times Q) .$$

An element $(q, d, q') \in \tau(x)$ means that when arriving at node $x \in V_T$ in state q , the automaton should send a copy in state q' to the node in direction d (and maybe other copies in other directions). A strategy must satisfy the transition of \mathcal{A} , and a strategy has to be followed:

$$\begin{aligned} & \forall x \in V_T, \forall (q, d, q') \in \tau(x) : \\ & \{(d_2, q_2) \mid (q, d_2, q_2) \in \tau(x)\} \text{ satisfies } \delta(q) \text{ and:} \\ & - \text{ if } d = \varepsilon \text{ then } \exists d_1, q_1, (q', d_1, q_1) \in \tau(x) \text{ or } \emptyset \text{ satisfies } \delta(q') , \\ & - \text{ if } d = [a] \text{ then } \forall y : x \xrightarrow{T}^a y \Rightarrow \exists d_1, q_1, (q', d_1, q_1) \in \tau(y) \text{ or } \emptyset \text{ satisfies } \delta(q') , \\ & - \text{ if } d = \langle a \rangle \text{ then } \exists y : x \xrightarrow{T}^a y \wedge \exists d_1, q_1, (q', d_1, q_1) \in \tau(y) \text{ or } \emptyset \text{ satisfies } \delta(q') . \end{aligned}$$

For the root $t_0 \in V_T$ we have:

$$\exists d_1, q_1, (q_0, d_1, q_1) \in \tau(t_0) \text{ or } \emptyset \text{ satisfies } \delta(q_0) . \quad (5.1)$$

Considering $St := \mathcal{P}(Q \times \text{next}(W) \times Q)$ as an alphabet, a St -labeled tree (T, τ) defines a positional strategy on the tree T . One can construct a *one-way* automaton that checks that this strategy is correct according to the previous requirements.

The second step of the reduction from two-way to one-way is concerned with the priorities seen along (a branch of) the run, when one follows a strategy τ . To check the acceptance condition, it is necessary to follow each path of \mathcal{A} in T up and down, and remember the priorities appearing. Such a path can be decomposed into a *downward* path and several finite *detours* from the path, that come back to their origin (in a loop). Because each node has a unique parent and \mathcal{A} starts at the root, we consider only downward detour (each move \bar{a} is in a detour). That is to say, if a node is visited more than once by a run, we know that the first time it was visited, the run came from above. This idea of detour is close to the idea of subgame in [Wal96b], see Section 4.2. To keep track of these finite detours, we use the following annotation. An annotation for \mathcal{A} , a given tree T and a strategy τ is a mapping

$$\eta : V_T \longrightarrow \mathcal{P}(Q \times [max] \times Q) .$$

Intuitively $(q, f, q') \in \eta(x)$ means that from node x and state q there is a detour that comes back to x with state q' and the smallest priority seen along this detour is f . The notion of annotation is very close to the “claims” of Player 0 in the method of Section 4.2. Again η can be considered as a labeling of T , and a one-way automaton can check that the annotation is consistent with respect to the strategy in reading both labellings. A typical requirement is:

$$(q, [a], q_1) \in \tau(x) \Rightarrow \forall y \in V_T : x \xrightarrow{a} y \Rightarrow [(q_1, \bar{a}, q') \in \tau(y) \Rightarrow (q, \min(\Omega(q_1), \Omega(q')), q') \in \eta(x)] .$$

The last step is to check every possible branch of the run by using the detours: it is easy to define a one-way automaton \mathcal{E} that “simulates” (follow) a branch of the run of \mathcal{A} . One can change the acceptance condition of \mathcal{E} such that it accepts a tree labeled by τ and η if and only if *there exists* a branch in the corresponding run of \mathcal{A} that *violates* the acceptance condition of \mathcal{A} . Then using techniques from [Tho97] one can determinize and complement \mathcal{E} . Finally the product of the previous automata has to be build, to check all conditions together, and a “projection” is necessary to nondeterministically guess the labels, *i.e.*, the strategy and the annotation.

Theorem 5.3.3 *Parity games on higher order pushdown systems are solvable: one can determine the winner and compute a winning strategy.*

As a corollary we get a new proof that the μ -calculus model checking of these graphs is decidable (it was known as a consequence of the MSO-decidability).

Proof: Given a game structure \mathcal{G} on a HPDGS H of level n , one obtains from

Theorem 5.2.1 a graph automaton \mathcal{A} and a tree $T \in Tree_n$ such that (T, \mathcal{A}) is a game simulation of \mathcal{G} . By successive reductions using Lemmas 5.3.1 and 5.3.2, one can obtain a game on a finite graph which is equivalent to the initial game. Using classical techniques (see [GTW02, ch. 7]), one can solve this game, and compute a positional strategy for the winner. Then one can step by step reconstruct the strategy for the graphs of higher levels. ■

5.3.1 Proof of Lemma 5.3.2

We adapt the construction from [Var98] to the case of infinite degree. Recall that $W \subseteq L \cup \bar{L}$. As remarked above, the question whether Player 0 wins the game (T, \mathcal{A}) is equivalent to the question whether there is an accepting run of \mathcal{A} on T . We want to check this condition with a *one-way* tree automaton.

We know from [EJ91] that if Player 0 has a winning strategy, then he has a memoryless winning strategy. In other words, if \mathcal{A} has an accepting run, then it has an accepting run using a memoryless strategy: choosing always the same “transitions” from the same node and state. We decompose the run of \mathcal{A} using this strategy.

Definition 5.3.4 A strategy for \mathcal{A} and a given tree T is a mapping

$$\tau : V_T \longrightarrow \mathcal{P}(Q \times next(W) \times Q) .$$

A strategy must satisfy the transitions of \mathcal{A} , and a strategy has to be followed:

$$\forall x \in V_T, \forall (q, d, q') \in \tau(x) : \{ (d_2, q_2) \mid (q, d_2, q_2) \in \tau(x) \} \text{ satisfies } \delta(q) \text{ and:} \quad (5.2)$$

$$\text{- if } d = \varepsilon \text{ then } \exists d_1, q_1, (q', d_1, q_1) \in \tau(x) \text{ or } \emptyset \text{ satisfies } \delta(q') , \quad (5.3)$$

$$\text{- if } d = [a] \text{ then } \forall y : x \xrightarrow{a}_T y \Rightarrow \exists d_1, q_1, (q', d_1, q_1) \in \tau(y) \text{ or } \emptyset \text{ satisfies } \delta(q') , \quad (5.4)$$

$$\text{- if } d = \langle a \rangle \text{ then } \exists y : x \xrightarrow{a}_T y \wedge \exists d_1, q_1, (q', d_1, q_1) \in \tau(y) \text{ or } \emptyset \text{ satisfies } \delta(q') . \quad (5.5)$$

