UMONS - Université de Mons
Faculty of Science
Computer Science Department

Antichain based algorithms for the synthesis of reactive systems

Aaron Bohy

A dissertation submitted in fulfillment of the requirements of
the degree of *Docteur en Sciences*

Advisors

Pr. Thomas Brihaye
Université de Mons, Belgium

Pr. Véronique Bruyère
Université de Mons, Belgium

Pr. Jean-François Raskin
Université Libre de Bruxelles, Belgium

Jury

Pr. Nathalie Bertrand
INRIA Rennes Bretagne-Atlantique, France

Pr. Thomas Brihaye
Université de Mons, Belgium

Pr. Véronique Bruyère
Université de Mons, Belgium

Pr. Emmanuel Filiot
Université Libre de Bruxelles, Belgium

Pr. Hadrien Mélot
Université de Mons, Belgium

Pr. Jean-François Raskin
Université Libre de Bruxelles, Belgium

May 2014

Acknowledgments

Prior to this thesis, I would like to express my gratitude to all the people who made it possible.

First and foremost, I would like to thank my advisors Thomas Brihaye, Véronique Bruyère and Jean-François Raskin for welcoming me in their team and guiding me through this thesis with wise advices, encouragements and enthusiasm. Working with you during these four years has been an extremely rewarding and enjoyable experiment.

I am also thankful to Emmanuel Filiot who co-authored two conference articles on which a part of this manuscript is based. I am especially grateful for the time you devoted to me when I was taking my first steps through this thesis.

I thank all the members of this thesis committee for the time spent reading this document.

My last purely scientific acknowledgments go to fellow scientists for fruitful discussions. In alphabetical order, I thank Julie De Pril, Marc Ducobu, Olivier Gauwin, Nicolas Maquet, Noémie Meunier, Youssouf Oualhadj and Mickael Randour.

Finally, in this last paragraph, I would like to thank all the people supporting me in my personal life. I want to express my gratitude to my family and friends, from and outside the university, for their continuous support. Last but not least, I would like to thank Marie for being by my side every day. Thank you for those moments of joy, and all those, even more wonderful, ahead of us...

Abstract

Formal verification is a branch of computer science which aims to provide formal proofs of correctness of computer systems. Nowadays, the research in formal verification is of prime importance due to the ubiquity of computer systems in our daily life. Model checking and synthesis are two main disciplines of formal verification. Given a formal model of a system and a formal specification of the desired properties for the system, *model checking* consists in automatically checking whether the model satisfies the specification. *Synthesis* amounts to automatically constructing the system from its specification.

In this thesis, we focus on the synthesis of *reactive systems*, that is, systems that have to react to events issued from their environment. We use the framework of *game theory* which provides strong mathematical tools for this task. We study several synthesis problems for which we provide algorithmic solutions using the symbolic data structure of *antichains*, and we validate our solutions with open-source implementations in tools. We consider two frameworks of synthesis.

On the one hand, we consider the synthesis in the *worst-case*. In this framework, the environment is purely antagonistic and *all* executions of the system must satisfy the given specification. We study the well-known problem of synthesis from specifications expressed in linear temporal logic (LTL). LTL specifications are purely *qualitative*: a system either conforms to them, or violates them. We extend the qualitative setting of LTL to a *quantitative* one by introducing the problems of synthesis from LTL specifications with mean-payoff or energy objectives.

On the other hand, we consider the synthesis of reactive systems with *good expected performance*. In this framework, the environment is not antagonistic,

but it is rather *stochastic*. We study *symblicit* algorithms, that is, algorithms that efficiently combine explicit and symbolic data representations. For that purpose, we introduce a new data structure extended from antichains, called *pseudo-antichain*. We propose two practical applications of our symblicit algorithms coming from automated planning and synthesis from LTL specifications with mean-payoff objectives.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

For several decades, computer systems have been more and more ever-present in our daily life. Nowadays, they are everywhere: smartphones, cars (airbag, ABS, cruise control...), air traffic control systems, online banking, *e*-commerce, medical devices... Moreover, when those systems are embedded in a larger environment, their complexity grows, what makes them more vulnerable to errors. Such systems are called *reactive systems* since they always have to react to events issued from their environment.

In many applications, the failure of reactive systems is unacceptable, mainly for two reasons. The first one is safety. Flaws or errors (commonly called *bugs*) in *safety-critical* systems like nuclear power plants and autopilots in aircrafts or rockets might have catastrophic consequences. A notorious example is the explosion of the rocket Ariane-5 in 1996, less than 40 seconds after its launch, due to a conversion of a 64-bit floating point number to a 16-bit signed integer in its guidance system. Another example happened in medicine. Between 1985 and 1987, at least six cancer patients died of radiation overdose due to software bugs in the control part of the radiation therapy machine Therac-25. The second reason is money. Indeed, a marketed product with bugs can have dramatic financial consequences for the manufacturer. The probably most known example is the floating-point division bug found in 1994 in Intel's Pentium II after its commercialization: it cost Intel $475 million to replace the affected processors.

Clearly, there is a need of techniques and tools to verify the correctness of systems despite their complexity. In software engineering, the traditional verification technique is *testing*, and almost 50% of the time allocated to a software project is devoted to the tests. For hardware verification, the most popular technique is *simulation*, and around 75% of the time spent on a chip is dedicated to error detection and prevention. However, testing and simulation are never complete: they only explore *some* behaviors and scenarios of the system. They can thus only detect the presence of bugs, not their absence. Moreover, detecting subtle bugs with those techniques may be very difficult. For many applications, testing and simulation are not enough anymore, and formal proofs of correctness are needed. A branch of theoretical computer science called *formal verification* provides solutions to these problems. In particular, this is the case of *model checking*.

Model checking. Model checking [CGP01, BK08] is a fully automated formal verification technique for proving correctness of software and hardware systems. Model checking requires two inputs: a model of the system under consideration, often expressed with a finite-state automaton, and a desired property (e.g. absence of deadlock) called the *specification*. It then performs an exhaustive exploration of all possible behaviors of the model and checks if those behaviors all satisfy the given specification. In the negative case, a counter-example (i.e. some behavior that violates the specification) is output for debugging purposes. In the early eighties, Clarke and Emerson [CE81], and Queille and Sifakis [QS82] independently developed model checking methods where the specifications are expressed in a propositional temporal logic. Temporal logic is an appropriate formalism to express specifications of reactive systems since it permits to describe the ordering of events in time. It allows for the specification of many system properties like reachability (*is it possible for the system to reach a deadlock situation?*), safety (*does something bad ever happen?*), liveness (*does something good eventually happen?*), fairness (*does an event occur repeatedly?*)...

A major challenge in model checking is to be able to examine the largest possible state space in order to verify systems that are more and more complex. At the beginning, model checking techniques only worked for small systems with few states. However, when the system under consideration is composed of

many sub-systems, the number of states grows rapidly. This is the manifestation of the well-known *state explosion problem*. Explicit enumeration of states and transitions then becomes infeasible. Symbolic data representations provide an alternative solution. In the early nineties, McMillan et al. [BCM+92, McM93] considered for the first time *symbolic model checking*. They used a symbolic data structure, based on the *binary decision diagrams (BDDs)* of Bryant [Bry86], to represent the finite-state automaton, and they were able to verify models of more than 10^{20} states. Moreover, improvements proposed in [BCL91] allowed to verify models of more than 10^{120} states. Other techniques like *partial-order reduction* [God90, Pel96, Val92] and *abstraction* [CGL94, CGJ+00] have also been successfully used in model checking to reduce the size of the state space. Nowadays, thanks to those enhancements of algorithms and data structures, together with hardware technology improvements, model checking techniques are applicable to realistic designs. As a matter of fact, efficient tools have been developed for industrial use like UPPAAL [LPY97], NuSMV [CCGR00] and SPIN [Hol04].

Synthesis. The main drawback of model checking, as for all verification techniques, is that the verification task is performed after many resources have been invested in the development of the system. One of the most ambitious challenge in computer science is the automated *synthesis* of systems, that is, the task of automatically deriving a system from its desired specifications. The produced system has no longer to be verified and no more resources are spent in debugging: the system is correct by construction! The problem of synthesis was stated in 1962 by Church [Chu62] and first solutions were proposed by Büchi and Landweber [BL69] and Rabin [Rab72]. In these pioneer studies, the specifications are expressed in monadic second order logic and the complexity of synthesis is non-elementary. From the end of the eighties, Pnueli and Rosner [PR89a], and Abadi, Lamport and Wolper [ALW89] studied for the first time the problem of synthesis from specifications expressed in linear temporal logic (LTL), that is, the *LTL synthesis problem*. This problem was proved to be 2ExpTime-complete [Ros92] and its classical solution is based on a costly translation from LTL to deterministic Rabin automata [PR89b]. Due to its high worst-case complexity, the LTL synthesis problem has been considered for a long time only of theoretical interest. Only recently, several progresses on algorithms and efficient data structures

showed that they can also be solved in practice.

Game theory. Game theory provides the mathematical tools needed to perform the tasks of model checking and synthesis. *Game theory* is a mathematical framework widely used in various disciplines such as biology, economics, political science, psychology, and in particular in computer science and logic (e.g. [Rab69, GH82, MS85, EJ91, BDBK+94]). The study of games originates from the work of Zermelo [Zer13] and Borel [Bor21], or even a century earlier with the work of Cournot [Cou38]. The foundations of modern game theory have been laid in 1944 by von Neumann and Morgenstern [vNM44] in the book *"Theory of Games and Economic Behavior"*. Since then, game theory has been widely studied and extensively developed.

Using games in verification. Games model interactions between players who try to achieve individual or common objectives. Game theoretic methods are central in formal verification of reactive systems. Games are used to model the interactions between the system under consideration (referred to as Player 1) and its surrounding environment (referred to as Player 2). Such games are played on directed graphs by two players and last forever. Each vertex is controlled by exactly one player in the sense that when the game is in a given vertex, the player which controls it chooses the next vertex among the available successors. A player selects the next vertex according to a strategy. The outcome of the game is an infinite path through the graph.

When considering *Boolean* objectives (called *qualitative* objectives in the sequel), the outcome of the game is either winning or losing for each player. For a given player, a strategy is said winning if, by playing according to this strategy, whatever the strategy of the adversary is, the outcome of the game is winning for the considered player. Winning outcomes are expressed with qualitative ω-regular objectives (e.g. parity, Büchi, Rabin...), that is, objectives such that the sets of winning outcomes are ω-regular [Tho97]. An important result in game theory is the determinacy of 2-player games with ω-regular objectives [Mar75] (if one player does not have a winning strategy for the given objective, then the other player has a winning strategy for the complementary objective). In the framework of games with qualitative objectives, checking that a reactive system

satisfies a given specification amounts to finding a winning strategy for Player 1 in the related game. Finally, the synthesis problem is to construct such a winning strategy.

Quantitative specifications. The previously described game theoretic framework played on graphs can be enriched with *quantitative* aspects. In the quantitative setting, the edges and/or the vertexes of the game graph are labeled with real valued weights. Measurable functions, like the total sum of the weights (total-payoff) or the long-run average of the weights (mean-payoff), assign real values to infinite paths in the graph, instead of Boolean values as in the qualitative case. In those games, players have to optimize this real value. Quantitative games have been extensively studied for various quantitative objectives such as mean-payoff (e.g. [EM79, ZP96]), total-payoff (e.g. [GZ04, GS09]) and energy (e.g. [CdAHS03, BFL+08]).

Using quantitative games in verification of reactive systems allows to quantify how good a system is with respect to the given specification. For instance, in the context of systems granting a shared resource between several clients, a natural specification is that *every request must be eventually granted*. Automated synthesis may produce undesirable, even though correct, systems for which the delay between requests and grants is high. A quantitative formulation of this specification could be to *minimize the average delay between requests and grants*. Such a specification would favor the synthesis of realistic and efficient systems.

For all the games presented above, either restricted to qualitative objectives or extended with quantitative aspects, the environment is purely antagonistic to the system in the sense that a strategy for the system is winning if, for all strategies of the environment, the outcome of the game is winning. For that reason, we call such winning strategies *worst-case winning strategies*, that is, even in the worst execution scenario of the system, the qualitative or quantitative specification is satisfied.

Stochastic environments. In contrast to purely antagonistic environments, it is also interesting to consider not strictly adversarial ones to have better models of real situations. In this case, the environments are rather stochastic [Var85, CY95]. Game theory provides the mathematical tools for this task. Stochas-

tic games were introduced in the early fifties by Shapley [Sha53]. In stochastic games, actions performed by players at each vertex determine a probability distribution on the successor vertexes, instead of determining a single successor. Stochastic games have been widely investigated in verification (e.g. [HSP83, HS84, PZ86, CY88]). An example of stochastic games is the Markov decision process (MDP) [Put94]. MDPs can be seen as 2-player games in which Player 2 is not antagonistic, but instead plays according to probability distributions. Such models allow one to verify qualitative specifications like *does some property holds with probability* 0 *or* 1*?*, and quantitative specifications like *what is the exact probability that some property holds?*. Moreover, when MDPs are equipped with weights, a common approach is to compute the strategies of Player 1 that offer the best expectation of a given measurable function. Examples of classical functions studied in the context of MDPs are the mean-payoff [Put94, FV97], the total-payoff [Put94, FV97] and the shortest path [BT91, dA99]. We call the strategies computed in this framework *good-for-expectation strategies*, that is, strategies that ensure a good expected performance.

Data structures. Adequate data structures are required for the efficient implementation of complex algorithms. Recently, model checking and synthesis tools have experienced major improvements in performance. These gains of performance are the result of a better understanding of classical formal verification problems, giving rise to efficient heuristics and careful implementations using adequate data structures. On the one hand, as already mentioned, symbolic data structures like binary decision diagrams (BDDs) [Bry86] allow to mitigate the state explosion problem. Moreover, the data structure of multi-terminal binary decision diagrams (MTBDDs) [CMZ+93, FMY97], a natural extension of BDDs, is used for the efficient manipulation of stochastic models. On the other hand, *antichains* have also been considered as an alternative to BDD like structures. Antichain based algorithms have proved worthy for solving classical problems in game theory, but also in logic and automata theory (e.g. [DDHR06, DR07, BHH+08, DR10]).

Structure and contributions of the thesis. We now present the organization of the thesis and the contributions of each chapter.

In Chapter 2, we recall the basic definitions and some known results used throughout the thesis. In Chapter 3, we summarize our contributions and we give pointers to related work. The complete presentation of our contributions through the remaining chapters follows a subdivision into three parts.

Part I - Data structures. This part is devoted to data structures, and in particular antichain like structures. In Chapter 4, we recall the notion of antichains and we present a new data structure extended from antichains, called pseudo-antichain, introduced for the first time in [BBR14b]. This new structure is used by our algorithms presented in Part III, for which antichains are not sufficiently expressive. We also briefly present the public library AaPAL [AaP] that we developed for the manipulation of antichains and pseudo-antichains.

Part II - Synthesis of worst-case winning strategies. In this part, we consider the framework of worst-case winning strategy synthesis. In Chapter 5, we consider the problem of synthesis from LTL specifications. In [FJR11], Filiot et al. present an efficient antichain based Safraless procedure for solving the LTL synthesis problem. We have implemented this procedure in a new tool called Acacia+ [BBF+12, Aca] made available for the research community. We first recall the implemented procedure and we then present Acacia+, together with several application scenarios and experimental results. In Chapter 6, we extend the purely qualitative setting of LTL to a quantitative setting. We introduce the problems of synthesis from LTL specifications with secondary mean-payoff or energy objectives [BBFR13]. We provide complexity results, algorithmic antichain based solutions, and a tool for the automated synthesis of reactive systems from quantitative specifications. Promising experimental results are also reported.

Part III - Synthesis of good-for-expectation strategies. In this third part, based on [BBR14b], we consider the framework of good-for-expectation strategy synthesis. In a recent work, Wimmer et al. [WBB+10] have proposed a so-called *symblicit* algorithm for the mean-payoff in MDPs. This algorithm efficiently combines symbolic and explicit data structures, and uses (MT)BDDs as symbolic representation. In Chapter 7, we show that the new data structure of pseudo-antichains provides an interesting alternative to BDDs, especially for the class of monotonic MDPs. We design efficient pseudo-antichain based symblicit algorithms (together with open-source implementations) for two quantitative

settings: the mean-payoff and the shortest path. In Chapter 8, we show that monotonic MDPs naturally arise in probabilistic planning and when optimizing systems synthesized from LTL specifications with mean-payoff objectives. For those two practical applications, we report promising experimental results of our implementations with respect to both the run time and the memory consumption.