For the root $t_0 \in V_T$ we have:

$$\exists d_1, q_1, (q_0, d_1, q_1) \in \tau(t_0) \text{ or } \emptyset \text{ satisfies } \delta(q_0) . \quad (5.6)$$

Considering $St := \mathcal{P}(Q \times next(W) \times Q)$ as an alphabet, a St -labeled tree (T, τ) defines a memoryless strategy on the tree T . We will construct a *one-way* automaton

\mathcal{C} that checks that this strategy is correct according to the previous requirements. For $(q, \langle a \rangle, q') \in \tau(x)$, if $a \in L$ it has just to check in the direction a downward that the strategy is well defined for q' , but if $(q, \langle \bar{a} \rangle, q') \in \tau(x)$, $\bar{a} \in \bar{L}$ he must have *remembered* that the strategy *was* defined for q' in the parent-node, and that the arc from the parent node is labeled by a . The states of \mathcal{C} are tuples $\langle Q_1, Q_2, \ell \rangle \in \mathcal{P}(Q) \times \mathcal{P}(Q) \times (W \cap L)$, where ℓ is the label of the arc from the parent node, $q' \in Q_1$ means that \mathcal{C} has to check (down) that the strategy can be followed for q' , and $q'' \in Q_2$ means that q'' is already allowed at the parent node. Note that if a is the label of the arc from the parent node, then the actions $[\bar{a}]$ and $\langle \bar{a} \rangle$ are equivalent, because in a tree there is no more than one parent node. So we shall write simply \bar{a} for both cases when we know from the context that it is right. If a is *not* the label from the parent node, then $[\bar{a}]$ means immediate “wins”, and $\langle \bar{a} \rangle$ means immediate “lost” (for Player 0). Let

$$\mathcal{C} := (\mathcal{P}(Q) \times \mathcal{P}(Q) \times \ell, St, \delta_{\mathcal{C}}, \langle \{q_0\}, \emptyset, b \rangle, true) \text{ where} \quad (5.7)$$

$$\delta_{\mathcal{C}}(\langle Q_1, Q_2, \ell \rangle, \tau_1) :=$$

$$\text{IF } \forall q \in Q_1, \{(d_2, q_2) : (q, d_2, q_2) \in \tau_1\} \text{ satisfies } \delta(q), \text{ and} \quad (5.8)$$

$$\forall (q', \varepsilon, q) \in \tau_1, \{(d_2, q_2) : (q, d_2, q_2) \in \tau_1\} \text{ satisfies } \delta(q), \text{ and} \quad (5.9)$$

$$\forall (q, [\bar{a}], q') \in \tau_1 : a = \ell \Rightarrow q' \in Q_2, \text{ and} \quad (5.10)$$

$$\forall (q, \langle \bar{a} \rangle, q') \in \tau_1 : a = \ell \wedge q' \in Q_2 \quad (5.11)$$

$$\text{THEN } \bigwedge_{d \in \text{next}(W \cap L) \setminus \{\varepsilon\}} (d, \langle \{q' : \exists (q, d, q') \in \tau_1\}, Q'_2, d \rangle) \quad (5.12)$$

$$\text{with } Q'_2 := \{q'' : \exists d_1, q_1, (q'', d_1, q_1) \in \tau_1 \text{ or } \emptyset \text{ satisfies } \delta(q'')\}, \quad (5.13)$$

$$\text{ELSE } false. \quad (5.14)$$

Condition (5.9) is related to (5.3), (5.10) and (5.11) to (5.4) and (5.5) where $a \in \bar{L}$, and (5.12) to (5.4) and (5.5) where $a \in L$. In condition (5.8) there is no requirement on the $q \notin Q_1$, that's why the condition (5.2) above is stronger. This is not a problem for the following, as we are searching *some* winning strategy (one could define the *minimal* valid strategy as in [Var98]).

The acceptance condition is easy to enunciate: it just requires that the run can be followed, *i.e.*, the transition is possible at each node. In the initial state $\langle \{q_0\}, \emptyset, b \rangle$, the letter b is not relevant. \mathcal{C} is a one-way automaton with $4^{|Q|}$ states.

Proposition 5.3.5 *A two-way alternating parity automaton accepts an input tree if and only if it has an accepting strategy over the input tree.*

With the help of a so called annotation, we will check in the following whether a strategy is accepting.

Annotation

The previous automaton \mathcal{C} just checks that the strategy can be followed (ad infinitum) but forgets about the priorities of \mathcal{A} . To check the acceptance condition, it is necessary to follow each path of \mathcal{A} up and down, and remember the priorities appearing. Such a path can be decomposed into a *downward* path and several finite *detours* from the path, that come back to their origin (in a loop). Because each node has a unique parent and \mathcal{A} starts at the root, we consider only downward detour (each move \bar{a} is in a detour). That is to say, if a node is visited more than once by a run, we know that the first time it was visited, the run came from above. This idea of detour is close to the idea of subgame in [Wal96b], see Section 4.2. To keep track of these finite detours, we use the following annotation.

Definition 5.3.6 *An annotation for \mathcal{A} and a given tree T is a mapping*

$$\eta : V_T \longrightarrow \mathcal{P}(Q \times [max] \times Q) .$$

Intuitively $(q, f, q') \in \eta(x)$ means that from node x and state q there is a detour that comes back to x with state q' and the smallest priority seen along this detour is f . By definition, the following conditions are required for the annotation η of a given strategy τ , indeed a detour can stay in the node x (5.15, 5.16), go down to a child y and come back immediately to x (5.17, 5.19), or go down to y , use another detour from y and then come back to x (5.18, 5.20).

$$\forall q, q' \in Q, x \in V_T, a \in W \cap L, f, f' \in [max] :$$

$$(q, \varepsilon, q') \in \tau(x) \Rightarrow (q, \Omega(q'), q') \in \eta(x) , \quad (5.15)$$

$$(q_1, f, q_2) \in \eta(x) \wedge (q_2, f', q_3) \in \eta(x) \Rightarrow (q_1, \min(f, f'), q_3) \in \eta(x) , \quad (5.16)$$

$$(q, [a], q_1) \in \tau(x) \Rightarrow \forall y \in V_T : x \xrightarrow{a} y \Rightarrow [(q_1, \bar{a}, q') \in \tau(y) \Rightarrow (q, \min(\Omega(q_1), \Omega(q')), q') \in \eta(x)] , \quad (5.17)$$

$$(q, [a], q_1) \in \tau(x) \Rightarrow \forall y \in V_T : x \xrightarrow{a} y \Rightarrow [(q_1, f, q_2) \in \eta(y) \wedge (q_2, \bar{a}, q') \in \tau(y) \Rightarrow (q, \min(\Omega(q_1), f, \Omega(q')), q') \in \eta(x)] . \quad (5.18)$$

The next conditions have to be “synchronized” with the automaton \mathcal{C} that checks

the strategy. The a -successor y of x is “chosen” by \mathcal{C} :

$$(q, \langle a \rangle, q_1) \in \tau(x) \Rightarrow \exists y \in V_T : x \xrightarrow{a} y \wedge [(q_1, \bar{a}, q') \in \tau(y) \Rightarrow (q, \min(\Omega(q_1), \Omega(q')), q') \in \eta(x)] , \quad (5.19)$$

$$(q, \langle a \rangle, q_1) \in \tau(x) \Rightarrow \exists y \in V_T : x \xrightarrow{a} y \wedge [(q_1, f, q_2) \in \eta(y) \wedge (q_2, \bar{a}, q') \in \tau(y) \Rightarrow (q, \min(\Omega(q_1), f, \Omega(q')), q') \in \eta(x)] . \quad (5.20)$$

Considering $An := \mathcal{P}(Q \times [max] \times Q)$ as an alphabet, the aim is now to construct a *one-way* automaton \mathcal{D} on $(An \times St)$ -labeled trees that checks that the annotation satisfies these requirements. Conditions 5.15 and 5.16 above can be checked in each node (independently) without memory. For conditions 5.17 to 5.20, the automaton has to remember the whole $\eta(x)$ from the parent node x , and the part of $\tau(x)$ leading to the current node. Let

$$\mathcal{D} := (An \times \mathcal{P}(Q \times Q), An \times St, \delta_{\mathcal{D}}, \langle \emptyset, \emptyset \rangle, true) ,$$

where

$$\delta_{\mathcal{D}}(\langle \eta_0, \alpha \rangle, \langle \eta_1, \tau_1 \rangle) :=$$

IF conditions 5.15 and 5.16 hold for η_1 and τ_1 AND

$$\forall (q, q_1) \in \alpha : (q_1, \bar{a}, q') \in \tau_1 \Rightarrow (q, \min(\Omega(q_1), \Omega(q')), q') \in \eta_0 \quad (5.21)$$

$$\forall (q, q_1) \in \alpha : (q_1, f, q_2) \in \eta_1 \wedge (q_2, \bar{a}, q') \in \tau_1 \Rightarrow (q, \min(\Omega(q_1), f, \Omega(q')), q') \in \eta_0 \quad (5.22)$$

THEN $\bigwedge_{d \in \text{next}(W \cap L) \setminus \{\varepsilon\}} (d, \langle \eta_1, \{(q, q_1) : \text{exists } (q, d, q_1) \in \tau_1 \} \rangle)$

ELSE *false* .

Condition (5.21) is related to (5.17, 5.19), and (5.22) to (5.18, 5.20). We recall that in the case that $(q, \langle a \rangle, q_1) \in \tau_1$, the $\langle a \rangle$ -transition has to be synchronized with the one of the automaton \mathcal{C} : the product automaton that we will define later has to chose an a -successor and follow the run of \mathcal{C} and \mathcal{D} from this successor.

Similarly to \mathcal{C} , \mathcal{D} is a one-way automaton with $2^{|Q|^{2m}} \cdot 2^{|Q|^2} = 2^{|Q|^2(m+1)}$ states, and the acceptance condition is again trivial. Note that if a part of the tree is not visited by the original automaton \mathcal{A} , the strategy and annotation can be empty on this part. The automaton \mathcal{D} does not check that the annotation is minimal, but this is not a problem. With the help of the annotation one can determine if a path of \mathcal{A} verify the acceptance condition or not, as showed in the next part of the proof.

Parity Acceptance

Up to now the automata \mathcal{C} and \mathcal{D} together just check that the strategy and annotation for the run of \mathcal{A} are correct, but do not verify that the run of \mathcal{A} is accepting, *i.e.*, that each path is valid. With the help of the annotation we can “simulate” (follow) a path of \mathcal{A} with a one-way automaton, and determine the parity condition for this path. This one-way automaton does not go into the detours, but reads the smallest priority appearing in them. Let

$$\begin{aligned} \mathcal{E} &:= (Q \times [max], An \times St, \delta_{\mathcal{E}}, \langle q_0, 0 \rangle, \Omega_0) , \\ \delta_{\mathcal{E}}(\langle q, i \rangle, \langle \eta_1, \tau_1 \rangle) &:= \bigvee_{(q,d,q') \in \tau_1, d=[a] \vee d=(a)} (\langle a \rangle, \langle q', \Omega(q') \rangle) \vee \bigvee_{(q,f,q') \in \eta_1} (\varepsilon, \langle q', f \rangle) . \end{aligned}$$

Once again if $d = \langle a \rangle$, the automaton has to be synchronized with \mathcal{C} and \mathcal{D} to choose the right a -successor. At each step \mathcal{E} either goes *down* following the strategy, or simulates a detour with an ε -move and the corresponding priority. The second component ($[max]$) of the states of \mathcal{E} just remembers the last priority seen. We can transform \mathcal{E} into a nondeterministic one-way automaton \mathcal{E}' without ε -moves with the same state space. Note that \mathcal{E} can possibly stay forever in the same node by using ε -transitions, either in an accepting run or not. This possibility can be checked by \mathcal{E}' just by reading the current annotation, with a transition *true* or *false*.

We will use \mathcal{E} and \mathcal{E}' to find the *invalid* paths of the run of \mathcal{A} , just by changing the acceptance condition: $\Omega_0(\langle q, i \rangle) := i + 1$.

Proposition 5.3.7 *The one-way tree automaton \mathcal{E}' accepts a $(An \times St)$ -labeled tree if and only if the corresponding run of \mathcal{A} is not accepting.*

But \mathcal{E}' is not deterministic, and accepts a tree if \mathcal{E}' has *some* accepting run. We can view \mathcal{E}' as a word automaton: it follows just a branch of the tree. For this reason it is possible to co-determinize it: determinize and complement it in a singly exponential construction (see [Tho97]) to construct the automaton $\overline{\mathcal{E}}$ that accepts those of the $(An \times St)$ -labeled trees that represent the accepting runs of \mathcal{A} .