Finally, we conclude this thesis with Chapter 9 by listing some possible extensions of our work.

CHAPTER 2

Preliminaries

In this chapter, we introduce the basic notions used throughout this thesis and we recall some known results. We tackle the notions of binary relations, partially ordered sets and lattices. We define the syntax and semantics of linear temporal logic (LTL), we consider infinite word automata, and we show that there is a tight relationship between linear temporal logic and infinite word automata. We also recall notions related to games and some known results about them. Finally, we introduce stochastic models.

2.1 Sets, functions, relations and lattices

In this section, we fix some notations and vocabulary related to sets and functions. We also consider binary relations, and more precisely equivalence relations and partial orders, and we recall notions about lattices.

Sets and functions. We denote by $\mathbb{N}$ the set of natural numbers, by $\mathbb{Z}$ the set of integers, by $\mathbb{Q}$ the set of rational numbers, and by $\mathbb{R}$ the set of real numbers. We use standard subscript notations to denote the subsets of negative or positive numbers of those sets. For instance, we denote by $\mathbb{R}_{>0}$ the set of strictly positive real numbers.

For any set S, we denote by $|S| \in \mathbb{N} \cup \{+\infty\}$ the *cardinality* of S and by

$2^S = \{P \mid P \subseteq S\}$ the *power set* of S, that is, the set of subsets of S. The *empty set* is denoted by $\emptyset$. Given two sets S_1 and S_2, we denote by $S_1 \uplus S_2$ the *disjoint union* of S_1 and S_2. We denote by S^* (resp. S^ω) the set of finite (resp. infinite) sequences over S. For any infinite sequence $u \in S^\omega$, we denote by $\mathsf{Inf}(u) \subseteq S$ the set of elements of S that appear infinitely often in u.

For any function $f : X \to Y$, we denote by $\mathsf{Dom}(f)$ the *domain of definition* of f, that is, $\mathsf{Dom}(f) = \{x \in X \mid f(x) \text{ is defined}\}$, and by $\mathsf{Img}(f)$ the *image* of f, that is, $\mathsf{Img}(f) = \{f(x) \mid x \in \mathsf{Dom}(f)\}$. A function $f : X \to Y$ is *injective* if for all $x_1, x_2 \in \mathsf{Dom}(f)$, $f(x_1) = f(x_2)$ implies $x_1 = x_2$. It is *surjective* if $\mathsf{Img}(f) = Y$. A *bijection* is a function that is both injective and surjective. Given an injective function $f : X \to Y$, we denote by $f^{-1} : Y \to X$ the *inverse* of f, that is, the function defined as follows. For all $x \in \mathsf{Dom}(f)$ and all $y \in \mathsf{Img}(f)$, $f^{-1}(y) = x$ if and only if $f(x) = y$. A function $f : X \to Y$ is a *total function* if $\mathsf{Dom}(f) = X$. Otherwise, f is a *partial function*. For all sets X, Y, we denote by $\mathcal{F}_{\text{tot}}(X, Y) = \{f : X \to Y \mid \mathsf{Dom}(f) = X\}$ the set of total functions from X to Y.

Binary relations. A *binary relation* on a set S is a set of pairs $R \subseteq S \times S$. A binary relation R on S is *reflexive* if and only if for all $s \in S$, it holds that $(s, s) \in R$. It is *symmetric* if and only if for all $s_1, s_2 \in S$, if $(s_1, s_2) \in R$, then $(s_2, s_1) \in R$. It is *antisymmetric* if and only if for all $s_1, s_2 \in S$, if $(s_1, s_2) \in R$ and $(s_2, s_1) \in R$, then $s_1 = s_2$. It is *transitive* if and only if for all $s_1, s_2, s_3 \in S$, if $(s_1, s_2) \in R$ and $(s_2, s_3) \in R$, then $(s_1, s_3) \in R$. Finally, it is *total* if and only if for all $s_1, s_2 \in S$, $(s_1, s_2) \in R$ or $(s_2, s_1) \in R$. Binary relations are often denoted by non-alphabetical symbols. Given a binary relation $\sim$ on S and $s_1, s_2 \in S$, we often write $s_1 \sim s_2$ for $(s_1, s_2) \in \sim$ and $s_1 \not\sim s_2$ for $(s_1, s_2) \notin \sim$.

Orders and equivalence relations. A *preorder* is a binary relation that is reflexive and transitive. An *equivalence relation* is a preorder that is symmetric. A *partial order* is a preorder that is antisymmetric. We often denote equivalence relations by $\sim$ and partial orders by $\preceq$ (with or without subscripts). Finally, a *total order* is a partial order that is total.

Given an equivalence relation $\sim$ on S, an *equivalence class* is a subset $T \subseteq S$ such that for all $t_1, t_2 \in T$ and all $t_3 \in S \setminus T$, it holds that $t_1 \sim t_2$ and $t_1 \not\sim t_3$.

An equivalence relation $\sim$ on S induces a *partition* $S_\sim$ of S such that each *block* $B \in S_\sim$ is an equivalence class of $\sim$. The *size* of an equivalence relation $\sim$ is its total number of equivalence classes $|S_\sim|$. Given a function $\cdot : S \times S \to S$, an equivalence relation $\sim$ on S is a *right congruence* if, for all $s, s_1, s_2 \in S$, $s_1 \sim s_2$ implies $s_1 \cdot s \sim s_2 \cdot s$. We say that a right congruence $\sim$ is *of finite index* if $|S_\sim| < \infty$.

Given a partial order $\preceq \subseteq S \times S$ and $s_1, s_2 \in S$, we say that s_1 and s_2 are *comparable* if $s_1 \preceq s_2$ or $s_2 \preceq s_1$. Otherwise, we say that s_1 and s_2 are *incomparable*. Moreover, if $s_1 \preceq s_2$, we say that s_1 is *less than or equal to* s_2 and that s_2 is *greater than or equal to* s_1.

Example 2.1. Let $\mathbb{N}^2$ be the set of pairs of natural numbers and let $\preceq \subseteq \mathbb{N}^2 \times \mathbb{N}^2$ be a binary relation such that for all $(n_1, n_1'), (n_2, n_2') \in \mathbb{N}^2$, $(n_1, n_1') \preceq (n_2, n_2')$ if and only if $n_1 \leq n_2$ and $n_1' \leq n_2'$. We have that $\preceq$ is reflexive, transitive and antisymmetric, that is, $\preceq$ is a partial order. $\lhd$

Lattices. A *partially ordered set* is a pair $\langle S, \preceq \rangle$ where S is a set and $\preceq \subseteq S \times S$ is a partial order on S. Let $\langle S, \preceq \rangle$ be a partially ordered set and $T \subseteq S$. The *greatest lower bound* of T is the greatest element of S that is less than or equal to all elements of T. Dually, the *least upper bound* of T is the least element of S that is greater than or equal to all elements of T. Given two elements $s_1, s_2 \in S$, we denote by $s_1 \sqcap s_2 \in S$ their greatest lower bound, that is, $s_1 \sqcap s_2 \preceq s_1$, $s_1 \sqcap s_2 \preceq s_2$ and for all $s' \in S$ such that $s' \preceq s_1$ and $s' \preceq s_2$, $s' \preceq s_1 \sqcap s_2$. Similarly, we denote by $s_1 \sqcup s_2 \in S$ the least upper bound of s_1 and s_2. A *lower semilattice* (resp. an *upper semilattice*) is a partially ordered set $\langle S, \preceq \rangle$ such that, for every pair of elements $s_1, s_2 \in S$, their greatest lower bound $s_1 \sqcap s_2$ (resp. their least upper bound $s_1 \sqcup s_2$) exists. A *complete lattice* is a partially ordered set that is both a lower and an upper semilattice with respect to the same partial order.

Example 2.2. Let $\preceq \subseteq \mathbb{N}^2 \times \mathbb{N}^2$ be the partial order of Example 2.1. The partially ordered set $\langle \mathbb{N}^2, \preceq \rangle$ is a complete lattice with least upper bound $\sqcup$ such that $(n_1, n_1') \sqcup (n_2, n_2') = (\max(n_1, n_2), \max(n_1', n_2'))$, and greatest lower bound $\sqcap$ such that $(n_1, n_1') \sqcap (n_2, n_2') = (\min(n_1, n_2), \min(n_1', n_2'))$. $\lhd$

2.2 Linear temporal logic

In this section, we recall the syntax and semantics of *linear temporal logic (LTL)*, which is a propositional logic extended by temporal operators. LTL is an adequate formalism to express properties of reactive systems. This section is inspired from [BK08, Chapter 5].

Syntax. LTL formulas are defined over a finite set P of *atomic propositions*. The syntax of LTL is defined by the following grammar:

$$\phi \ ::= \ \mathsf{true} \mid p \mid \phi \wedge \phi \mid \neg\phi \mid \mathsf{X}\phi \mid \phi\mathsf{U}\phi$$

where $p \in P$. The above definition introduces *logical operators* $\wedge$ (*conjunction*) and $\neg$ (*negation*), and *temporal operators* X (*next*) and U (*until*). The set of logical operators is extended with operators $\vee$ (*disjunction*), $\Rightarrow$ (*implication*) and $\Leftrightarrow$ (*equivalence*) derived from $\wedge$ and $\neg$ such that:

$$\begin{array}{lcl} \phi_1 \vee \phi_2 & \stackrel{\mathsf{def}}{=} & \neg(\neg\phi_1 \wedge \neg\phi_2) \\ \phi_1 \Rightarrow \phi_2 & \stackrel{\mathsf{def}}{=} & \neg\phi_1 \vee \phi_2 \\ \phi_1 \Leftrightarrow \phi_2 & \stackrel{\mathsf{def}}{=} & (\phi_1 \Rightarrow \phi_2) \wedge (\phi_2 \Rightarrow \phi_1) \end{array}$$

The set of temporal operators is extended with operators $\Diamond$ (*eventually*) and $\Box$ (*always*) derived from U as follows:

$$\begin{array}{lcl} \Diamond\phi & \stackrel{\mathsf{def}}{=} & \mathsf{true}\mathsf{U}\phi \\ \Box\phi & \stackrel{\mathsf{def}}{=} & \neg\Diamond\neg\phi \end{array}$$

We also derive the constant $\mathsf{false} \stackrel{\mathsf{def}}{=} \neg\mathsf{true}$. The *length* of an LTL formula ϕ, denoted $|\phi|$, is the total number of operators it contains.

Semantics. An *evaluation* over the set of atomic propositions P is a set $\sigma \in 2^P$. An evaluation σ over P identifies a *truth assignment* of P: propositions in σ are assigned to true and propositions in $P \setminus \sigma$ are assigned to false. LTL formulas ϕ are interpreted on infinite sequences of evaluations, that is, *infinite words* $u = \sigma_0\sigma_1\sigma_2\cdots \in (2^P)^\omega$, via a *satisfaction relation* $u \models \phi$ inductively defined as follows:

$$
\begin{array}{lll}
u \models \mathsf{true} & & \\
u \models p & \text{iff} & p \in \sigma_0 \\
u \models \phi_1 \wedge \phi_2 & \text{iff} & u \models \phi_1 \text{ and } u \models \phi_2 \\
u \models \neg\phi & \text{iff} & u \not\models \phi \\
u \models \mathsf{X}\phi & \text{iff} & \sigma_1\sigma_2 \ldots \models \phi \\
u \models \phi_1 \mathsf{U} \phi_2 & \text{iff} & \exists i \geq 0, \sigma_i\sigma_{i+1} \ldots \models \phi_2 \text{ and} \\
 & & \forall j, 0 \leq j < i, \sigma_j\sigma_{j+1} \ldots \models \phi_1
\end{array}
$$

LTL formulas with derived temporal operators $\Diamond$ and $\Box$ have the expected semantics:

$$
\begin{array}{lll}
u \models \Diamond\phi & \text{iff} & \exists i \geq 0, \sigma_i\sigma_{i+1} \ldots \models \phi \\
u \models \Box\phi & \text{iff} & \forall i \geq 0, \sigma_i\sigma_{i+1} \ldots \models \phi
\end{array}
$$

Given an LTL formula ϕ and an infinite word $u \in (2^P)^\omega$, we say that u *satisfies* ϕ if $u \models \phi$. Otherwise, we say that u *falsifies* ϕ and we write $u \not\models \phi$. We denote by $[\![\phi]\!]$ the set of words that satisfy ϕ, that is, $[\![\phi]\!] = \{u \in (2^P)^\omega \mid u \models \phi\}$. Two LTL formulas ϕ_1 and ϕ_2 are *equivalent*, written $\phi_1 \equiv \phi_2$, if and only if $[\![\phi_1]\!] = [\![\phi_2]\!]$.

Example 2.3. Let $P = \{a, b\}$ be a set of atomic propositions. We illustrate each temporal operator with an example and a counter-example:

$$
\begin{array}{llllll}
\{a\}\{b\}\{b\}\{a\}^\omega & \models \mathsf{X}b & \text{and} & \{a\}\boldsymbol{\{a\}}\{b\}\{a\}^\omega & \not\models \mathsf{X}b \\
\{a\}\{a\}\{b\}\{a\}^\omega & \models a\mathsf{U}b & \text{and} & \{a\}\boldsymbol{\{\}}\{b\}\{a\}^\omega & \not\models a\mathsf{U}b \\
\{a\}\{a\}\{b\}\{a\}^\omega & \models \Diamond b & \text{and} & \boldsymbol{\{a\}^\omega} & \not\models \Diamond b \\
(\{b\}\{ab\})^\omega & \models \Box b & \text{and} & \{b\}\{ab\}(\{b\}\boldsymbol{\{a\}})^\omega & \not\models \Box b
\end{array}
$$

◁

2.3 Infinite word automata

This section is devoted to infinite word automata, that is, finite automata that interpret infinite words. We detail several acceptance conditions and we state some known results about LTL and infinite word automata. We first recall the notions of alphabet, word and language.

Alphabets, words and languages. An *alphabet* is a finite set Σ of elements that we call *letters*. A *finite word* u over an alphabet Σ is a finite sequence

of letters $u = \sigma_0\sigma_1 \dots \sigma_n \in \Sigma^*$. Given a finite word $u = \sigma_0\sigma_1 \dots \sigma_n$, we denote by $|u| = n + 1$ the *length* of u. The word of length 0 is called the *empty word* and is denoted by ϵ. An *infinite word* u over Σ is an infinite sequence $u = \sigma_0\sigma_1 \dots \in \Sigma^\omega$. The length of an infinite word u is denoted by $|u| = \infty$. For all $n \in \mathbb{N}$, the *prefix of length* n of an infinite word $u = \sigma_0\sigma_1 \cdots \in \Sigma^\omega$ is the finite word $u(n) = \sigma_0 \dots \sigma_{n-1} \in \Sigma^*$. A *finite word language* $\mathcal{L}$ is a (finite or infinite) set of finite words, that is $\mathcal{L} \subseteq \Sigma^*$. An *infinite word language* $\mathcal{L}$ is a (finite or infinite) set of infinite words, that is $\mathcal{L} \subseteq \Sigma^\omega$. Given a finite word language $\mathcal{L} \subseteq \Sigma^*$, we denote by $\overline{\mathcal{L}}$ its *complement*, that is, $\overline{\mathcal{L}} = \Sigma^* \setminus \mathcal{L}$. The definition and notation of complement naturally extend to infinite word languages.

Infinite word automata. Unlike classical automata, that run on finite words, infinite word automata run on infinite words. We start with their formal definition.

Definition 2.4 (Infinite word automata). An *infinite word automata* over the finite alphabet Σ is a tuple $\mathcal{A} = (\Sigma, Q, q_0, \Omega, \delta)$ such that:

- Q is a finite set of states,
- $q_0 \in Q$ is the initial state,
- $\Omega \subseteq Q^\omega$ is the acceptance condition (several acceptance conditions are described below), and
- $\delta : Q \times \Sigma \to 2^Q$ is the transition function.