We will define the product $\mathcal{B}' := \mathcal{C} \times \mathcal{D} \times \overline{\mathcal{E}}$ of the previous automata, that accepts a $(An \times St)$ -labeled input tree if and only if the corresponding run of \mathcal{A} is accepting. Let

$$\begin{aligned} \mathcal{B}' &:= (Q_{\mathcal{C}} \times Q_{\mathcal{D}} \times Q_{\overline{\mathcal{E}}}, An \times St, \delta_{\mathcal{B}'}, q_{0,\mathcal{B}'}, Acc) , \\ \delta_{\mathcal{B}'}(\langle q_{\mathcal{C}}, q_{\mathcal{D}}, q_{\overline{\mathcal{E}}} \rangle, \langle \eta_1, \tau_1 \rangle) &:= \langle \delta_{\mathcal{C}}(q_{\mathcal{C}}, \langle \tau_1 \rangle), \delta_{\mathcal{D}}(q_{\mathcal{D}}, \langle \eta_1, \tau_1 \rangle), \delta_{\overline{\mathcal{E}}}(q_{\overline{\mathcal{E}}}, \langle \eta_1, \tau_1 \rangle) \rangle , \end{aligned}$$

where $Q_{\mathcal{C}}$ is the state space of \mathcal{C} , and so on. The acceptance condition Acc of \mathcal{B}' is then exactly the one of $\overline{\mathcal{E}}$: $\Omega_{\mathcal{B}'}(\langle q_{\mathcal{C}}, q_{\mathcal{D}}, q_{\overline{\mathcal{E}}} \rangle) = \Omega_{\overline{\mathcal{E}}}(q_{\overline{\mathcal{E}}})$.

We define the automaton \mathcal{B} to be the “projection” of \mathcal{B}' : \mathcal{B} nondeterministically guesses the labels from $An \times St$, \mathcal{B} has no input alphabet. Finally \mathcal{B} is a one-way tree-automaton that is equivalent to \mathcal{A} : it accepts the same trees. The strategy and annotation depended on the input tree, now after the projection, \mathcal{B} can search the run of \mathcal{A} for each input tree. The automaton \mathcal{B} has (like \mathcal{B}') $4^{|Q|} \cdot 2^{|Q|^2(m+1)} \cdot 2^{c|Q|^m} = 2^{|Q|^2(m+1)} \cdot 2^{|Q|(2+cm)}$ states. As a one-way automaton, \mathcal{B} cannot distinguish T and G . In other words, it has an accepting run on T if and only if it has one on G . Positional strategy on G gives a “regular” run on T .

5.4 Complexity and Strategies

The one-step reduction of Lemma 5.3.2 is in exponential time in the description of $T \in Tree_{n+1}$ and \mathcal{A} , and the size of the output is also exponential. For this reason the complexity of the complete solution of a parity game on a Caucal graph or on a HPDGS is a tower of exponentials whose height is the level of the graph. More precisely to obtain the best complexity bound one should not reduce the game to a finite graph and rather use the results from [KV00] or [Wal96b] that give an exponential time solution to parity games over 1-HPDGS. We define *Tower*, the “tower of exponentials” function, by $Tower(0, k) = k$ and $Tower(n + 1, k) = 2^{Tower(n, k)}$. So [KV00] or [Wal96b] together with Theorem 5.2.1 and Lemmas 5.3.1 and 5.3.2 yield the following:

Theorem 5.4.1 *Given a Higher Order Pushdown Game System of level n and size k (of the finite description) one can determine the winner in time $Tower(n, k)$.*

The classical translation from parity game to μ -calculus to MSO and the corresponding decision procedure is already non-elementary (in the number of priorities) for level 1 graphs. And following [Wal96a] the (one-step) transformation of an MSO-formula from the unfolding to the original graph is also non-elementary.

Using the reductions presented here, one can compute a winning strategy for a 1-HPDGS game which is a finite automaton that reads the current configuration and outputs the “next move”, like in [KV00, Cac02b]. But it is more natural to consider a *pushdown strategy* as introduced in [Wal96b]. It is a pushdown transducer that reads the moves of Player 1 and outputs the moves of Player 0. It needs additional memory (the stack), but the computation of the “next move” can be done in constant time. When we recompose the game, a strategy for an n -HPDGS game is an n -HPDGS with input and output which possibly needs to execute several transitions to compute the “next move” from a given configuration.

Our aim was to solve parity games and μ -calculus model-checking on HPDGS and Caucal graphs, but after developing these techniques, we have as a by-product a new proof of the decidability of the MSO logic over Caucal graphs, with new complexity bounds. Given a graph $G \in Graph_n$ and an MSO formula φ , by definition $G = h^{-1}(T)$ for some tree $T \in Tree_n$, and one can transform φ into a formula ψ such that φ is true in G if and only if ψ is true in T . Then one can transform ψ into a tree automaton \mathcal{A} running on T and use the previous reductions to determine if \mathcal{A} has an accepting run on T .

Chapter 6

Conclusion

In this thesis we have considered different winning conditions: reachability, Büchi, parity, Σ_3 ; and different game graphs: Pushdown Game System (PDGS) and Higher Order Pushdown Game System (HPDGS). We summarize now the different results.

6.1 Comparison of Both Approaches

After having developed a symbolic approach in Chapter 3 and techniques of game-simulation in Chapters 4 and 5, we shall compare them (see Figure 6.1). The first distinction is that both approaches do not solve the same problems. It was possible to solve the Σ_3 game with a symbolic method, whereas we did not find any game reduction for it. On the other hand the game reduction was appropriate for parity games in general and for HPDGS and Caucal graphs in particular. A reason why no symbolic approach seems to be able to solve parity games is that there is no *simple* expression of the winning region of one player in terms of attractor or fixed-point (in the known μ -calculus formula the fixed-points are imbricated and cannot be computed one after the other).

Nevertheless, reachability games and Büchi games, the typical cases in analysis of safety and liveness conditions, are solved by both approaches and it is possible to compare the results in terms of complexity. Given a PDGS $(P_0, P_1, \Gamma, \Delta)$, $P = P_0 \uplus P_1$, and a Büchi winning condition, which is a parity game with two colors, the game reduction from [Wal96b] presented in Chapter 4 constructs a finite graph with $\mathcal{O}(|\Gamma|^2 2^{c|P|})$ vertices and $\mathcal{O}(|\Gamma|^3 2^{c|P|})$ transitions. Then the Büchi game on this finite graph can be solved in time $\mathcal{O}(|V|(|V| + |E|))$ where $|V|$ is the number of vertices and $|E|$ the number of edges. So the upper bound is comparable to that of the symbolic approach (Algorithm 3.4.8) which is $\mathcal{O}(|\Gamma| |\Delta| 2^{c'|P|^2})$. But the major

game graph / winning condition	symbolic approach	game-simulation
PDGS, reachability	$\mathcal{O}(\Delta 2^{c_1(P + Q)^2})$ (Theorem 3.1.3)	$\mathcal{O}(\Gamma ^3 2^{c_2 P Q })$
PDGS, Büchi	$\mathcal{O}(\Gamma \Delta 2^{c_3 P ^2})$ (Theorem 3.4.9)	$\mathcal{O}(\Gamma ^5 2^{c_4 P })$
PDGS, parity	?	EXPTIME (Theorem 4.2.2)
PDGS, Σ_3	EXPTIME (Theorem 3.6.9)	?
HPDGS of level n , parity	?	tower of exponentials in n (Theorem 5.4.1)

$|P|$: number of control states

$|\Gamma|$: size of the stack alphabet

$|\Delta|$: size of the transition relation

$|Q|$: number of states of the automaton defining the goal set R

Figure 6.1: Summary of the main results

difference is that in game reduction the whole finite graph is always constructed, even if the winning region is trivial. This leads to a lower bound, whereas in the symbolic approach the algorithm can be “lucky” if the winning region is small or simple, or if the PDGS has several connected components, or other special effects apply.

It is possible to adapt the game reduction of Chapter 4 to solve reachability games. In this case the reachability game over a PDGS is transformed into a reachability game over a finite graph, which is then solved in linear time. Nevertheless we face again the problem that the whole finite graph is constructed, which is exponential in the description of the PDGS. Moreover if the goal set R is given by a finite automaton with state space Q , then one has first to transform the PDGS using Proposition 3.4.3 such that the membership in R is determined only by the control state. For that P is replaced by $P \times Q$ and Γ by $\Gamma \times Q$. So this transformation has an extra computational cost whereas the symbolic approach of Section 3.1 can start directly with the finite automaton recognizing R . Again this is particularly relevant if the winning region is trivial or simple.

Using the symbolic method we have also the new feature of optimal strategy. It

is open how to extend this notion to the case of parity game.

6.2 Outlook

We mention here some possible tracks for future work.

One advantage of the symbolic approach is that the input and the output of the algorithm are both alternating automata recognizing respectively the goal set and the winning region. This might be used in a compositional way to consider more refined winning conditions. On the other hand the result of a game reduction is a pushdown strategy that could be used in his turn to define another game: it is a PDGS.

After the Σ_3 -condition of Section 3.6 it is possible to consider other conditions at higher levels in the Borel hierarchy, or combinations of different conditions. During the redaction of this thesis this has been partially done by Alexis Bouquet, Olivier Serre and Igor Walukiewicz [BSW03]. They have used a game reduction and it seems difficult to extend the symbolic approach to higher levels.

In the case of PDGS, an EXPTIME lower bound was found by Igor Walukiewicz [Wal96b] in the simple case of reachability games and this bound matches the upper bound for parity games. Concerning HPDGS it is open how to find a lower bound for reachability games or to simplify the construction in the restricted case of reachability. Recall that the construction of Chapter 5 transforms a reachability game at a higher level into a parity game on the lower levels.

Bibliography

- [BEF⁺00] Ahmed Bouajjani, Javier Esparza, Alain Finkel, Oded Maler, Peter Rossmanith, Bernard Willems, and Pierre Wolper. An efficient automata approach to some problems on context-free grammars. *Information Processing Letters*, 74(5-6):221–227, 2000. [23](#), [27](#)
- [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 '97*, volume 1243 of *Lecture Notes in Computer Science*, pages 135–150. Springer, 1997. [5](#), [6](#), [13](#), [23](#), [27](#)
- [BH70] J. Richard Büchi and William H. Hosken. Canonical systems which produce periodic sets. *Mathematical Systems Theory*, 4(1):81–90, 1970. [4](#)
- [BJW02] Julien Bernet, David Janin, and Igor Walukiewicz. Permissive strategies: from parity games to safety games. *Theoretical Informatics and Applications (RAIRO)*, 36:261–275, 2002. [88](#)
- [BL69] J. Richard Büchi and Lawrence H. Landweber. Solving sequential conditions by finite-state strategies. *Transactions of the American Mathematical Society*, 138:295–311, 1969. [5](#)
- [Bou01] Ahmed Bouajjani. Languages, rewriting systems, and verification of infinite-state systems. In *Proceedings of the 28th International Colloquium on Automata, Languages and Programming, ICALP'01*, volume 2076 of *Lecture Notes in Computer Science*, pages 24–39. Springer, 2001. [20](#)
- [BSW03] Alexis Bouquet, Olivier Serre, and Igor Walukiewicz. Pushdown games with unboundedness and regular conditions. In *Proceedings of the*

- 23th Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS'03*, 2003. to appear. [63](#), [131](#)
- [Büc64] J. Richard Büchi. Regular canonical systems. *Archiv für Mathematische Grundlagenforschung*, 6:91–111, 1964. [4](#)
- [Büc83] J. Richard Büchi. State-strategies for games in $f_{\sigma\delta} \cap g_{\delta\sigma}$. *Journal of Symbolic Logic*, 48(4):1171–1198, 1983. [63](#), [64](#)
- [Cac02a] Thierry Cachat. Symbolic strategy synthesis for games on pushdown graphs. In *Proceedings of the 29th International Colloquium on Automata, Languages, and Programming, ICALP'02*, volume 2380 of *Lecture Notes in Computer Science*, pages 704–715. Springer, 2002. [7](#)
- [Cac02b] Thierry Cachat. Two-way tree automata solving pushdown games. In Erich Grädel, Wolfgang Thomas, and Thomas Wilke, editors, *Automata, Logics, and Infinite Games [GTW02]*, volume 2500 of *Lecture Notes in Computer Science*, pages 303–317. Springer, 2002. [126](#)
- [Cac02c] Thierry Cachat. Uniform solution of parity games on prefix-recognizable graphs. In Antonin Kucera and Richard Mayr, editors, *Proceedings of the 4th International Workshop on Verification of Infinite-State Systems*, volume 68 of *Electronic Notes in Theoretical Computer Science*. Elsevier Science Publishers, 2002. [7](#), [87](#)
- [Cac03] 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'03*, volume 2719 of *Lecture Notes in Computer Science*, pages 556–569. Springer, 2003. [8](#)
- [Cau90] Didier Caucal. On the regular structure of prefix rewriting. In *Proceedings of the 15th Colloquium on Trees in Algebra and Programming, CAAP '90*, volume 431 of *Lecture Notes in Computer Science*, pages 87–102. Springer, 1990. see also the full version [[Cau92](#)]. [4](#), [5](#), [30](#)
- [Cau92] Didier Caucal. On the regular structure of prefix rewriting. *Theoretical Computer Science*, 106(1):61–86, 1992. [134](#)