We let $|\mathcal{A}| = |Q| + |\delta|$, where $|\delta|$ is the total number of transitions in $\mathcal{A}$, be the *size* of $\mathcal{A}$. We say that $\mathcal{A}$ is *deterministic* if for all $q \in Q$ and all $\sigma \in \Sigma$, $|\delta(q, \sigma)| \leq 1$. Otherwise, we say that $\mathcal{A}$ is *nondeterministic*. We say that $\mathcal{A}$ is *complete* if for all $q \in Q$ and all $\sigma \in \Sigma$, $\delta(s, \sigma) \neq \emptyset$. Unless otherwise stated, we always assume that the automata are complete. A *run* of $\mathcal{A}$ on a word $u = \sigma_0\sigma_1 \cdots \in \Sigma^\omega$ is an infinite sequence of states, or *infinite path*, $\rho = \rho_0\rho_1 \cdots \in Q^\omega$ such that $\rho_0 = q_0$ and $\forall k \geq 0, \rho_{k+1} \in \delta(\rho_k, \sigma_k)$. We denote by $\mathsf{Runs}_\mathcal{A}(u)$ the set of runs of $\mathcal{A}$ on u. Note that given a deterministic complete automaton $\mathcal{A}$, we have that $|\mathsf{Runs}_\mathcal{A}(u)| = 1$ for all $u \in \Sigma^\omega$. A state $q \in Q$ is *reachable* from the initial state if there is a sequence $q_0q_1 \dots q_n \in S^*$ of states such that $q_n = q$ and for all i, $0 \leq i < n$, there exists $\sigma_i \in \Sigma$, $q_{i+1} \in \delta(q_i, \sigma_i)$.

Finally, we denote by $\mathsf{Visit}(\rho, q)$ the number of times the state q occurs along the run ρ.

Acceptance conditions. Several *acceptance conditions* (e.g. Büchi [Büc62], Muller [Mul63], Rabin [Rab69], Streett [Str82], Parity [Mos84]...) coexist for infinite word automata. Let $\mathcal{A} = (\Sigma, Q, q_0, \Omega, \delta)$ be an infinite word automaton. Those conditions determine the set of runs that are *accepting* in $\mathcal{A}$. When an acceptance condition Ω is associated with an infinite word automaton $\mathcal{A}$, we say that $\mathcal{A}$ is an Ω *automaton*. For instance, if Ω is the Büchi acceptance condition, we say that $\mathcal{A}$ is a *Büchi automaton*. We formally define four acceptance conditions.

Büchi. The Büchi acceptance condition is given by a subset $\Omega = \alpha$ with $\alpha \subseteq Q$ of states called *final states*. A run $\rho \in Q^\omega$ is accepting in a *Büchi automaton* $\mathcal{A}$ if and only if $\mathsf{Inf}(\rho) \cap \alpha \neq \emptyset$.

Co-Büchi. Like the Büchi acceptance condition, the co-Büchi one is given by a subet $\Omega = \alpha$ with $\alpha \subseteq Q$ of final states. A run $\rho \in Q^\omega$ is accepting in a *co-Büchi automaton* $\mathcal{A}$ if and only if $\mathsf{Inf}(\rho) \cap \alpha = \emptyset$.

Parity. The parity acceptance condition is given by a total *priority function* $\Omega = p$ with $p : Q \to \mathbb{N}$ that assigns a *priority* to each state. A run $\rho \in Q^\omega$ is accepting in a *parity automaton* $\mathcal{A}$ if and only if $\min\{p(q) \mid q \in \mathsf{Inf}(\rho)\}$ is even. The total number of priorites of p, that is, $|\mathsf{Img}(p)|$, is called the *index* of $\mathcal{A}$.

Rabin. The Rabin acceptance condition is given by a finite set $\Omega = \{(E_1, F_1), \ldots, (E_k, F_k)\}$ of *final pairs* with $E_i, F_i \subseteq Q$ for all $i \in \{1, \ldots, k\}$. A run $\rho \in Q^\omega$ is accepting in a *Rabin automaton* $\mathcal{A}$ if and only if there exists a final pair $(E, F) \in \Omega$ such that $\mathsf{Inf}(\rho) \cap E = \emptyset$ and $\mathsf{Inf}(\rho) \cap F \neq \emptyset$. The total number of final pairs in Ω, that is, $|\Omega|$, is called the *index* of $\mathcal{A}$.

The infinite word language $\mathcal{L}(\mathcal{A})$ *recognized* by an infinite word automaton $\mathcal{A}$ is the set of words $u \in \Sigma^\omega$ for which there exists an accepting run in $\mathcal{A}$, that is, $\mathcal{L}(\mathcal{A}) = \{u \in \Sigma^\omega \mid \exists \rho \in \mathsf{Runs}_\mathcal{A}(u) \text{ s.t. } \rho \text{ is accepting in } \mathcal{A}\}$. Two infinite word automata $\mathcal{A}_1$ and $\mathcal{A}_2$ are *equivalent* if and only if they recognize the exact same language, that is, $\mathcal{L}(\mathcal{A}_1) = \mathcal{L}(\mathcal{A}_2)$. When $\mathcal{A}_1$ and $\mathcal{A}_2$ are equivalent, we write $\mathcal{A}_1 \equiv \mathcal{A}_2$.

Remark 2.5. Parity automata generalize Büchi and co-Büchi automata. Indeed,

for any Büchi automaton $\mathcal{A}_{\mathrm{b}} = (\Sigma, Q, q_0, \alpha, \delta)$, there exists an equivalent parity automaton $\mathcal{A}_{\mathrm{p}} = (\Sigma, Q, q_0, p, \delta)$ where $p : Q \to \{0, 1\}$ such that for all $q \in Q$, $p(q) = 0$ if and only if $q \in \alpha$. Similarly, for any co-Büchi automaton $\mathcal{A}_{\mathrm{cb}} = (\Sigma, Q, q_0, \alpha, \delta)$, there exists an equivalent parity automaton $\mathcal{A}_{\mathrm{p}} = (\Sigma, Q, q_0, p, \delta)$ where $p : Q \to \{1, 2\}$ such that for all $q \in Q, p(q) = 1$ if and only if $q \in \alpha$. ◁

Notations. We use acronyms in $\{\mathsf{D}, \mathsf{N}\} \times \{\mathsf{B}, \mathsf{CB}, \mathsf{P}, \mathsf{R}\}$ to denote the different types of infinite word automata, where D (resp. N) stands for deterministic (resp. nondeterministic), and B, CB, P and R respectively stand for Büchi, co-Büchi, parity and Rabin automata. For instance, NB means nondeterministic Büchi automaton. We denote the languages recognized by infinite word automata using subscripts that identify their acceptance condition. Given an infinite word automaton $\mathcal{A}$, we respectively write $\mathcal{L}_{\mathrm{b}}(\mathcal{A}), \mathcal{L}_{\mathrm{cb}}(\mathcal{A}), \mathcal{L}_{\mathrm{p}}(\mathcal{A})$ and $\mathcal{L}_{\mathrm{r}}(\mathcal{A})$ when $\mathcal{A}$ is a Büchi, co-Büchi, parity and Rabin automaton.

Example 2.6. Let $\mathcal{A} = (\Sigma_{\mathrm{bool}}, Q, q_0, \alpha, \delta)$ be a NB over the alphabet of booleans $\Sigma_{\mathrm{bool}} = \{0, 1\}$ such that $Q = \{q_0, q_1\}$, $\alpha = \{q_1\}$ and $\delta : Q \times \Sigma_{\mathrm{bool}} \to 2^Q$ is such that $\delta(q_0, 0) = \{q_0, q_1\}, \delta(q_0, 1) = \{q_0\}, \delta(q_1, 0) = \{q_1\}$ and $\delta(q_1, 1) = \{q_0\}$. The automaton $\mathcal{A}$ is depicted in Figure 2.1. Note that $\mathcal{A}$ is nondeterministic because $|\delta(q_0, 0)| > 1$. The language recognized by $\mathcal{A}$ is the set of words with infinitely many 0's, that is, $\mathcal{L}_{\mathrm{b}}(\mathcal{A}) = \{u \in \Sigma_{\mathrm{bool}}^{\omega} \mid 0 \in \mathsf{Inf}(u)\}$. By replacing the Büchi acceptance condition α of $\mathcal{A}$ by a parity condition $p : Q \to \mathbb{N}$ such that $p(q_0) = 1$ and $p(q_1) = 0$, we have that $\mathcal{L}_{\mathrm{p}}(\mathcal{A}) = \mathcal{L}_{\mathrm{b}}(\mathcal{A})$, as indicated in Remark 2.5. ◁

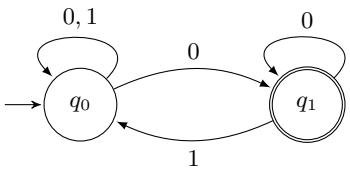

Figure 2.1: Nondeterministic Büchi automaton on Σ_{bool}.

From LTL to infinite word automata. It is well-known that NBs subsume LTL in the sense that for all LTL formula ϕ, there exists a (possibly exponentially

larger) NB accepting exactly the language $[\![\phi]\!]$, as indicated by the next theorem[1].

Theorem 2.7 ([WVS83])**.** *Let ϕ be an* LTL *formula over a set P of propositions. One can construct a* NB $\mathcal{A}_\phi = (\Sigma, Q, q_0, \alpha, \delta)$ *where* $\Sigma = 2^P$ *and* $|Q| \leq 2^{\mathcal{O}(|\phi|)}$ *such that* $\mathcal{L}_{\mathrm{b}}(\mathcal{A}_\phi) = [\![\phi]\!]$.

While NBs, NRs, DRs, NPs and DPs are equally expressive (e.g. [Tho97]), that is, they recognize the same infinite word languages, DBs are strictly less expressive [Lan69], that is, there exists infinite word languages that are recognized by NBs but cannot be recognized by DBs. Such a language is given in the next example.

Example 2.8. The infinite word language $\mathcal{L} = \{0,1\}^*0^\omega$ of words with finitely many 1's is recognized by the NB depicted of Figure 2.2. One can show that there is no DB that recognizes $\mathcal{L}$ (see [Rog01]). ◁

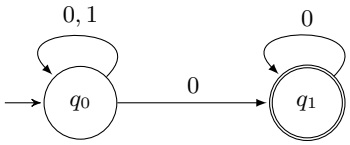

Figure 2.2: Nondeterministic Büchi automaton that recognizes $\{0,1\}^*0^\omega$.

The construction of deterministic automata equivalent to NBs thus require to consider another acceptance condition than Büchi. A classical determinizing procedure for NBs, called *Safra's determinization*, transforms NBs into (possibly exponentially larger) DRs, as indicated by the next theorem.

Theorem 2.9 ([Saf88])**.** *For any* NB *of size n, there exists an equivalent* DR *of size $2^{\mathcal{O}(n \log n)}$ and index n.*

Combining Theorems 2.7 and 2.9, we have that for any LTL formula ϕ, there exists a potentially doubly exponential DR that recognize $[\![\phi]\!]$. We have a similar result about DPs (e.g. [KV05]).

Theorem 2.10. *Let ϕ be an* LTL *formula over a set P of propositions such that $|\phi| = n$. One can construct a* DP $\mathcal{A}_\phi$ *of size $2^{2^{\mathcal{O}(n)}}$ and index $2^{\mathcal{O}(n)}$ such that* $\mathcal{L}_{\mathrm{p}}(\mathcal{A}_\phi) = [\![\phi]\!]$.

[1] We assume that the reader is familiar with notions of computational complexity [Pap03, Sip06].

Given an LTL formula ϕ and an infinite word automaton $\mathcal{A}$, we say that ϕ and $\mathcal{A}$ are *equivalent* if and only if $\mathcal{L}(\mathcal{A}) = [\![\phi]\!]$.

2.4 Games

In this section, we focus on games. Our games are played on directed graphs such that each vertex, called *state*, is controlled by exactly one player. In this thesis, we only consider 2-player games, where Player 1 is the protagonist and Player 2 has the role of the antagonist. We present several classical objectives on games and we recall some known results.

2.4.1 Definitions and objectives

Game graph. We start with the formal definitions of game graph and related notions.

Definition 2.11 (Game graph). A *game graph* is a tuple $\mathcal{G} = (S, s_0, E)$ such that:

- $S = S_1 \uplus S_2$ is a finite set of states, partitioned into S_1, the states of Player 1, and S_2, the states of Player 2,
- $s_0 \in S$ is the initial state, and
- $E \subseteq S \times S$ is the set of edges such that for all $s \in S$, there exists a state $s' \in S$ such that $(s, s') \in E$.

A game on $\mathcal{G}$ starts from the initial state s_0 and is played in rounds as follows. If the game is in a state belonging to S_1, then Player 1 chooses the successor state among the set of outgoing edges; otherwise Player 2 chooses the successor state. Such a game results in a *play*, that is, an *infinite path* $\rho = s_0 s_1 \cdots \in S^\omega$ with $(s_i, s_{i+1}) \in E$ for all $i \geq 0$. The prefix of length[2] n of the play ρ is the finite path $s_0 s_1 \dots s_n \in S^*$. It is denoted by $\rho(n)$. We denote by $\mathsf{Plays}(\mathcal{G})$ the set of all plays in $\mathcal{G}$ and by $\mathsf{Pref}(\mathcal{G})$ the set of all prefixes of plays in $\mathcal{G}$. A state $s \in S$ is *reachable* from the initial state if there is a sequence $s_0 s_1 \dots s_n \in S^*$ of states such that $s_n = s$ and for all i, $0 \leq i < n$, $(s_i, s_{i+1}) \in E$.

[2]The length is counted as the number of edges.

A *turn-based* game is a game graph $\mathcal{G}$ such that $E \subseteq (S_1 \times S_2) \cup (S_2 \times S_1)$, meaning that each game is played in rounds alternatively by Player 1 and Player 2.

Example 2.12. Let us consider the game graph $\mathcal{G} = (S = S_1 \uplus S_2, s_0, E)$ with five states such that $S_1 = \{s_1, s_3\}$, $S_2 = \{s_0, s_2, s_4\}$ and $E = \{(s_0, s_1), (s_1, s_0), (s_1, s_3), (s_2, s_1), (s_2, s_3), (s_3, s_4), (s_4, s_3)\}$. The game graph $\mathcal{G}$ is depicted in Figure 2.3. By convention, the states of Player 1 (resp. Player 2) are represented by circles (resp. squares). Note that $\mathcal{G}$ is not turn-based because of the edge (s_1, s_3). ◁

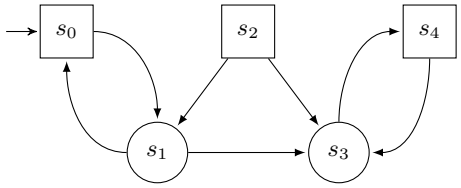

Figure 2.3: Game graph.

Objectives. Given a game graph $\mathcal{G} = (S, s_0, E)$, an *objective* for $\mathcal{G}$ is a set $\Omega \subseteq S^\omega$. Let $p : S \to \mathbb{N}$ be a *priority function* and $w : E \to \mathbb{Z}$ be a *weight function* where positive weights represent rewards[3]. We denote by W the largest weight (in absolute value) according to w. The *energy level* of a prefix $\gamma = s_0 s_1 \dots s_n$ of a play is $\mathsf{EL}_\mathcal{G}(\gamma) = \sum_{i=0}^{n-1} w(s_i, s_{i+1})$, and the *mean-payoff value* of a play $\rho = s_0 s_1 \dots$ is $\mathsf{MP}_\mathcal{G}(\rho) = \liminf_{n\to\infty} \frac{1}{n} \cdot \mathsf{EL}_\mathcal{G}(\rho(n))$. We consider the following objectives.

Safety. Given a set $\alpha \subseteq S$, the *safety objective* is defined as $\mathsf{Safety}_\mathcal{G}(\alpha) = \mathsf{Plays}(\mathcal{G}) \cap \alpha^\omega$.

Parity. The *parity objective* is defined as $\mathsf{Parity}_\mathcal{G}(p) = \{\rho \in \mathsf{Plays}(\mathcal{G}) \mid \min\{p(s) \mid s \in \mathsf{Inf}(\rho)\} \text{ is even}\}$.

Energy. Given an initial credit $c_0 \in \mathbb{N}$, the *energy objective* is defined as $\mathsf{PosEnergy}_\mathcal{G}(c_0) = \{\rho \in \mathsf{Plays}(\mathcal{G}) \mid \forall n \geq 0 : c_0 + \mathsf{EL}_\mathcal{G}(\rho(n)) \geq 0\}$.

Mean-payoff. Given a threshold $\nu \in \mathbb{Q}$, the *mean-payoff objective* is defined as $\mathsf{MeanPayoff}_\mathcal{G}(\nu) = \{\rho \in \mathsf{Plays}(\mathcal{G}) \mid \mathsf{MP}_\mathcal{G}(\rho) \geq \nu\}$.

[3]In some statements and proofs, we take the freedom to use *rational* weights (i.e. $w : E \to \mathbb{Q}$) as it is equivalent up to rescaling. However we always assume that weights are integers encoded in binary for complexity results.

Energy safety. Given $c_0 \in \mathbb{N}$ and $\alpha \subseteq S$, the *energy safety objective* $\mathsf{PosEnergy}_{\mathcal{G}}(c_0) \cap \mathsf{Safety}_{\mathcal{G}}(\alpha)$ combines the requirements of energy and safety objectives.

Energy parity. Given $c_0 \in \mathbb{N}$, the *energy parity objective* $\mathsf{PosEnergy}_{\mathcal{G}}(c_0) \cap \mathsf{Parity}_{\mathcal{G}}(p)$ combines the requirements of energy and parity objectives.

Mean-payoff parity. Given a threshold $\nu \in \mathbb{Q}$, the *mean-payoff parity objective* $\mathsf{MeanPayoff}_{\mathcal{G}}(\nu) \cap \mathsf{Parity}_{\mathcal{G}}(p)$ combines the requirements of mean-payoff and parity objectives.

When an objective Ω is imposed on a game $\mathcal{G}$, we say that $\mathcal{G}$ is an Ω *game*. For instance, if Ω is an energy safety objective, we say that $\mathcal{G}$ is an *energy safety game*.

Notations. We use acronyms in $\{\mathsf{S}, \mathsf{P}, \mathsf{E}, \mathsf{MP}, \mathsf{ES}, \mathsf{EP}, \mathsf{MPP}\} \times \{\mathsf{G}\}$, where S, P, E, MP, ES, EP and MPP respectively stand for safety, parity, energy, mean-payoff, energy safety, energy parity and mean-payoff parity, to denote the different types of games according to their associated objective. For instance, MPPG means mean-payoff parity game.

Strategies. Given a game graph $\mathcal{G} = (S, s_0, E)$, a *strategy* for Player 1 is a function $\lambda_1 : S^*S_1 \to S$ such that $(s, \lambda_1(\gamma \cdot s)) \in E$ for all $\gamma \in S^*$ and $s \in S_1$. A play $\rho = s_0 s_1 \dots$ starting from the initial state s_0 is *compatible* with λ_1 if for all $k \geq 0$ such that $s_k \in S_1$ we have $s_{k+1} = \lambda_1(\rho(k))$. The notions of strategy and play compatibility are defined symmetrically for Player 2. The set of strategies of Player 1 (resp. Player 2) is denoted by Π_1 (resp. Π_2). We denote by $\mathsf{Outcome}_{\mathcal{G}}(\lambda_1, \lambda_2)$ the unique play from s_0, called *outcome*, that is compatible with λ_1 and λ_2. The set of all outcomes $\mathsf{Outcome}_{\mathcal{G}}(\lambda_1, \lambda_2)$, with λ_2 any strategy of Player 2, is denoted by $\mathsf{Outcome}_{\mathcal{G}}(\lambda_1)$, that is, $\mathsf{Outcome}_{\mathcal{G}}(\lambda_1)$ is the set of plays starting from s_0 compatible with λ_1. A strategy λ_1 for Player 1 is *winning* for an objective Ω if $\mathsf{Outcome}_{\mathcal{G}}(\lambda_1) \subseteq \Omega$. We also say that λ_1 is winning in the Ω game $\mathcal{G}$.

Let $\cdot : S^* \times S^* \to S^*$ be the *concatenation* function, that is, for all $\gamma_1, \gamma_2 \in S^*$ with $\gamma_1 = s_1 \dots s_{k_1}$ and $\gamma_2 = s'_1 \dots s'_{k_2}$, $\gamma_1 \cdot \gamma_2 = s_1 \dots s_{k_1} s'_1 \dots s'_{k_2}$. A strategy λ_1 of Player 1 is *finite-memory* if there exists a right-congruence $\sim$ on $\mathsf{Pref}(\mathcal{G})$ of finite index such that $\lambda_1(\gamma \cdot s_1) = \lambda_1(\gamma' \cdot s_1)$ for all $\gamma \sim \gamma'$ and $s_1 \in S_1$. The

size of the memory is equal to the number of equivalence classes of $\sim$. We say that λ_1 is *memoryless* if $\sim$ has only one equivalence class. In other words, λ_1 is a mapping $S_1 \to S$ that only depends on the current state.

Example 2.13. Let us come back the game graph of Example 2.12. Let $w : E \to \mathbb{Z}$ be a weight function which assigns a weight to each edge such that $w((s_0, s_1)) = w((s_2, s_1)) = w((s_4, s_3)) = -3$, $w((s_1, s_0)) = 0$, $w((s_1, s_3)) = -1$, $w((s_2, s_3)) = 6$ and $w((s_3, s_4)) = 5$ (see Figure 2.4). Let $\lambda_1 : S_1 \to S$ be a memoryless strategy for Player 1 such that $\lambda_1(s_1) = s_3$ and $\lambda_1(s_3) = s_4$. The unique play $\mathsf{Outcome}_{\mathcal{G}}(\lambda_1) = s_0 s_1 (s_3 s_4)^\omega$ for all strategy $\lambda_2 \in \Pi_2$ of Player 2 has a mean-payoff value $\mathsf{MP}_{\mathcal{G}}(\mathsf{Outcome}_{\mathcal{G}}(\lambda_1)) = 1$. Note that the play $\mathsf{Outcome}_{\mathcal{G}}(\lambda_1)$ is unique since there is only one outgoing edge on states s_0 and s_4. $\lhd$

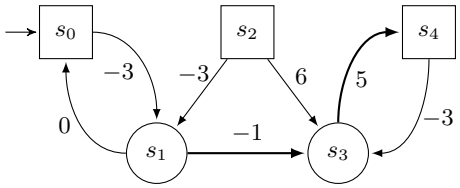

Figure 2.4: Game graph with weight function.

2.4.2 Some known results about games

We now recall some known results about SGs, EGs, EPGs and MPPGs.

Fixpoint algorithm for SGs. Let us consider a SG $\langle \mathcal{G}, \alpha \rangle$ with the safety objective $\mathsf{Safety}_{\mathcal{G}}(\alpha)$, or equivalently, with the objective to avoid $S \setminus \alpha$. The next classical fixpoint algorithm allows one to check whether Player 1 has a winning strategy (e.g. [GTW02]). Given a set $L \subseteq S$, we denote by $\mathsf{CPre}_1(L)$ the set $\{s \in S_1 \mid \exists (s, s') \in E, s' \in L\}$ and by $\mathsf{CPre}_2(L)$ the set $\{s \in S_2 \mid \forall (s, s') \in E, s' \in L\}$. We define the fixpoint $\mathsf{Win}_1(\alpha)$ of the sequence:

$$\begin{array}{rrcl} & W_0 & = & \alpha, \\ \forall k \geq 0, & W_{k+1} & = & W_k \bigcap \left\{ \mathsf{CPre}_1(W_k) \cup \mathsf{CPre}_2(W_k) \right\} \end{array}$$

It is well-known that Player 1 has a winning strategy λ_1 in the SG $\langle \mathcal{G}, \alpha \rangle$ if and only if $s_0 \in \mathsf{Win}_1(\alpha)$, and that the set $\mathsf{Win}_1(\alpha)$ can be computed in polynomial time. Moreover, the subgraph $\mathcal{G}'$ of $\mathcal{G}$ induced by $\mathsf{Win}_1(\alpha)$ is again a game graph

(i.e. every state has an outgoing edge), and if $s_0 \in \mathsf{Win}_1(\alpha)$, then λ_1 can be chosen as a memoryless strategy $S_1 \to S$ such that $(s, \lambda_1(s))$ is an edge in $\mathcal{G}'$ for all $s \in \mathsf{Win}_1(\alpha)$ (Player 1 forces to stay in $\mathcal{G}'$).

Initial credit problem for EGs and EPGs. We now consider EGs and EPGs. For an EG $\langle \mathcal{G}, w \rangle$ (resp. an EPG $\langle \mathcal{G}, w, p \rangle$), the *initial credit problem* asks whether there exists an initial credit $c_0 \in \mathbb{N}$ and a winning strategy for Player 1 for the objective $\mathsf{PosEnergy}_{\mathcal{G}}(c_0)$ (resp. $\mathsf{PosEnergy}_{\mathcal{G}}(c_0) \cap \mathsf{Parity}_{\mathcal{G}}(p)$). We have the next known results about EGs and EPGs.

Theorem 2.14 ([BFL+08, CdAHS03])**.** *The initial credit problem for a given* EG $\langle \mathcal{G} = (S, s_0, E), w \rangle$ *can be solved in* $\mathsf{NP} \cap \mathsf{coNP}$*. Moreover, if Player* 1 *has a winning strategy for the initial credit problem in* $\langle \mathcal{G}, w \rangle$*, then he can win with a memoryless strategy and with an initial credit of* $C = (|S| - 1) \cdot W$*.*

Theorem 2.15 ([CD10])**.** *The initial credit problem for a given* EPG $\langle \mathcal{G} = (S, s_0, E), w, p \rangle$ *can be solved in time* $\mathcal{O}(|E| \cdot d \cdot |S|^{d+3} \cdot W)$ *where* $d = |\mathsf{Img}(p)|$*. Moreover, if Player* 1 *has a winning strategy in* $\langle \mathcal{G}, w, p \rangle$*, then he has a finite-memory winning strategy with a memory size bounded by* $4 \cdot |S| \cdot d \cdot W$*.*

Example 2.16. In [CD10], the authors present a family of EPGs with n states where memory of size $2 \cdot (n-1) \cdot W + 1$ is necessary. This family of EPGs is depicted in Figure 2.5, where states are labeled by priorities. To satisfy the parity condition, the initial state must be visited infinitely often, and to maintain the energy objective, the rightmost state with the positive-weighted self-loop has to be visited. Paths between these two states have weight $-(n-1) \cdot W$. Thus, the self-loop has to be taken $M = 2 \cdot (n-1) \cdot W$ times, involving a memory of size $M + 1$. Note that initial credit $(n-1) \cdot W$ is needed. ◁

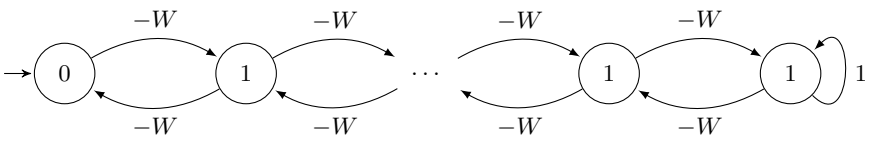

Figure 2.5: Family of EPGs where Player 1 needs memory of size $2 \cdot (n-1) \cdot W + 1$.

Optimal value in MPPGs. Let us now consider MPPGs $\langle \mathcal{G}, w, p \rangle$. With each play $\rho \in \mathsf{Plays}(\mathcal{G})$, we associate a *value* $\mathsf{Val}_{\mathcal{G}}(\rho)$ defined as follows:

$$\mathsf{Val}_{\mathcal{G}}(\rho) = \begin{cases} \mathsf{MP}_{\mathcal{G}}(\rho) & \text{if } \rho \in \mathsf{Parity}_{\mathcal{G}}(p) \\ -\infty & \text{otherwise.} \end{cases}$$

The *optimal value* for Player 1 is defined as:

$$\nu_{\mathcal{G}} = \sup_{\lambda_1 \in \Pi_1} \inf_{\lambda_2 \in \Pi_2} \mathsf{Val}_{\mathcal{G}}(\mathsf{Outcome}_{\mathcal{G}}(\lambda_1, \lambda_2)).^{4}$$

Given a real-valued $\epsilon \geq 0$, a strategy λ_1 for Player 1 is called *ϵ-optimal* if $\mathsf{Val}_{\mathcal{G}}(\mathsf{Outcome}_{\mathcal{G}}(\lambda_1, \lambda_2)) \geq \nu_{\mathcal{G}} - \epsilon$ against all strategies λ_2 of Player 2. It is *optimal* if it is ϵ-optimal with $\epsilon = 0$. If Player 1 cannot achieve the parity objective, then $\nu_{\mathcal{G}} = -\infty$, otherwise optimal strategies exist [CHJ05] and $\nu_{\mathcal{G}}$ is the largest threshold ν for which Player 1 can hope to achieve $\mathsf{MeanPayoff}_{\mathcal{G}}(\nu)$. We have the following known results about MPPGs.

Theorem 2.17 ([CHJ05, BMOU11, CRR12])**.** *The optimal value of a given* MPPG $\langle \mathcal{G} = (S, s_0, E), w, p \rangle$ *can be computed in time* $\mathcal{O}(|E| \cdot |S|^{d+2} \cdot W)$ *where* $d = |\mathsf{Img}(p)|$*. When* $\nu_{\mathcal{G}} \neq -\infty$*, optimal strategies for Player* 1 *may require infinite memory; however, for all* $\epsilon > 0$*, Player* 1 *has a finite-memory* ϵ*-optimal strategy.*

Example 2.18. In [BMOU11], an example of MPPG $\langle \mathcal{G}, w, p \rangle$ where optimal strategies for Player 1 require infinite memory is given. Such an MPPG is depicted in Figure 2.6, where states are labeled by priorities. We have $\nu_{\mathcal{G}} = 1$ since Player 1 can delay visiting the state with priority 0 while still ensuring to visit it infinitely often. However, no finite-memory strategy achieves this value. ◁

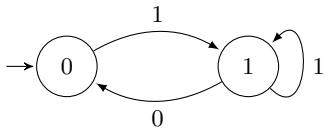

Figure 2.6: MPPG where optimal strategies for Player 1 require infinite memory.

[4] In the literature, this value is often called the *lower value* and denoted by $\underline{\nu_{\mathcal{G}}}$, in contrast with the *upper value* $\overline{\nu_{\mathcal{G}}} = \inf_{\lambda_2 \in \Pi_2} \sup_{\lambda_1 \in \Pi_1} \mathsf{Val}_{\mathcal{G}}(\mathsf{Outcome}_{\mathcal{G}}(\lambda_1, \lambda_2))$. From [Mar75], we have that $\underline{\nu_{\mathcal{G}}} = \overline{\nu_{\mathcal{G}}}$, that is, MPPGs are determined (see for instance [CHJ05]).

Multi-dimensional weights. Given a game graph $\mathcal{G} = (S, s_0, E)$, the objectives for $\mathcal{G}$ related to a weight function w can be naturally extended to multi-dimensional weights. Let $w : E \to \mathbb{Z}^m$ be a the weight function that assigns a vector of weights of dimension m to each edge of $\mathcal{G}$. Given a play $\rho \in \mathsf{Plays}(\mathcal{G})$, the notions of energy level $\mathsf{EL}_{\mathcal{G}}(\rho)$, mean-payoff value $\mathsf{MP}_{\mathcal{G}}(\rho)$ and value $\mathsf{Val}_{\mathcal{G}}(\rho)$ are defined similarly to the one-dimensional case. Given a game graph $\mathcal{G} = (S, s_0, E)$ and an objective $\Omega \subseteq S^\omega$ related to a weight function $w : E \to \mathbb{Z}^m$ with $m > 1$, we say that $\mathcal{G}$ is a *multi-Ω game*. For instance, if Ω is an energy objective, we say that $\mathcal{G}$ is a *multi-energy game*. We denote those games by acronyms in $\{\mathsf{M}\} \times \{\mathsf{E}, \mathsf{MP}, \mathsf{EP}, \mathsf{MPP}\} \times \{\mathsf{G}\}$. For instance, MMPPG means multi-mean-payoff parity game.

In [CRR12], the authors study the initial credit problem for MEPGs $\langle \mathcal{G} = (S, s_0, E), w, p, m\rangle$, m being the dimension of w. They extend the notion of self-covering tree, introduced in [BJK10], associated with the game. Intuitively, a self-covering tree is a finite unfolding of the game describing a winning strategy for Player 1.[5] They especially show that the depth of the self-covering tree is bounded by a constant $l = 2^{(h-1)\cdot|S|} \cdot (W \cdot |S| + 1)^{c \cdot m^2}$ where h is the highest number of outgoing edges on any state of S and c is a constant independent of the game. The next proposition states that MEPGs reduce to MEGs.

Proposition 2.19 ([CRR12])**.** *Let $\langle \mathcal{G} = (S, s_0, E), w, p, m\rangle$ be a* MEPG *with a priority function $p : S \to \{0, 1, \ldots, 2 \cdot d\}$ and a self-covering tree of depth bounded by l. One can construct a* MEG *$\langle \mathcal{G}, w', m'\rangle$ with $m' = m + d$ dimensions and a largest absolute weight W' bounded by l, such that a strategy is winning for Player 1 in $\langle \mathcal{G}, w, p, m\rangle$ if and only if it is winning in $\langle \mathcal{G}, w', m'\rangle$.*

The next results about MEGs and MEPGs come from [CDHR10] and [CRR12].