- [Cau96] Didier Caucal. On infinite transition graphs having a decidable monadic theory. In *Proceedings of the 23rd International Colloquium on Automata, Languages and Programming, ICALP'96*, volume 1099 of *Lecture Notes in Computer Science*, pages 194–205. Springer, 1996. see also the full version [Cau03a]. 4, 83, 87, 98, 106, 107, 112, 135
- [Cau01] Didier Caucal. On the transition graphs of Turing machines. In *Proceedings of the 3rd International Conference Machines, Computations, and Universality, MCU'01*, volume 2055 of *Lecture Notes in Computer Science*, pages 177–189. Springer, 2001. see also the full version [Cau03b]. 135
- [Cau02] Didier Caucal. On infinite terms having a decidable monadic theory. In *Proceedings of the 27th International Symposium on Mathematical Foundations of Computer Science 2002, MFCS 2002*, volume 2420 of *Lecture Notes in Computer Science*, pages 165–176. Springer, 2002. 5, 7, 103, 105, 106, 107
- [Cau03a] Didier Caucal. On infinite transition graphs having a decidable monadic theory. *Theoretical Computer Science*, 290:79–115, 2003. full version of [Cau96]. 135
- [Cau03b] Didier Caucal. On the transition graphs of Turing machines. *Theoretical Computer Science*, 296(2):195–223, 2003. full version of [Cau01]. 101, 135
- [CBMS01] Didier Caucal, Olaf Burkart, Faron Moller, and Bernhard Steffen. Verification on infinite structures. In *Handbook of process algebra*, chapter 9, pages 545–623. Elsevier, 2001. 6
- [CDGV94] Jean-Luc Coquidé, Max Dauchet, Rémi Gilleron, and Sándor Vágvölgyi. Bottom-up tree pushdown automata: Classification and connection with rewrite systems. *Theoretical Computer Science*, 127(1):69–98, 1994. 4
- [CDT02] Thierry Cachat, Jacques Duparc, and Wolfgang Thomas. Solving pushdown games with a Σ_3 winning condition. In *Proceedings of the 11th Annual Conference of the European Association for Computer Science Logic, CSL'02*, volume 2471 of *Lecture Notes in Computer Science*, pages 322–336. Springer, 2002. 3, 7, 63, 82

- [CK02] Didier Caucal and Teodor Knapik. A chomsky-like hierarchy of infinite graphs. In *Proceedings of the 27th International Symposium on Mathematical Foundations of Computer Science, MFCS'02*, volume 2420 of *Lecture Notes in Computer Science*, pages 177–187. Springer, 2002. [20](#)
- [Cor01] Olivier Corolleur. Étude de jeux sur les graphes de transitions des automates à pile. Rapport de stage d'option informatique, Ecole Polytechnique, 2001. [27](#)
- [Cou90] Bruno Courcelle. Graph rewriting : An algebraic and logic approach. In Elsevier, editor, *Handbook of Theoretical Computer Science, Volume B*, J. Van Leeuwen ed., Elsevier, pages pp.193–242., 1990. [5](#)
- [Cou03] Bruno Courcelle. The monadic second-order logic of graphs XIV: Uniformly sparse graphs and edge set quantifications. *Theoretical Computer Science*, 299:1–36, 2003. [19](#)
- [CW98] Bruno Courcelle and Igor Walukiewicz. Monadic second-order logic, graph coverings and unfoldings of transition systems. *Annals of Pure and Applied Logic*, 92(1):35–62, 1998. [107](#), [119](#)
- [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 Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2003*, Lecture Notes in Computer Science. Springer, 2003. to appear. [111](#)
- [EHRS00a] 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'00*, volume 1855 of *Lecture Notes in Computer Science*, pages 232–247. Springer, 2000. see also the full version [[EHRS00b](#)]. [6](#), [23](#), [24](#), [27](#), [136](#)
- [EHRS00b] Javier Esparza, David Hansel, Peter Rossmanith, and Stefan Schwoon. Efficient algorithms for model checking pushdown systems. Technical Report TUM-I0002, SFB-Bericht 342/1/00 A, Technische Universität München, February 2000. full version of [[EHRS00a](#)]. [136](#)

- [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'91*, pages 368–377, Los Alamitos, California, October 1991. IEEE Computer Society Press. 5, 18, 19, 89, 93, 103, 119, 121
- [EJS93] E. Allen Emerson, Charanjit S. Jutla, and A. Prasad Sistla. On model-checking for fragments of mu-calculus. In *Proceedings of the 5th International Conference on Computer Aided Verification, CAV'93*, volume 697 of *Lecture Notes in Computer Science*, pages 385–396. Springer, June/July 1993. 5, 110
- [EKS01] Javier Esparza, Antonin Kucera, and Stefan Schwoon. Model-checking LTL with regular valuations for pushdown systems. In *Proceedings of the 4th international symposium on Theoretical Aspects of Computer Software, TACS'01*, volume 2215 of *Lecture Notes in Computer Science*, pages 316–339. Springer, 2001. 47
- [Eng83] Joost Engelfriet. Iterated pushdown automata and complexity classes. In *Proceedings of the 15th Annual ACM Symposium on Theory of Computing, STOC '83*, pages 365–373, 1983. 104, 117
- [FWW97] Alain Finkel, Bernard Willems, and Pierre Wolper. A direct symbolic approach to model checking pushdown systems. In *Proceedings of the 2nd International Workshop on Verification of Infinite State Systems, INFINITY'97*, volume 9 of *Electronic Notes in Theoretical Computer Science*. Elsevier Science Publishers, 1997. 47
- [GTW02] Erich Grädel, Wolfgang Thomas, and Thomas Wilke, editors. *Automata, Logics, and Infinite Games: A Guide to Current Research*, volume 2500 of *Lecture Notes in Computer Science*. Springer, 2002. 10, 11, 14, 15, 26, 45, 88, 109, 110, 121, 134
- [GW99] Erich Grädel and Igor Walukiewicz. Guarded fixed point logic. In *Proceedings of the 4th Annual IEEE Symposium on Logic in Computer Science, LICS'99*, pages 45–54, Los Alamitos, California, July 1999. IEEE Computer Society Press. 108
- [HU69] John E. Hopcroft and Jeffrey D. Ullman. *Formal Languages and their Relation to Automata*. Addison-Wesley, 1969. 13, 72