Theorem 2.20 ([CDHR10, CRR12])**.** *The following statements hold.*

- *The initial credit problem for a* MEG *is coNP-complete.*
- *If Player 1 has a winning strategy for the initial credit problem in a* MEPG*, then he can win with a finite-memory strategy of at most exponential size.*
- *Let $\langle \mathcal{G}, w, m\rangle$ be a* MEG *with a self-covering tree of depth bounded by l. If Player 1 has a winning strategy for the initial credit problem, then he can win with an initial credit $(C, \ldots, C) \in \mathbb{N}^m$ such that $C = 2 \cdot l \cdot W$.*

[5] See [CRR12] for the formal definition and results.

The third result of the previous theorem is extended in [CRR12] to MEPGs thanks to Proposition 2.19.

Example 2.21. The exponential upper bound on the size of winning strategies for Player 1 in MEPGs is tight. In [CRR12], the authors present a family of MEGs $\langle \mathcal{G}_K, w, m \rangle$ (without parity objective) which require exponential memory in the number of dimensions. Given $K \geq 1$, the EPG $\langle \mathcal{G}_K, w, m \rangle$ with $\mathcal{G}_K = (S = S_1 \uplus S_2, s_0, E)$ is an assembly of $m = 2 \cdot K$ gadgets such that the first K belong to Player 2 and the remaining K belong to Player 1. Precisely, we have $|S_1| = |S_2| = 3 \cdot K$, $|S| = |E| = 6 \cdot K = 3 \cdot m$ (linear in m) and $m = 2 \cdot K$. This family of MEGs is depicted in Figure 2.7. The weight function $w : E \to \{-1, 0, 1\}^m$ is defined such that

$$\forall 1 \leq i \leq K,\ w((\circ, s_i)) = w((\circ, t_i)) = (0, \dots, 0),$$

$$\forall 1 \leq j \leq m, w((s_i, s_{i,L}))(j) = \begin{cases} 1 \text{ if } j = 2 \cdot i - 1 \\ -1 \text{ if } j = 2 \cdot i \\ 0 \text{ otherwise, and} \end{cases}$$

$$w((s_i, s_{i,L})) = -w((s_i, s_{i,R})) = w((t_i, t_{i,L})) = -w((t_i, t_{i,R}))$$

where $\circ$ denotes any valid predecessor state. The idea is the following. If Player 1 does not remember the exact choices of Player 2 (which requires an exponential size memory), there will exist some sequence of choices of Player 2 such that Player 1 cannot counteract a decrease of energy. Thus, by playing this sequence long enough, Player 2 can force Player 1 to lose, whatever his initial credit is. $\lhd$

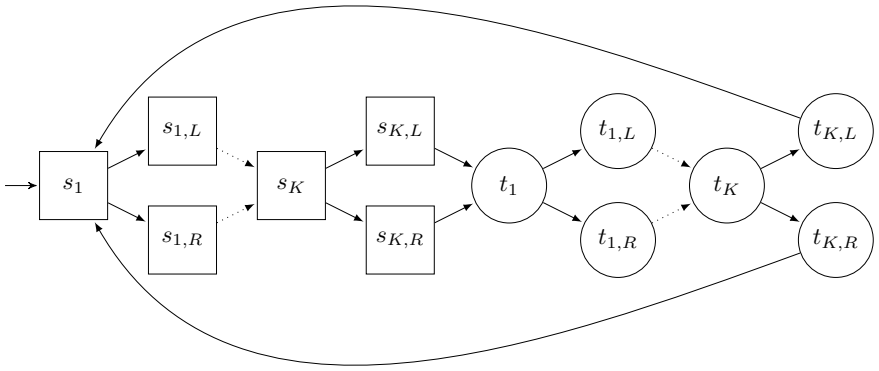

Figure 2.7: Family of MEGs requiring exponential memory [CRR12].

Relationship between (M)MPPGs and (M)EPGs. Let $\mathcal{G} = (S, s_0, E)$ be a game graph, $w : E \to \mathbb{Z}^m$ be a weight function and $\nu \in \mathbb{Q}^m$ be a threshold value. We denote by $w - \nu$ the weight function such that $(w - \nu)(e) = w(e) - \nu$, for all edge $e \in E$. The next proposition establishes a tight relationship between finite-memory strategies of (M)MPPGs and those of some related (M)EPGs. The result comes from [CD10] for the one-dimensional case, and from [CDHR10] for the extension to multiple dimensions.

Proposition 2.22 ([CD10, CDHR10])**.** *Let $\mathcal{G} = (S, s_0, E)$ be a game graph with a priority function $p : S \to \mathbb{N}$ and a weight function $w : E \to \mathbb{Z}^m$ of dimension $m \geq 1$. Let $\nu \in \mathbb{Q}^m$ be a threshold and λ_1 be a finite-memory strategy for Player 1. Then, λ_1 is winning in the MMPPG $\langle \mathcal{G}, w, p, m\rangle$ with threshold ν if and only if λ_1 is winning in the MEPG $\langle \mathcal{G}, w-\nu, p, m\rangle$ for some initial credit $c_0 \in \mathbb{N}^m$.*

We illustrate this relationship in the next example for the one-dimensional case for EGs and MPGs, that is, we do not consider the parity objective for sake of simplicity.

Example 2.23. Let us consider again the MPG $\langle \mathcal{G} = (S, s_0, E), w\rangle$ of Example 2.13 with threshold $\nu = 1$. Let $\langle \mathcal{G}, w'\rangle$ be an EG with $w' : E \to \mathbb{Z}$ such that $w' = w - \nu$. Those games are depicted in Figure 2.8. The memoryless strategy $\lambda_1 \in \Pi_1$ for Player 1 such that $\lambda_1(s_1) = s_3$ and $\lambda_1(s_3) = s_4$ is winning in the MPG $\langle \mathcal{G}, w\rangle$ since $\mathsf{MP}_{\mathcal{G}}(\mathsf{Outcome}_{\mathcal{G}}(\lambda_1)) = 1 \geq \nu$ (see Example 2.13). One can easily check that λ_1 is also a winning strategy in the EG $\langle \mathcal{G}, w'\rangle$ for initial credit $c_0 = 6$. ◁

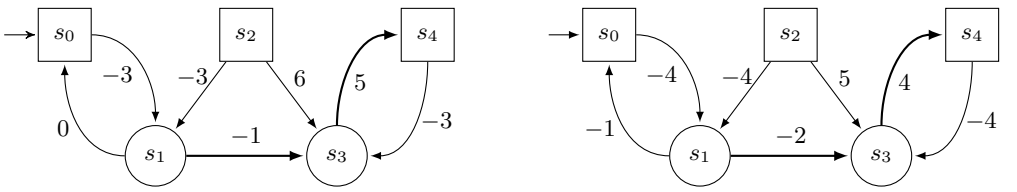

Figure 2.8: Relationship between mean-payoff (left) and energy games (right).

2.5 Stochastic models

In this section, we introduce two stochastic models that are Markov decision processes and Markov chains. A Markov decision process can be seen as a 2-player

game in which Player 2 is stochastic, that is, his choices are always made according to probability distributions. This is why Markov decision processes are sometimes referred to as $1\frac{1}{2}$-player games. A Markov chain is a Markov decision process for which a strategy (for Player 1) is fixed. We also state two classical problems on Markov decision processes and we briefly recall complexity results.

Stochastic models. A *probability distribution* over a finite set X is a total function $\pi : X \to [0,1]$ such that $\sum_{x \in X} \pi(x) = 1$. Its *support* is the set $\mathsf{Supp}(\pi) = \{x \in X \mid \pi(x) > 0\}$. We denote by $\mathsf{Dist}(X)$ the set of probability distributions over X. Given a probability distribution π over $X = \{x_1, \dots x_n\}$ such that $\pi(x_i) = p_i$, we sometimes write $\pi = \{p_1 : x_1, p_2 : x_2, \dots, p_n : x_n\}$ for convenience.

Definition 2.24 (Markov chain). A *discrete-time Markov chain (MC)* is a tuple $(S, \mathcal{P})$ where S is a finite set of states and $\mathcal{P} : S \to \mathsf{Dist}(S)$ is a stochastic transition function.

For all $s, s' \in S$, we often write $\mathcal{P}(s, s')$ for $\mathcal{P}(s)(s')$. A play in an MC results in an *infinite path* $\rho = s_0 s_1 s_2 \dots \in S^\omega$ such that $\mathcal{P}(s_i, s_{i+1}) > 0$ for all $i \geq 0$. Finite paths are defined similarly, and $\mathcal{P}$ is naturally extended to finite paths, that is, given a finite path $\rho = s_0 s_1 \dots s_n$, $\mathcal{P}(\rho) = \prod_{i=0}^{n-1} \mathcal{P}(s_i, s_{i+1})$. Given two states $s, s' \in S$, s' is *reachable* from s if there is a finite path $s_0 s_1 \dots s_n \in S^*$ such that $s_0 = s$ and $s_n = s'$. Given an MC $\mathcal{M} = (S, \mathcal{P})$, a subset $T \subseteq S$ is *strongly connected* if for all $t, t' \in T$, t' is reachable from t. A *strongly connected component* of $\mathcal{M}$ is a strongly connected set $T \subseteq S$ such that there is no $T' \subseteq S$ with $T \subseteq T'$ such that T' is strongly connected. Finally, a *bottom strongly connected component* of $\mathcal{M}$ is a strongly connected component T such that no state in $S \backslash T$ is reachable from states in T, that is, for all $t \in T$, $\sum_{t' \in T} \mathcal{P}(t, t') = 1$.

Definition 2.25 (Markov decision process). A *Markov decision process (MDP)* is a tuple $(S, \Sigma, \mathcal{P})$ where S is a finite set of states, Σ is a finite set of actions and $\mathcal{P} : S \times \Sigma \to \mathsf{Dist}(S)$ is a partial stochastic transition function.

We often write $\mathcal{P}(s, \sigma, s')$ for $\mathcal{P}(s, \sigma)(s')$. For each state $s \in S$, we denote by $\Sigma_s \subseteq \Sigma$ the set of enabled actions in s, where an action $\sigma \in \Sigma$ is *enabled* in s if $(s, \sigma) \in \mathsf{Dom}(\mathcal{P})$. For all state $s \in S$, we require $\Sigma_s \neq \emptyset$, and we thus say that the MDP is Σ-*non-blocking*. For all action $\sigma \in \Sigma$, we also introduce the

notation S_σ for the set of states in which σ is enabled. For $s \in S$ and $\sigma \in \Sigma_s$, we denote by $\mathsf{succ}(s,\sigma) = \mathsf{Supp}(\mathcal{P}(s,\sigma))$ the set of possible successors of s for the enabled action σ.

Strategies. The notion of strategy is naturally extended from games to MDPs. However, in this thesis, in the context of MDPs, we only consider memoryless strategies, that is, total functions $\lambda : S \to \Sigma$ mapping each state s to an enabled action $\sigma \in \Sigma_s$. We denote by Λ the set of all memoryless strategies. Given an MDP $\mathcal{M} = (S, \Sigma, \mathcal{P})$, a memoryless strategy λ induces an MC $(S, \mathcal{P}_\lambda)$ such that for all $s, s' \in S$, $\mathcal{P}_\lambda(s,s') = \mathcal{P}(s,\lambda(s),s')$. Given an MDP $(S,\Sigma,\mathcal{P})$ and two states $s, s' \in S$, s' is *reachable* from s if there exists a memoryless strategy λ such that s' is reachable from s in the induced MC $(S,\mathcal{P}_\lambda)$.

The notions of MDP, memoryless strategy and MC are illustrated on the next example.

Example 2.26. Let us consider the MDP $\mathcal{M} = (S,\Sigma,\mathcal{P})$ where $S = \{s_0, s_1, s_2\}$, $\Sigma = \{\sigma_1, \sigma_2\}$ and $\mathcal{P} : S \times \Sigma \to \mathsf{Dist}(S)$ is defined such that $\mathcal{P}(s_0,\sigma_1) = \{\frac{1}{2} : s_0, \frac{1}{2} : s_1, 0 : s_2\}$, $\mathcal{P}(s_0,\sigma_2) = \{\frac{5}{6} : s_0, 0 : s_1, \frac{1}{6} : s_2\}$, $\mathcal{P}(s_1,\sigma_1) = \{0 : s_0, 1 : s_1, 0 : s_2\}$ and $\mathcal{P}(s_2,\sigma_1) = \{\frac{1}{5} : s_0, 0 : s_1, \frac{4}{5} : s_2\}$. Let $\lambda : S \to \Sigma$ be a memoryless strategy such that $\lambda(s) = \sigma_1$, for all $s \in S$. It induces an MC $\mathcal{M}_\lambda = (S, \mathcal{P}_\lambda)$ where $\mathcal{P}_\lambda(s) = \mathcal{P}(s,\sigma_1)$, for all $s \in S$. The MDP $\mathcal{M}$ and the MC $\mathcal{M}_\lambda$ induced by λ are depicted in Figure 2.9. ◁

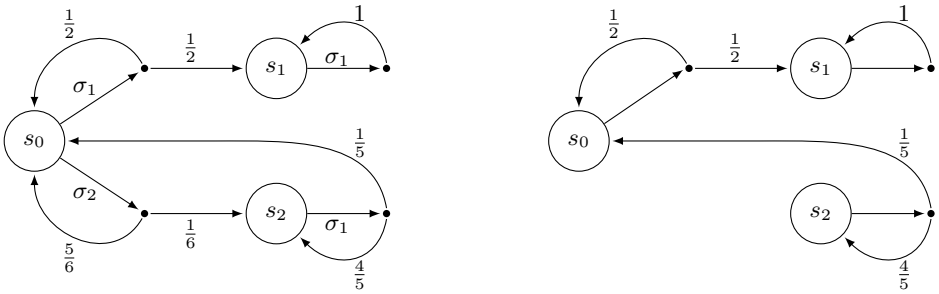

Figure 2.9: Markov decision process (left) with induced Markov chain (right).

Weights and value functions. Additionally to an MDP $(S,\Sigma,\mathcal{P})$, we consider a partial *weight function* $w : S \times \Sigma \to \mathbb{R}$ with $\mathsf{Dom}(w) = \mathsf{Dom}(\mathcal{P})$ that

associates a weight with a state s and an enabled action σ in s, such that positive weights represent rewards.[6] A memoryless strategy λ assigns a total weight function $w_\lambda : S \to \mathbb{R}$ to the induced MC $(S, \mathcal{P}_\lambda)$, such that $w_\lambda(s) = w(s, \lambda(s))$. Given a subset $G \subseteq S$ of *goal* states and a finite path ρ reaching a state of G, the *truncated sum up to* G of ρ is $\mathsf{TS}_G(\rho) = \sum_{i=0}^{n-1} w_\lambda(s_i)$ where $s_n \in G$ and for all i such that $0 \leq i < n$, $s_i \notin G$.

Given an MDP with a weight function w, and a memoryless strategy λ, we consider two classical value functions of λ that are the *expected mean-payoff* and the *expected truncated sum*, defined as follows. For all states $s \in S$, the *expected mean-payoff* of λ is $\mathbb{E}_\lambda^{\mathsf{MP}}(s) = \lim_{n\to\infty} \frac{1}{n} \sum_{i=0}^{n-1} \mathcal{P}_\lambda^i w_\lambda(s)$. Given a subset $G \subseteq S$, and assuming that λ reaches G from state s with probability 1, the *expected truncated sum up to* G of λ is $\mathbb{E}_\lambda^{\mathsf{TS}_G}(s) = \sum_\rho \mathcal{P}_\lambda(\rho)\mathsf{TS}_G(\rho)$ where the sum is over all finite paths $\rho = s_0 s_1 \dots s_n$ such that $s_0 = s$, $s_n \in G$, and $s_0, \dots, s_{n-1} \notin G$. Let λ^* be a memoryless strategy. Given a value function $\mathbb{E}_\lambda^{\cdot} \in \{\mathbb{E}_\lambda^{\mathsf{MP}}, \mathbb{E}_\lambda^{\mathsf{TS}_G}\}$, we say that λ^* is *optimal* if $\mathbb{E}_{\lambda^*}^{\cdot}(s) = \sup_{\lambda\in\Lambda} \mathbb{E}_\lambda^{\cdot}(s)$ for all $s \in S$, and $\mathbb{E}_{\lambda^*}^{\cdot}$ is called the *optimal* value function.[7] Note that we might have considered other classes of strategies (e.g. finite-memory strategies) but it is known that for these value functions, there always exists a memoryless strategy that maximizes the expected value of all states [BK08, Put94].

We now consider two classical problems on MDPs that are the expected mean-payoff and the stochastic shortest path problems.