- [HU79] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison Wesley, 1979. 9, 20
- [Kec94] Alexander S. Kechris. *Classical descriptive set theory*, volume 156 of *Graduate texts in mathematics*. Springer, 1994. 3
- [KNU02] Teodor Knapik, Damian Niwinski, and Pawel Urzyczyn. Higher-order pushdown trees are easy. In *Proceedings of the 5th International Conference Foundations of Software Science and Computation Structures, FOSSACS '02*, volume 2303 of *Lecture Notes in Computer Science*, pages 205–222. Springer, 2002. 5, 7, 19, 103, 104, 117
- [Koz83] Dexter Kozen. Results on the propositional mu-calculus. *Theoretical Computer Science*, 27:333–354, 1983. 109
- [KPV02] Orna Kupferman, Nir Piterman, and Moshe Y. Vardi. Model checking linear properties of prefix-recognizable systems. In *Proceedings of the 14th International Conference Computer Aided Verification, CAV'02*, volume 2404 of *Lecture Notes in Computer Science*, pages 371–385. Springer, 2002. 6
- [KV97] Orna Kupferman and Moshe Y. Vardi. Weak alternating automata are not that weak. In *Proceedings of the Fifth Israel Symposium on Theory of Computing and Systems, ISTCS'97*, pages 147–158, Los Alamitos, California, 1997. IEEE Computer Society Press. 24
- [KV00] Orna Kupferman and Moshe Y. Vardi. An automata-theoretic approach to reasoning about infinite-state systems. In E. A. Emerson and A. P. Sistla, editors, *Proceedings of the 12th International Conference on Computer Aided Verification, CAV'00*, volume 1855 of *Lecture Notes in Computer Science*, pages 36–52. Springer, 2000. 6, 7, 30, 47, 63, 94, 101, 103, 107, 108, 109, 112, 118, 126
- [KV01] Orna Kupferman and Moshe Y. Vardi. On bounded specifications. In *Proceedings of the 8th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning, LPAR'01*, volume 2250 of *Lecture Notes in Computer Science*, pages 24–38. Springer, 2001. 42
- [LT00] Christof Löding and Wolfgang Thomas. Alternating automata and logics over infinite words. In *Proceedings of the IFIP International Confer-*

- ence on Theoretical Computer Science, IFIP TCS2000*, volume 1872 of *Lecture Notes in Computer Science*, pages 521–535. Springer, 2000. [24](#)
- [Löd03] Christof Löding. *Infinite Graphs Generated by Tree Rewriting*. PhD thesis, RWTH Aachen, 2003. [4](#), [20](#)
- [Mar75] Donald A. Martin. Borel Determinacy. *Annals of Mathematics*, 102:363–371, 1975. [12](#)
- [Mor99] Christophe Morvan. On rational graphs. In *Proceedings of the Third International Conference on Foundations of Software Science and Computation Structures, FoSSaCS'99*, volume 1784 of *Lecture Notes in Computer Science*, pages 252–266. Springer, 1999. see also the full version [[Mor00](#)]. [21](#), [139](#)
- [Mor00] Christophe Morvan. On rational graphs. Rapport de recherche 3944, Institut National de Recherche en Informatique et en Automatique (INRIA), Mai 2000. full version of [[Mor99](#)]. [139](#)
- [MS85] David E. Muller and Paul E. Schupp. The theory of ends, pushdown automata, and second-order logic. *Theoretical Computer Science*, 37:51–75, 1985. [13](#), [15](#), [17](#)
- [Rab69] Michael O. Rabin. Decidability of second-order theories and automata on infinite trees. *Transactions of the American Mathematical Society*, 141:1–35, July 1969. [4](#), [105](#)
- [Sch02] Stefan Schwoon. *Model-Checking Pushdown Systems*. PhD thesis, Technische Universität München, 2002. [27](#)
- [Sei96] Sebastian Seibert. Effektive Strategiekonstruktionen für Gale-Stewart-Spiele auf Transitionsgraphen. Technical Report 9611, Institut für Informatik und Praktische Mathematik, Christian-Albrechts-Universität zu Kiel, Germany, July 1996. [63](#)
- [Ser03] Olivier Serre. Note on winning positions on pushdown games with omega-regular conditions. *Information Processing Letter*, 85:285–291, 2003. [87](#)
- [Tho95] Wolfgang Thomas. On the synthesis of strategies in infinite games. In C. Puech E. W. Mayr, editor, *Proceedings of the 12th Annual Symposium*

- on Theoretical Aspects of Computer Science, STACS'95*, volume 900 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 1995. [21](#), [22](#), [25](#), [45](#)
- [Tho97] Wolfgang Thomas. Languages, automata, and logic. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume III, pages 389–455. Springer, New York, 1997. [5](#), [120](#), [125](#)
- [Var98] Moshe Y. Vardi. Reasoning about the past with two-way automata. In Kim Guldstrand Larsen, Sven Skyum, and Glynn Winskel, editors, *Proceedings of the 25th International Colloquium on Automata, Languages and Programming, ICALP'98*, volume 1443 of *Lecture Notes in Computer Science*, pages 628–641, Aalborg, Denmark, 1998. Springer. [6](#), [7](#), [90](#), [94](#), [101](#), [103](#), [119](#), [121](#), [122](#)
- [VJ00] Jens Vöge and Marcin Jurdziński. A discrete strategy improvement algorithm for solving parity games. In *Proceedings of the 12th International Conference on Computer Aided Verification, CAV 2000*, volume 1855 of *Lecture Notes in Computer Science*, pages 202–215. Springer, 2000. [97](#)
- [Wad84] William W. Wadge. *Reducibility and Determinateness on the Baire Space*. PhD thesis, University of California, Berkeley, 1984. [63](#)
- [Wal96a] Igor Walukiewicz. Monadic second order logic on tree-like structures. In *Proceedings of the 13th Annual Symposium on Theoretical Aspects of Computer Science, STACS'96*, volume 1046 of *Lecture Notes in Computer Science*, pages 401–413. Springer, 1996. see also the full version [[Wal02](#)]. [107](#), [126](#), [141](#)
- [Wal96b] Igor Walukiewicz. Pushdown processes: Games and model checking. In *Proceedings of the 8th International Conference on Computer Aided Verification, CAV'96*, volume 1102 of *Lecture Notes in Computer Science*, pages 62–74. Springer, 1996. see also the full version [[Wal01](#)]. [5](#), [7](#), [10](#), [15](#), [27](#), [30](#), [40](#), [63](#), [87](#), [88](#), [89](#), [90](#), [97](#), [120](#), [123](#), [126](#), [129](#), [131](#), [140](#)
- [Wal01] Igor Walukiewicz. Pushdown processes: Games and model checking. *Information and Computation*, 164(2):234–263, January 2001. Full version of [[Wal96b](#)]. [140](#)