Expected mean-payoff problem. Let $(S, \Sigma, \mathcal{P})$ be an MDP with a weight function $w : S \times \Sigma \to \mathbb{R}$. The *expected mean-payoff (EMP) problem* is to compute an optimal strategy for the expected mean-payoff value function. As explained above, such a memoryless optimal strategy always exists, and the problem is solvable in polynomial time via linear programming [Put94, FV97]. The EMP problem is illustrated in the following example.

Example 2.27. Let us consider the MDP $\mathcal{M} = (S, \Sigma, \mathcal{P})$ with $S = \{s_0, s_1\}$, $\Sigma = \{\sigma_0, \sigma_1\}$ and $\mathcal{P} : S \times \Sigma \to \mathsf{Dist}(S)$ defined such that $\mathcal{P}(s_0, \sigma_0) = \{1 : s_0, 0 : s_1\}$,

[6] In the context of MDPs, the weight function is not restricted to the set of integers.

[7] An alternative objective might be to minimize the value function, in which case λ^* is optimal if $\mathbb{E}_{\lambda^*}^{\cdot}(s) = \inf_{\lambda\in\Lambda} \mathbb{E}_\lambda^{\cdot}(s)$ for all $s \in S$.

$\mathcal{P}(s_0, \sigma_1) = \{\frac{1}{2} : s_0, \frac{1}{2} : s_1\}$ and $\mathcal{P}(s_1, \sigma_0) = \{0 : s_0, 1 : s_1\}$. The MDP $\mathcal{M}$ is depicted in Figure 2.10. We associate with $\mathcal{M}$ the weight function $w : S \times \Sigma \to \mathbb{R}$ such that $w(s_0, \sigma_0) = 2$, $w(s_0, \sigma_1) = 5$ and $w(s_1, \sigma_0) = 1$. The optimal strategy in $\mathcal{M}$ for the EMP problem is the strategy λ such that $\lambda(s_0) = \sigma_1$ and $\lambda(s_1) = \sigma_0$. The expected mean-payoff of λ is $\mathbb{E}^{\mathsf{MP}}_{\lambda}(s_0) = \mathbb{E}^{\mathsf{MP}}_{\lambda}(s_1) = 1$. ◁

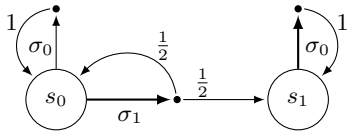

Figure 2.10: Markov decision process for the expected mean-payoff problem.

Stochastic shortest path problem. When the weight function w is restricted to *strictly negative* values in $\mathbb{R}_{<0}$, and a subset $G \subseteq S$ of goal states is given, the *stochastic shortest path (SSP) problem* is to compute an optimal strategy for the expected truncated sum value function, among the set of strategies that reach G with probability 1, provided such strategies exist. For all $s \in S$, we denote by Λ^{p}_s the set of *proper strategies* for s that are the strategies that lead from s to G with probability 1. Solving the SSP problem consists of two steps. The first step is to determine the set $S^{\mathsf{p}} = \{s \in S \mid \Lambda^{\mathsf{p}}_s \neq \emptyset\}$ of *proper* states, that is, the states having at least one proper strategy. The second step consists in computing an optimal strategy λ^* such that $\mathbb{E}^{\mathsf{TS}_G}_{\lambda^*}(s) = \sup_{\lambda \in \Lambda^{\mathsf{p}}_s} \mathbb{E}^{\mathsf{TS}_G}_{\lambda}(s)$ for all states of $s \in S^{\mathsf{p}}$.[8] It is known that memoryless optimal strategies exist for the SSP, and the problem can be solved in polynomial time through linear programming [BT91, BT96]. Note that the existence of at least one proper strategy for each state is often stated as an assumption on the MDP. It that case, an algorithm for the SSP problem is limited to the second step. The SSP problem is illustrated in the following example.

Example 2.28. Let us consider the MDP $\mathcal{M} = (S, \Sigma, \mathcal{P})$ with $S = \{s_0, s_1\}$, $\Sigma = \{\sigma_0, \sigma_1, \sigma_2\}$ and $\mathcal{P} : S \times \Sigma \to \mathsf{Dist}(S)$ defined such that $\mathcal{P}(s_0, \sigma_0) = \{1 : s_0, 0 : s_1\}$, $\mathcal{P}(s_0, \sigma_1) = \{\frac{9}{10} : s_0, \frac{1}{10} : s_1\}$, $\mathcal{P}(s_0, \sigma_2) = \{\frac{1}{5} : s_0, \frac{4}{5} : s_1\}$ and

[8] Usually the shortest path problem is presented as the problem of minimizing the expected truncated sum up to G. In that case, the weight function is restricted to strictly positive real values and $\mathbb{E}^{\mathsf{TS}_G}_{\lambda^*}(s) = \inf_{\lambda \in \Lambda^{\mathsf{p}}_s} \mathbb{E}^{\mathsf{TS}_G}_{\lambda}(s)$.

$\mathcal{P}(s_1, \sigma_0) = \{0 : s_0, 1 : s_1\}$. The MDP $\mathcal{M}$ is depicted in Figure 2.11. We associate with $\mathcal{M}$ the strictly negative weight function $w : S \times \Sigma \to \mathbb{R}_{<0}$ such that $w(s_0, \sigma_0) = w(s_0, \sigma_1) = w(s_1, \sigma_0) = -1$ and $w(s_0, \sigma_2) = -3$. Given the set $G = \{s_1\}$ of goal states, the optimal strategy in $\mathcal{M}$ for the stochastic shortest path problem is the strategy λ such that $\lambda(s_0) = \sigma_2$ and $\lambda(s_1) = \sigma_0$. Note that since $s_1 \in G$, $\lambda(s_1)$ can be defined arbitrarily. The expected truncated sum up to G of λ from s_0 is $\mathbb{E}_{\lambda}^{\mathsf{TS}_G}(s_0) = -3.75$. Finally, notice that the strategy λ' such that $\lambda'(s_0) = \sigma_0$ is not a proper strategy for s_0 since λ' leads from s_0 to G with probability 0. ◁

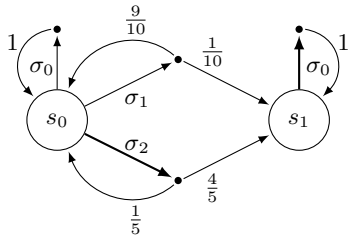

Figure 2.11: Markov decision process for the stochastic shortest path problem.

CHAPTER 3

Contributions and related work

In this chapter, we summarize our contributions and we give pointers to related work. The presentation of this chapter follows the subdivision of this thesis into parts. We first consider symbolic data structures, and in particular binary decision diagrams (BDDs) and antichains. We then address the subject of worst-case winning strategy synthesis, that is, the synthesis of strategies for systems playing against purely antagonistic environments. Finally, we summarize our results about good-for-expectation strategy synthesis, that is, the synthesis of strategies for systems playing against stochastic environments.

3.1 Data structures

In this section, we consider symbolic data structures. We present our contributions and we give pointers to several works in which such structures have been used.

3.1.1 Context

When considering complex systems obtained from high level descriptions or as the product of several components, the number of states to consider grows rapidly. This is the manifestation of the well-known *state explosion problem*. Moreover, when treating models with large state spaces, using explicit represen-

tations often exhausts the available memory and is thus infeasible. Symbolic representations like (multi-terminal) binary decision diagrams or antichains are then an alternative solution.

A *binary decision diagram (BDD)* [Bry86] is a data structure that allows to compactly represent boolean functions of n boolean variables, i.e. functions $\{0,1\}^n \to \{0,1\}$. A *multi-terminal binary decision diagram (MTBDD)* [CMZ+93, FMY97] is a generalization of a BDD used to represent functions of n boolean variables, i.e. functions $\{0,1\}^n \to V$, where V is a finite set. One of the most interesting applications of BDDs and MTBDDs is the representation of vectors and matrices, restricted to Boolean values in the case of BDDs. For instance, those data structures permit to compactly represent (stochastic) transition matrices.

Given a lower semilattice $\langle S, \preceq \rangle$, an *antichain* is a subset $\alpha \subseteq S$ whose elements are pairwise incomparable with respect to $\preceq$. Antichains permit to compactly represent subsets $L \subseteq S$ that are closed, that is, sets L such that for all $s \in L$ and $s' \in S$, if $s' \preceq s$, then $s' \in L$, by the set of their maximal elements $\lceil L \rceil = \{s \in L \mid \forall s' \in L \cdot s \preceq s' \Rightarrow s = s'\}$. Under the condition that the states of the model under consideration are partially ordered, antichains thus offer compact representations of closed subsets of states [DDHR06].

3.1.2 Contributions

In a joint work with Véronique Bruyère and Jean-François Raskin [BBR14b, BBR14a], we study antichain based algorithms for solving Markov decision processes. To handle several steps of those algorithms, we need to generalize existing antichain based data structures. Given a lower semilattice $\langle S, \preceq \rangle$, we introduce a new data structure called *pseudo-antichain* which permits to compactly represent *any* subset $L \subseteq S$, that is, L must not be necessarily closed.

Moreover, our work throughout this thesis has led us to implement antichain based [BBF+12, BBFR13] and pseudo-antichain based [BBR14b] algorithms in several contexts. Public and generic libraries for the manipulation of such data structures, even antichains, are missing. We fill this gap by proposing a generic library, called AaPAL, that implements functions in C for the representation and the manipulation of antichains and pseudo-antichains.

In Chapter 4, we recall the notion of antichains, which will be useful in all the remaining chapters, and we introduce the new structure of pseudo-antichains,

used in Chapters 7 and 8. We also briefly present the library AaPAL.

3.1.3 Related work

BDDs have been widely used in model checking (e.g. [BCM+92, CCGR00]) and synthesis (e.g. [WHT03, BCG+10, Ehl12]), and MTBDDs have been used in model checking of probabilistic systems (e.g. [dAKN+00, WBB+10, KNP11]). Packages for the manipulation of (MT)BDDs are available (e.g. CUDD [Som12], BuDDy [LN]). It has been shown that antichains might outperform BDDs. For instance, antichains have already been successfully applied for subset construction in automata theory (e.g. [DDHR06, DR07, BHH+08, ACH+10, DR10]).

3.2 Synthesis of worst-case winning strategies

In this section, we summarize our contributions and related work about the *synthesis of worst-case winning strategies*. In this framework, the environment of the system under consideration is purely antagonistic. Given a formal specification, the synthesized reactive system must satisfy the specification even in the most unlikely (and catastrophic) scenario of its execution, that is, even in the *worst-case*. We use two-player games to model the interaction between the system and its environment, and a strategy for the system is a *worst-case winning strategy* if it is winning against *all* strategies of its adversary.

3.2.1 Context

LTL realizability and synthesis are central problems when reasoning about specifications for reactive systems. In the LTL realizability problem, the uncontrollable input signals are generated by the environment whereas the controllable output signals are generated by the system which tries to satisfy the specification against any behavior of the environment. The *LTL realizability problem* can be stated as a two-player game as follows [PR89a]. Let ϕ be an LTL formula over a set P partitioned into O (output signals controlled by Player O, the system) and I (input signals controlled by Player I, the environment). In the first round of the play, Player O starts by giving a subset $o_1 \subseteq O$ and Player I responds by giving a subset $i_1 \subseteq I$. Then, the second round starts. Player O gives $o_2 \subseteq O$ and

Player I responds by $i_2 \subseteq I$, and so on for an infinite number of rounds. The outcome of this interaction is the infinite word $w = (i_1 \cup o_1)(i_2 \cup o_2) \dots (i_k \cup o_k) \dots$ Player O wins the play if w satisfies ϕ, otherwise Player I wins. The realizability problem asks to decide whether Player O has a winning strategy to satisfy ϕ, in which case we say that ϕ is *realizable*. Otherwise, we say that ϕ is *unrealizable*. The *LTL synthesis problem* asks to produce such a winning strategy when ϕ is realizable.

LTL realizability and synthesis problems have been first studied starting from the end of the eighties in the seminal work by Pnueli and Rosner [PR89a], and Abadi, Lamport and Wolper [ALW89]. The LTL realizability problem has been shown 2ExpTime-complete and finite-memory strategies suffice to win the realizability game [PR89b, Ros92]. The classical procedure is based on a reduction to the emptiness problem of Rabin automata [PR89b]. It can be summarized as follows. Given an LTL formula ϕ, first construct a nondeterministic Büchi automaton $\mathcal{A}_\phi$ equivalent to ϕ. Then, construct from $\mathcal{A}_\phi$ an equivalent deterministic Rabin automaton using Safra's construction [Saf88]. The obtained deterministic automaton is doubly exponential in the size of the LTL formula ϕ (see Theorems 2.7 and 2.9). Unfortunately, even if recent progresses have been made [Sch09, TFVT10], Safra's construction is intricate and notoriously difficult to implement efficiently [ATW06]. Due to their high worst-case complexities, the LTL realizability and synthesis problems have been considered for a long time only of theoretical interest. Only recently, several progresses on algorithms and efficient data structures showed that they can also be solved in practice. It triggered a renewed interest in these problems and a need for tools solving them.

Formalisms like LTL only allow the specifier to express *Boolean properties* (called *qualitative properties* in the sequel) in the sense that a reactive system either conforms to them, or violates them. Additionally to those qualitative formalisms, there is a clear need for another family of formalisms that are able to express *quantitative properties* of reactive systems. Abstractly, a quantitative property can be seen as a function that maps an execution of a reactive system to a numerical value. For example, in a client-server application, this numerical value could be the mean number of steps that separate the time at which a request has been emitted by a client and the time at which this request has been granted by the server along an execution. Quantitative properties are concerned with a

large variety of aspects like quality of service, bandwidth, energy consumption... But quantities are also useful to compare the merits of alternative solutions, e.g. we may prefer a solution in which the quality of service is high and the energy consumption is low. Currently, there is a large effort of the research community with the objective to lift the theory of formal verification and synthesis from the *qualitative world* to the richer *quantitative world* [Hen12].

3.2.2 Contributions

In a joint work with Véronique Bruyère, Emmanuel Filiot, Naiyong Jin and Jean-François Raskin [BBF+12], we present a new tool, called Acacia+, for solving the LTL realizability and synthesis problems. Acacia+ implements an antichain based algorithm proposed in [FJR11]. This algorithm is based on the procedure of [KV05], called *Safraless* since it avoids the costly determinization step involving Safra's construction. The algorithm proposed in [FJR11], is based on a reduction to safety games. This reduction offers very interesting properties in practice, like efficient symbolic incremental algorithms based on antichains, the synthesis of small winning strategies (when they exist), and a compositional approach for large conjunctions of LTL formulas.

While the procedure of [FJR11] has been first implemented in a Perl prototype called Acacia, we have reimplemented it from scratch in a new tool now made available to the research community. This tool has been developed in Python and C, with emphasis on modularity, code efficiency, and usability. We hope that this will motivate other researchers to take up the available code and extend it. Acacia+ can be downloaded or simply used via a web interface. While its performances are better than or similar to other existing tools, its main advantage is certainly the generation of *compact strategies* that are easily usable in practice. This aspect of Acacia+ is very useful in several application scenarios, like synthesis of control code from high-level LTL specifications, debugging of unrealizable specifications by inspecting compact counter strategies (Acacia+ can check both the realizability and the unrealizability of a given LTL specification), and generation of small deterministic automata from LTL formulas (when they exist). The tool Acacia+ and the implemented procedure are presented in Chapter 5.

In a joint work with Véronique Bruyère, Emmanuel Filiot and Jean-François

Raskin [BBFR13, BBFR12], we participate to the research effort of extending the theory of formal verification and synthesis from purely qualitative settings to quantitative ones. We provide theoretical complexity results, practical algorithmic solutions, and a tool for the automated synthesis of reactive systems from *quantitative specifications* expressed in LTL extended with (multi-dimensional) mean-payoff and (multi-dimensional) energy objectives. This work is presented in Chapter 6.

We define for the first time the realizability problems for $\mathsf{LTL}_{\mathsf{MP}}$ (LTL extended with mean-payoff objectives) and $\mathsf{LTL}_{\mathsf{E}}$ (LTL extended with energy objectives), and we show that, as for the LTL realizability problem, both the $\mathsf{LTL}_{\mathsf{MP}}$ and $\mathsf{LTL}_{\mathsf{E}}$ realizability problems are 2ExpTime-complete (Theorems 6.3 and 6.7). Our proofs follow a similar structure as the proof for LTL realizability. The 2ExpTime upper bound is obtained as follows. The formula is first turned into an equivalent nondeterministic Büchi automaton. This automaton is then transformed into a deterministic automaton using Safra's construction, which is doubly exponential in the size of the LTL formula. For the $\mathsf{LTL}_{\mathsf{MP}}$ realizability problem, the latter automaton can be seen as a two-player mean-payoff parity game in which Player 1 wins if and only if the $\mathsf{LTL}_{\mathsf{MP}}$ specification is realizable (Proposition 6.4). For the $\mathsf{LTL}_{\mathsf{E}}$ realizability problem, the same construction is used with a reduction to a two-player energy parity game. By a careful analysis of the complexity of all the steps involved in those two constructions, we build, on the basis of results in [CD10] and [CHJ05], solutions that provide the announced 2ExpTime upper bound.