- [Wal02] Igor Walukiewicz. Monadic second-order logic on tree-like structures. *Theoretical Computer Science*, 275(1–2):311–346, 2002. full version of [\[Wal96a\]](#). 140
- [Wil01] Thomas Wilke. Alternating tree automata, parity games, and modal μ -calculus. *Bull. Soc. Math. Belg.*, 8(2), May 2001. 110

List of Figures

1.1	The final part of the game graph	3
2.1	Example of a recursive game graph	22
3.1	Automaton from Algorithm 3.1.2 and Example 3.2.1	32
3.2	Alternating automaton recognizing $Attr_0(R)$ for Example 3.3.1	43
3.3	The run DAG of this automaton on the word $a^7\perp$	43
3.4	Automaton from Algorithm 3.4.4 and Example 3.4.5	48
3.5	Automaton from Algorithm 3.4.8 and Example 3.4.5	51
3.6	Automaton from Algorithm 3.4.8 and Section 3.4.3, first generation	55
3.7	Automaton from Algorithm 3.4.8 and Section 3.4.3, second generation	56
3.8	Example of a Büchi game (Section 3.4.4)	57
3.9	Some ordinals (Section 3.4.4)	58
3.10	Automata from Algorithm 3.4.8 and Section 3.4.4	60
3.11	Example (3.5.1) of a simple co-Büchi game	61
3.12	Example of a game on a Prefix-recognizable graph	66
3.13	A simple alternating automaton	74
3.14	Pushdown game of Example 3.7.1, where $u \in \Sigma^*$	75
3.15	Automaton \mathcal{B}'_1 from Algorithm 3.6.2	76
3.16	Automaton \mathcal{B}_1 (simplified)	76
3.17	Automaton \mathcal{B}'_2 from Algorithm 3.6.2	77
3.18	Automaton \mathcal{B}_2 (simplified)	77
3.19	Automaton \mathcal{C}' recognizing $Y_\infty^N \bullet \Gamma^*$	77
3.20	Rough view of Automaton \mathcal{B}'_1 from Algorithm 3.6.2	80
3.21	Rough view of Automaton \mathcal{C}'' recognizing W_0	83
4.1	Example of a simple pushdown game (Section 4.3)	95
4.2	FSP corresponding to the PDGS (Section 4.3)	96
4.3	Example (4.5.2) of a game on a Prefix-recognizable graph	102

4.4	Graph of the corresponding PDGS (Example 4.5.2)	102
5.1	The complete $\{a, b\}$ -tree T_1 obtained by unfolding of a finite graph . .	112
5.2	The same with a “bottom stack symbol”	113
5.3	Graph G_1 for $\Gamma = \{a, b\}$	114
5.4	An initial part of the tree T_2	115
6.1	Summary of the main results	130

Index

Symbols	
$Attr_0(R)$	25
$Attr_0^+$	45, 64
χ	25
Büchi ₀ (R)	45
μ -calculus	109
ϕ	49
$\pi^i(S)$	50
$pre^*(R)$	27
A	
annotation	123
automaton	
\mathcal{P} -	24
B	
Büchi game	11
basic notations	9
C	
Caucal Hierarchy	106
co-Büchi game	59
configuration	14, 105
controllable predecessors	25
controller	13
counter	78
D	
determined	12
E	
environment	13
F	
Finite State Parity game	89
G	
game	10
game graph	10
game structure	10
game-simulated	88
goal set	25
graph	105
graph automaton	107
H	
Higher Order Pushdown Game System	105
Higher Order Pushdown Graphs	19
L	
level n store	104
level 1 store	104
M	
model-checking	15
model-checking game	14, 111
monadic second-order logic	15
O	
one-way	124
P	
parity game (structure)	11
parity game structure on a HPDGS H	105

path	106
play	10
player	
existential	111
universal	111
positional	12
positional strategy	17
Prefix-Recognizable Graph	98
PRG	98
priority function	88
pushdown game graph	14
Pushdown Game System (PDGS) ..	13
pushdown strategy	40, 89
 R _____	
rank	32
reachability game	10
 S _____	
safety	59
solution	12
strategy	12, 121
substitution	106
 T _____	
tower of exponentials	126
transition system	15
transition systems	109
tree	106
 U _____	
unfolding	106
uniform solution	12
 W _____	
winning conditions	10
winning region	12
winning strategy	12
wins the game from	12

Curriculum Vitae

Zur Person

Name: Thierry Cachat
geboren: 6. Juni 1975 in Montélimar, Drôme, Frankreich
Staatsangehörigkeit: französisch

Bildungsgang

1981-1986 Grundschule in Valaurie (Drôme)
1986-1990 „collège“ (Gymnasium 1. Teil) Gérard de Nerval, Pierrelatte
1990-1993 „lycée“ (Gymnasium 2. Teil) Gustave Jaume, Pierrelatte (Drôme)
1993 „baccalauréat“ (Abitur) in Montélimar (Drôme)

1993-1995 „Mathématiques supérieures“ und „spéciales M“,
lycée Clemenceau, Nantes

1995-1997 Diplom von „licence“ und dann „maîtrise“ in Mathematik an der
Universität Grenoble-1

1997-1998 Eintritt in die „École Normale Supérieure“ von Cachan
1998-1999 zugelassen zur „agrégation“ in Mathematik
1999-2000 „diplôme d'études approfondies (DEA)“ in Informatik
an der Universität Rennes-1, Titel der Diplomarbeit:
Rationalité du produit d'ensembles rationnels d'entiers

2000-2003 wissenschaftlicher Mitarbeiter am Lehrstuhl für Informatik VII
(Prof. Dr. Wolfgang Thomas)
Rheinisch-Westfälische Technische Hochschule Aachen

2003-2004 „attaché temporaire d'enseignement et de recherche (ATER)“
(Wissenschaftlicher Mitarbeiter)
am LSV, École Normale Supérieure de Cachan