As recalled in Theorem 2.17, it is known that optimal strategies in mean-payoff parity games may require infinite memory, but there exist ϵ-optimal finite-memory strategies. In contrast, for energy parity games, it is known that finite-memory optimal strategies exist (see Theorem 2.15). We show that those results transfer to $\mathsf{LTL}_{\mathsf{MP}}$ (resp. $\mathsf{LTL}_{\mathsf{E}}$) realizability problems thanks to the reduction of these problems to mean-payoff (resp. energy) parity games. Furthermore, we show that under finite-memory strategies, $\mathsf{LTL}_{\mathsf{MP}}$ realizability is in fact equivalent to $\mathsf{LTL}_{\mathsf{E}}$ realizability: a specification is MP-realizable (i.e. realizable with the mean-payoff objective) under finite-memory strategies if and only if it is E-realizable (i.e. realizable with the energy objective), by simply shifting the weights by the threshold value (Theorem 6.9). Because finite-memory strategies

are more interesting in practice, we thus concentrate on the LTL_E realizability problem.

Following [KV05], we develop a Safraless procedure for the LTL_E realizability problem, that is based on a reduction to a safety game, with the nice property to transform a quantitative objective into a simple qualitative objective (Theorem 6.18). Our results can be extended to multi-dimensional mean-payoff and multi-dimensional energy objectives (Theorems 6.26 and 6.27).

Finally, we discuss some implementation issues. The proposed Safraless construction has the advantage that the state space of the safety game can be partially ordered and solved by a backward fixpoint algorithm. Since the latter manipulates sets of states closed for this order, it can be made efficient and symbolic by working only on the antichain of their maximal elements. The algorithm has been implemented in our tool Acacia+, and promising experimental results are reported.

3.2.3 Related work

In [KV05], the authors have proposed for the first time a Safraless procedure which avoids the determinization step by reducing the LTL realizability problem to Büchi games. It has been implemented in the tool Lily [JB06]. Another Safraless approach has been given in [SF07] for the distributed LTL synthesis problem. It is based on a novel emptiness-preserving translation from LTL to safety tree automata. In [Ehl11, Ehl12], a procedure for the LTL synthesis problem is proposed and implemented in the tool Unbeast, based on the approach of [SF07] and symbolic game solving with BDDs. The construction proposed in [FJR11] implemented in our tool Acacia+ is also similar to [SF07].

Given an LTL formula ϕ, the idea of checking the realizability of ϕ by Player O in parallel with the realizability of $\neg\phi$ for Player I (that is, the unrealizability of ϕ) has been proposed in [BK09], and reused in [FJR11]. In [SS09, FJR11], the authors consider the synthesis problem from LTL formulas given as conjunctions of sub-formulas, that is, formulas of the form $\wedge_i \phi_i$. They propose efficient algorithms to construct compositionally the games related to such specifications.

Note that the complexity of the realizability problem is much lower for LTL formulas of specific forms like $\wedge_i \Box\Diamond p_i$ (polynomial time) or $\wedge_i(\Box\Diamond p_i \Rightarrow \Box\Diamond q_i)$ (coNP-complete) [Rab72, EJ88]. Moreover, the problem of synthesis

from fragments of LTL has been studied [AL04], solutions have been implemented [WHT03, HRS05, PPS06, BGJ+07, BDLL13], and tools were developed (e.g. ANZU [JGWB07], and its extension RATSY [BCG+10], for the Generalized Reactivity(1) fragment of LTL, that is, formulas of the form $(\wedge_i \Box\Diamond p_i) \Rightarrow (\wedge_j \Box\Diamond q_j)$). Finally, Safra's construction, together with some heuristics, has been implemented in the tool ltl2dstar [KB05]. Moreover, several constructions of deterministic Rabin automata for fragments of LTL have been proposed and implemented [KE12, BBKS13, KLG13].

Games with quantitative objectives have been extensively studied recently. Indeed, mean-payoff games [ZP96] and energy games [BFL+08, BCD+11], and their extensions with parity conditions [CHJ05, CD10, BMOU11] or to multi-dimensions [CDHR10, CRR12] have received a large attention from the research community. From a practical point of view, the value iteration algorithm of [ZP96] for solving mean-payoff games has been implemented in the tool QUASY [CHJS11]. The use of such game formalisms has been advocated in [BCHJ09] for specifying quantitative properties of reactive systems. All these related works make the assumption that the game graph is given explicitly (and not implicitly using, for instance, an LTL formula). In [BCHK11], Boker et al. introduce extensions of temporal logics with operators to express constraints on values accumulated along the paths of a weighted Kripke structure. One of their extensions is similar to $\mathsf{LTL}_{\mathsf{MP}}$. However, the authors of [BCHK11] only study the complexity of model checking problems whereas we focus on realizability and synthesis problems.

3.3 Synthesis of good-for-expectation strategies

In this section, we consider our contributions and related work about *good-for-expectation* strategy synthesis. In this framework, we are interested in the synthesis of reactive systems which ensure a *good expected performance*, for a given quantitative measure. The environment is not antagonistic like in the worst-case framework, but it is rather *stochastic*. We use Markov decision processes and Markov chains to model the interaction of the system and the stochastic environment. The system is thus playing against a stochastic model of its environment, and a strategy for the system is a *good-for-expectation strategy* if it is optimal

for the given quantitative measure.

3.3.1 Context

Markov decision processes (MDPs) [Put94] are rich models that exhibit both nondeterministic choices and stochastic transitions. There are two main families of algorithms for MDPs. First, *value iteration* algorithms [Bel57] assign values to states of the MDPs and refine locally those values by successive approximations. If a fixpoint is reached, the value at a state s represents a probability or an expectation that can be achieved by an optimal strategy that resolves the choices present in the MDP starting from s. This value can be, for example, the maximal probability to reach a set of goal states. Second, *strategy iteration* algorithms [How60] start from an arbitrary strategy and iteratively improve the current strategy by local changes up to the convergence to an optimal strategy. Both methods have their advantages and disadvantages. Value iteration algorithms usually lead to easy and efficient implementations, but in general the fixpoint is not guaranteed to be reached in a finite number of iterations, and so only approximations are computed. On the other hand, strategy iteration algorithms have better theoretical properties as convergence towards an optimal strategy in a finite number of steps is usually ensured, but they often require to solve systems of linear equations, and so they are more difficult to implement efficiently.

When considering large MDPs, explicit methods often exhaust available memory and are thus impractical. Symbolic representations with MTBDDs are useful but they are usually limited to systems with around 10^6 or 10^7 states only. Also, as mentioned above, some algorithms for MDPs rely on solving linear systems, and there is no easy use of BDD like structures for implementing such algorithms.

Recently, Wimmer et al. [WBB+10] have proposed a method that *mixes* symbolic and explicit representations to efficiently implement the Howard and Veinott strategy iteration algorithm [How60, Vei66] to compute optimal strategies for mean-payoff objectives in MDPs. Their solution is as follows. First, the MDP is represented and handled symbolically using MTBDDs. Second, a strategy is fixed symbolically and the MDP is transformed into a Markov chain (MC). To analyze this MC, a linear system needs to be constructed from its state space. As this state space is potentially huge, the MC is first reduced by

lumping [KS60, Buc94] (bisimulation reduction), and then a (hopefully) compact linear system can be constructed and solved. Solutions to this linear system allow to show that the current strategy is optimal, or to obtain sufficient information to improve it. A new iteration is then started. The main difference between this method and the other methods proposed in the literature is its *hybrid nature*: it is symbolic for handling the MDP and for computing the lumping, and it is explicit for the analysis of the reduced MC. This is why the authors of [WBB+10] have coined their approach *symblicit*.

3.3.2 Contributions

In a joint work with Véronique Bruyère and Jean-François Raskin [BBR14b, BBR14a], we build on the symblicit approach described above. Our contributions are threefold.

First, we show that the symblicit approach and the strategy iteration algorithm can also be efficiently applied to the *stochastic shortest path* problem. We start from an algorithm proposed by Bertsekas and Tsitsiklis [BT96] with a preliminary step by de Alfaro [dA99], and we show how to cast it in the symblicit approach. Second, we show that alternative data structures can be more efficient than BDDs or MTBDDs for implementing a symblicit approach, both for mean-payoff and stochastic shortest path objectives. In particular, we consider a natural class of MDPs with *monotonic properties* on which the structure of antichains, and its extension to pseudo-antichains, is more efficient. Our antichain based symblicit algorithms are presented in Chapter 7.

Third, we have implemented our algorithms and we show that they are more efficient than existing solutions on natural examples of monotonic MDPs. We show that monotonic MDPs naturally arise in probabilistic planning [BL00] and when optimizing systems synthesized from LTL specifications with mean-payoff objectives [BBFR13]. In the latter context, we have plugged the symblicit algorithm into Acacia+, building a bridge between worst-case and good-for-expectation strategy synthesis. To the best of our knowledge, this is the first time that a tool performs automated synthesis of winning strategies in the worst-case which also ensure good expected performance. The two applications of our antichain based symblicit algorithms are tackled in Chapter 8, where experimental results are also reported.

3.3.3 Related work

Model checking algorithms for MDPs exist for logical properties expressible in the logic PCTL [HJ94], a stochastic extension of CTL [CE81], and are implemented in tools like PRISM [KNP11], MODEST [Har12], MRMC [KZH+11], LiQuor [CB06]... There also exist algorithms for *quantitative properties* such as the mean-payoff or the stochastic shortest path, that have been implemented in tools like QUASY [CHJS11] and PRISM [vEJ12].

In [BFRR14], the framework of *beyond worst-case synthesis* is introduced. Complexities, memory requirements and algorithms are established for the expected mean-payoff and the stochastic shortest path problems within this framework. While their problem seems similar, it differs from ours in the following sense. They compute the optimal strategy among the set of *all* worst-case winning strategies, while we limit ourselves to a *subset* of worst-case winning strategies computed by our procedure for $\mathsf{LTL}_{\mathsf{MP}}$ synthesis. Note that no implementation of the algorithms of [BFRR14] is proposed.

Part I

Data structures

CHAPTER 4

Compact representations of partially ordered sets

This chapter presents data structures for the compact representation and efficient manipulation of partially ordered sets. We first recall the notion of antichain, a structure that provides a canonical representation of closed sets. We then present the notion of pseudo-antichain, a structure extended from antichains, that has been first introduced in [BBR14b]. Pseudo-antichains are compact representations of partially ordered sets that are not necessarily closed. Finally, we briefly present the library AaPAL for the manipulation of antichains and pseudo-antichains.

4.1 Antichains

In this section, we introduce the data structure of antichain and related notions. We also state classical properties on antichains.

Closed sets. Let $\langle S, \preceq \rangle$ be a lower semilattice. A set $L \subseteq S$ is *closed* for $\preceq$ if for all $s \in L$ and all $s' \in S$ such that $s' \preceq s$, we have $s' \in L$. When it is clear from context, we usually omit to specify for which partial order a set is closed. If $L_1, L_2 \subseteq S$ are two closed sets, then $L_1 \cap L_2$ and $L_1 \cup L_2$ are

closed, but $L_1 \setminus L_2$ is not necessarily closed. The *closure* $\downarrow L$ of a set L is the set $\downarrow L = \{s' \in S \mid \exists s \in L \cdot s' \preceq s\}$. Note that $\downarrow L = L$ for all closed sets L.

Antichains. A set $\alpha \subseteq S$ is an *antichain* if all its elements are pairwise incomparable with respect to $\preceq$. For $L \subseteq S$, we denote by $\lceil L \rceil$ the set of its maximal elements, that is $\lceil L \rceil = \{s \in L \mid \forall s' \in L \cdot s \preceq s' \Rightarrow s = s'\}$. This set $\lceil L \rceil$ is an antichain. If L is closed, then $\downarrow \lceil L \rceil = L$, and $\lceil L \rceil$ is called the *canonical representation* of L. The interest of antichains is that they are *compact* representations of closed sets.

Example 4.1. Let $\mathbb{N}^2_{\leq 3}$ be the set of pairs of natural numbers in $[0, 3]$ and $\langle \mathbb{N}^2_{\leq 3}, \preceq \rangle$ be a complete lattice such that the partial order $\preceq$, the least upper bound $\sqcup$ and the greatest lower bound $\sqcap$ are defined as in Example 2.2 (page 11) for the complete lattice $\langle \mathbb{N}^2, \preceq \rangle$. The set $\alpha = \{(2,1),(0,2)\} \subseteq \mathbb{N}^2_{\leq 3}$ is an antichain, and its closure is the set $\downarrow \alpha = \{(0,0),\ (1,0),\ (2,0),\ (0,1),\ (1,1),\ (2,1),\ (0,2)\}$. A graphical representation of $\downarrow \alpha$ is given in Figure 4.1. ◁

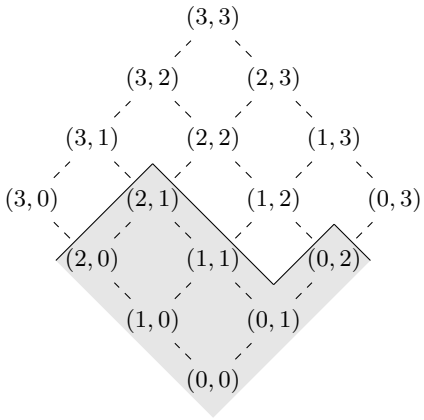

Figure 4.1: Graphical representation of the closure of an antichain over $\langle \mathbb{N}^2_{\leq 3}, \preceq \rangle$.

We have the next classical properties on antichains.

Proposition 4.2 ([FJR11])**.** *Let $\alpha_1, \alpha_2 \subseteq S$ be two antichains and $s \in S$. Then:*

- $s \in \downarrow \alpha_1$ *iff* $\exists a \in \alpha_1$ *such that* $s \preceq a$,
- $\downarrow \alpha_1 \subseteq \downarrow \alpha_2$ *iff* $\forall a_1 \in \alpha_1$, $\exists a_2 \in \alpha_2$ *such that* $a_1 \preceq a_2$,
- $\downarrow \alpha_1 \cup \downarrow \alpha_2 = \downarrow \lceil \alpha_1 \cup \alpha_2 \rceil$, *and*

- $\downarrow\alpha_1 \cap \downarrow\alpha_2 = \downarrow\lceil \alpha_1 \sqcap \alpha_2 \rceil$, *where* $\alpha_1 \sqcap \alpha_2 \stackrel{\text{def}}{=} \{a_1 \sqcap a_2 \mid a_1 \in \alpha_1, a_2 \in \alpha_2\}$.

For convenience, when α_1 and α_2 are antichains, we use notation $\alpha_1 \dot{\cup} \alpha_2$ (resp. $\alpha_1 \dot{\cap} \alpha_2$) for the antichain $\lceil \downarrow\alpha_1 \cup \downarrow\alpha_2 \rceil$ (resp. $\lceil \downarrow\alpha_1 \cap \downarrow\alpha_2 \rceil$).

The following example illustrates the third and fourth statements of Proposition 4.2.

Example 4.3. Let $\langle S, \preceq \rangle$ be a lower semilattice and let $\alpha = \{a\}$ and $\beta = \{b_1, b_2\}$, with $a, b_1, b_2 \in S$, be two antichains as depicted in Figure 4.2. We have $\downarrow\alpha \cup \downarrow\beta = \downarrow\{a, b_1, b_2\}$ and $\downarrow\alpha \cap \downarrow\beta = \downarrow\{a \sqcap b_1, a \sqcap b_2\}$. ◁

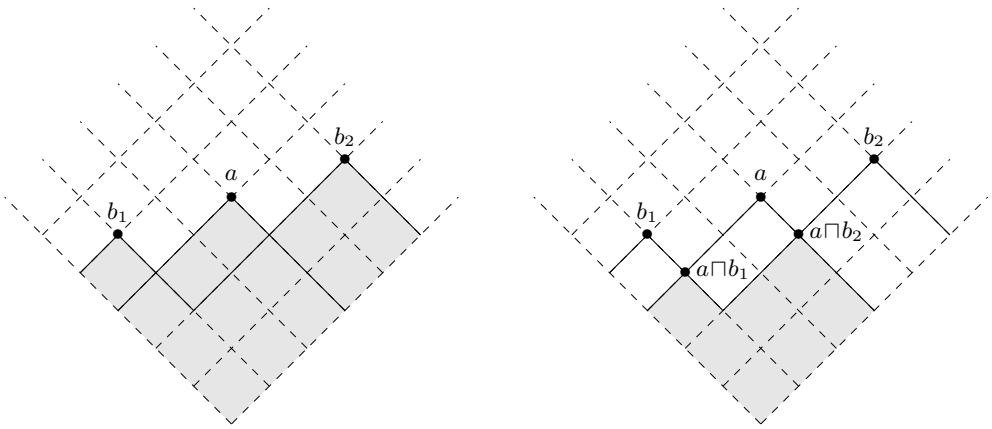

Figure 4.2: Union (left) and intersection (right) of antichains closure.

4.2 Pseudo-antichains

Let $L_1, L_2 \subseteq S$ be two closed sets. Unlike the union or intersection, the difference $L_1 \setminus L_2$ is not necessarily a closed set. There is thus a need for a new structure that "represents" $L_1 \setminus L_2$ in a compact way, as antichains compactly represent closed sets. To this end, in this section, we begin by introducing the notion of pseudo-element, and we then introduce the notion of pseudo-antichain. We also describe some properties that can be used by algorithms using pseudo-antichains. This section is based on [BBR14b, BBR14a].

4.2.1 Pseudo-elements and pseudo-closures

Definitions. Let $\langle S, \preceq \rangle$ be a lower semilattice. A *pseudo-element* is a pair (x, α) where $x \in S$ and $\alpha \subseteq S$ is an antichain such that $x \notin \downarrow\alpha$. The *pseudo-closure* of a pseudo-element (x, α), denoted by $\updownarrow(x, \alpha)$, is the set $\updownarrow(x, \alpha) = \{s \in S \mid s \preceq x \text{ and } s \notin \downarrow\alpha\} = \downarrow\{x\} \backslash \downarrow\alpha$. Notice that $\updownarrow(x, \alpha)$ is non empty since $x \notin \downarrow\alpha$ by definition of a pseudo-element, i.e. $x \in \updownarrow(x, \alpha)$.

The following example illustrates the notion of pseudo-closure of pseudo-elements.

Example 4.4. Let $\langle \mathbb{N}^2_{\leq 3}, \preceq \rangle$ be the complete lattice of Example 4.1 (page 48). Let $x = (3, 2) \in \mathbb{N}^2_{\leq 3}$ and $\alpha = \{(2, 1), (0, 2)\} \subseteq \mathbb{N}^2_{\leq 3}$. Then, the pseudo-closure of the pseudo-element (x, α) is the set $\updownarrow(x, \alpha) = \{(3, 2), (3, 1), (3, 0), (1, 2), (2, 2)\} = \downarrow\{x\} \backslash \downarrow\alpha$. A graphical representation of $\updownarrow(x, \alpha)$ is given in Figure 4.3. ◁

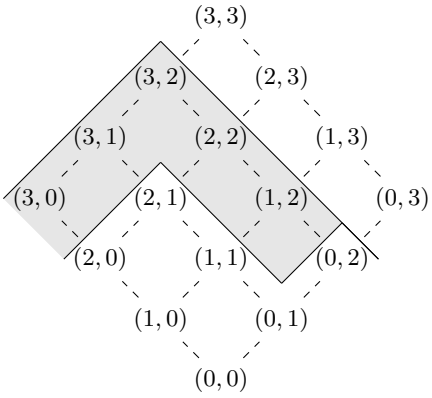

Figure 4.3: Graphical representation of the pseudo-closure of a pseudo-element over $\langle \mathbb{N}^2_{\leq 3}, \preceq \rangle$.

Canonical representations. There may exist two pseudo-elements (x, α) and (y, β) such that $\updownarrow(x, \alpha) = \updownarrow(y, \beta)$. We say that the pseudo-element (x, α) is in *canonical form* if for all $a \in \alpha$, $a \preceq x$. The next proposition and its corollary show that the canonical form is unique. Notice that for all pseudo-elements (x, α), there exists a pseudo-element in canonical form (y, β) such that $\updownarrow(x, \alpha) = \updownarrow(y, \beta)$: it is equal to $(x, \{x\} \mathbin{\dot\cap} \alpha)$. We say that such a pair (y, β) is the *canonical representation* of $\updownarrow(x, \alpha)$.

Proposition 4.5. *Let (x, α) and (y, β) be two pseudo-elements. Then $\updownarrow(x, \alpha) \subseteq \updownarrow(y, \beta)$ if and only if $x \preceq y$ and for all $b \in \beta, b \sqcap x \in \downarrow\alpha$.*

Proof. We prove the two implications:

- $\Rightarrow$: Suppose that $\updownarrow(x, \alpha) \subseteq \updownarrow(y, \beta)$ and let us prove that $x \preceq y$ and $\forall b \in \beta$, $b \sqcap x \in \downarrow\alpha$. As $x \in \updownarrow(x, \alpha) \subseteq \updownarrow(y, \beta)$, then $x \preceq y$. Consider $s = b \sqcap x$ for some $b \in \beta$. We have $s \notin \updownarrow(y, \beta)$ because $s \preceq b$ and thus $s \notin \updownarrow(x, \alpha)$. As $s \preceq x$, it follows that $s \in \downarrow\alpha$.
- $\Leftarrow$: Suppose that $x \preceq y$ and $\forall b \in \beta, b \sqcap x \in \downarrow\alpha$. Let us prove that $\forall s \in \updownarrow(x, \alpha)$, we have $s \in \updownarrow(y, \beta)$. As $s \preceq x$, and $x \preceq y$ by hypothesis, we have $s \preceq y$. Suppose that $s \in \downarrow\beta$, that is $s \preceq b$, for some $b \in \beta$. As $s \preceq x$, we have $s \preceq b \sqcap x$ and thus $s \in \downarrow\alpha$ by hypothesis. This is impossible since $s \in \updownarrow(x, \alpha)$. Therefore, $s \notin \downarrow\beta$, and thus $s \in \updownarrow(y, \beta)$.

□

The following example illustrates Proposition 4.5.

Example 4.6. Let $\langle S, \preceq \rangle$ be a lower semilattice and let $(x, \{a\})$ and $(y, \{b_1, b_2\})$, with $x, y, a, b_1, b_2 \in S$, be two pseudo-elements as depicted in Figure 4.4. The pseudo-closure of $(x, \{a\})$ is depicted in dark gray, whereas the pseudo-closure of $(y, \{b_1, b_2\})$ is depicted in (light and dark) gray. We have $x \preceq y$, $b_1 \sqcap x = b_1 \in \downarrow\{a\}$ and $b_2 \sqcap x = b_2 \in \downarrow\{a\}$. We thus have $\updownarrow(x, \{a\}) \subseteq \updownarrow(y, \{b_1, b_2\})$. ◁

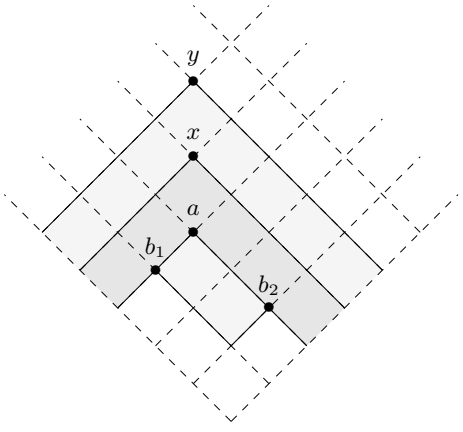

Figure 4.4: Inclusion of pseudo-closure of pseudo-elements.

The next corollary is a direct consequence of Proposition 4.5.

Corollary 4.7. *Let (x, α) and (y, β) be two pseudo-elements in canonical form. Then $\updownarrow(x, \alpha) = \updownarrow(y, \beta)$ if and only if $x = y$ and $\alpha = \beta$.*

Proof. We only prove $\updownarrow(x, \alpha) = \updownarrow(y, \beta) \Rightarrow x = y$ and $\alpha = \beta$, the other implication being trivial.

Since $\updownarrow(x, \alpha) = \updownarrow(y, \beta)$, we have $\updownarrow(x, \alpha) \subseteq \updownarrow(y, \beta)$ and $\updownarrow(y, \beta) \subseteq \updownarrow(x, \alpha)$. By Proposition 4.5, from $\updownarrow(x, \alpha) \subseteq \updownarrow(y, \beta)$, we know that $x \preceq y$ and $\forall b \in \beta$, $b \sqcap x \in \downarrow\alpha$, and from $\updownarrow(y, \beta) \subseteq \updownarrow(x, \alpha)$, we know that $y \preceq x$ and $\forall a \in \alpha$, $a \sqcap y \in \downarrow\beta$. As $x \preceq y$ and $y \preceq x$, we thus have $x = y$.

Since $x = y$, by definition of canonical form of pseudo-elements, we have $\forall b \in \beta, b \sqcap x = b \sqcap y = b \in \downarrow\alpha$ and $\forall a \in \alpha, a \sqcap y = a \in \downarrow\beta$. It follows by Proposition 4.2 that $\downarrow\alpha \subseteq \downarrow\beta$ and $\downarrow\beta \subseteq \downarrow\alpha$, i.e. $\downarrow\alpha = \downarrow\beta$. Since antichains are canonical representations of closed sets, we finally get $\alpha = \beta$, which terminates the proof. □

4.2.2 Pseudo-antichains

We are now ready to introduce the new structure of pseudo-antichain.

Definitions. A *pseudo-antichain* A is a finite set of pseudo-elements, that is, $A = \{(x_i, \alpha_i) \mid i \in I\}$ with I finite. The *pseudo-closure* $\updownarrow A$ of A is defined as the set $\updownarrow A = \bigcup_{i \in I} \updownarrow(x_i, \alpha_i)$. Let (x_i, α_i), $(x_j, \alpha_j) \in A$. We have the two following observations:

1. If $x_i = x_j$, then (x_i, α_i) and (x_j, α_j) can be replaced in A by the pseudo-element $(x_i, \alpha_i \dot{\cap} \alpha_j)$.
2. If $\updownarrow(x_i, \alpha_i) \subseteq \updownarrow(x_j, \alpha_j)$, then (x_i, α_i) can be removed from A.

From these observations, we say that a pseudo-antichain $A = \{(x_i, \alpha_i) \mid i \in I\}$ is *simplified* if for all $i \in I$, (x_i, α_i) is in canonical form, and for all $i, j \in I$ such that $i \neq j$, $x_i \neq x_j$ and $\updownarrow(x_i, \alpha_i) \not\subseteq \updownarrow(x_j, \alpha_j)$. Notice that two distinct pseudo-antichains A and B can have the same pseudo-closure $\updownarrow A = \updownarrow B$ even if they are simplified. We thus say that A is a PA-*representation*[1] of $\updownarrow A$ (without saying

[1] "PA-representation" means pseudo-antichain based representation.

that it is a canonical representation), and that $\updownarrow A$ is PA-*represented* by A. For efficiency purposes, our algorithms always work on simplified pseudo-antichains.

Any antichain α can be seen as the pseudo-antichain $A = \{(x, \emptyset) \mid x \in \alpha\}$. Furthermore, notice that *any* set X can be represented by the pseudo-antichain $A = \{(x, \alpha_x) \mid x \in X\}$, with $\alpha_x = \lceil \{s \in S \mid s \preceq x \text{ and } s \neq x\} \rceil$. Indeed, $\updownarrow(x, \alpha_x) = \{x\}$ for all x, and thus $X = \updownarrow A$.

The interest of pseudo-antichains is that given two antichains α and β, the difference $\downarrow\alpha \backslash \downarrow\beta$ is PA-represented by the pseudo-antichain $\{(x, \beta) \mid x \in \alpha\}$, as indicated by the following lemma.

Lemma 4.8. *Let $\alpha, \beta \subseteq S$ be two antichains. Then $\downarrow\alpha \backslash \downarrow\beta = \updownarrow\{(x, \beta) \mid x \in \alpha\}$.*

Operations. The next proposition indicates how to compute pseudo-closures of pseudo-elements with respect to the union, intersection and difference operations. This method can be extended for computing the union, intersection and difference of pseudo-closures of pseudo-antichains, by using the classical properties from set theory like $X \setminus (Y \cup Z) = X \setminus Y \cap X \setminus Z$. From an algorithmic point of view, it is important to note that the computations only manipulate (pseudo-)antichains instead of their (pseudo-)closure.

Proposition 4.9. *Let $(x, \alpha), (y, \beta)$ be two pseudo-elements. Then:*

- $\updownarrow(x, \alpha) \cup \updownarrow(y, \beta) = \updownarrow\{(x, \alpha), (y, \beta)\}$,
- $\updownarrow(x, \alpha) \cap \updownarrow(y, \beta) = \updownarrow\{(x \sqcap y, \alpha \mathbin{\dot\cup} \beta)\}$, *and*
- $\updownarrow(x, \alpha) \setminus \updownarrow(y, \beta) = \updownarrow(\{(x, \{y\} \mathbin{\dot\cup} \alpha)\} \cup \{(x \sqcap b, \alpha) \mid b \in \beta\})$.

Notice that there is an abuse of notation in the previous proposition. Indeed, the definition of a pseudo-element (x, α) requires that $x \notin \downarrow\alpha$, whereas this condition could not be satisfied by pairs like $(x \sqcap y, \alpha \mathbin{\dot\cup} \beta)$, $(x, \{y\} \mathbin{\dot\cup} \alpha)$ and $(x \sqcap b, \alpha)$. This can be easily tested by Proposition 4.2, and when this happens, such a pair should not be added to the related pseudo-antichain. For instance, $\updownarrow(x, \alpha) \cap \updownarrow(y, \beta)$ is either equal to $\updownarrow\{(x \sqcap y, \alpha \mathbin{\dot\cup} \beta)\}$ or to $\updownarrow\{\}$. Notice also that the pseudo-antichains computed in the previous proposition are not necessarily simplified. However, our algorithms implementing those operations always simplify the computed pseudo-antichains for the sake of efficiency.

Proof (of Proposition 4.9). We prove the three statements:

- $\updownarrow(x,\alpha) \cup \updownarrow(y,\beta) = \updownarrow\{(x,\alpha),(y,\beta)\}$:
  This result comes directly from the definition of pseudo-closure of pseudo-antichains.
- $\updownarrow(x,\alpha) \cap \updownarrow(y,\beta) = \updownarrow\{(x \sqcap y, \alpha \dot\cup \beta)\}$:

  $$\begin{aligned} s \in \updownarrow(x,\alpha) \cap \updownarrow(y,\beta) \\ &\Leftrightarrow s \preceq x, s \notin \downarrow\alpha \text{ and } s \preceq y, s \notin \downarrow\beta \\ &\Leftrightarrow s \preceq x \sqcap y \text{ and } s \notin \downarrow\alpha \cup \downarrow\beta = \downarrow(\alpha \dot\cup \beta) \text{ (by Proposition 4.2)} \\ &\Leftrightarrow s \in \updownarrow\{(x \sqcap y, \alpha \dot\cup \beta)\} \end{aligned}$$

- $\updownarrow(x,\alpha) \setminus \updownarrow(y,\beta) = \updownarrow(\{(x,\{y\} \dot\cup \alpha)\} \cup \{(x \sqcap b, \alpha) \mid b \in \beta\})$:
  We prove the two inclusions:

  1. $\underline{\subseteq}$: Let $s \in \updownarrow(x,\alpha) \setminus \updownarrow(y,\beta)$, i.e. $s \in \updownarrow(x,\alpha)$ and $s \notin \updownarrow(y,\beta)$. Then, $s \preceq x$, $s \notin \downarrow\alpha$ and ($s \not\preceq y$ or $s \in \downarrow\beta$). Thus, if $s \not\preceq y$, then $s \in \updownarrow(x,\{y\} \dot\cup \alpha)$. Otherwise, $s \in \downarrow\beta$, i.e. $\exists b \in \beta$ such that $s \preceq b$. It follows that $s \in \updownarrow(x \sqcap b, \alpha)$.
  2. $\underline{\supseteq}$: Let $s \in \updownarrow(\{(x,\{y\} \dot\cup \alpha)\} \cup \{(x \sqcap b, \alpha) \mid b \in \beta\})$. Suppose first that $s \in \updownarrow(x,\{y\} \dot\cup \alpha)$. Then $s \preceq x$, $s \not\preceq y$ and $s \notin \downarrow\alpha$. We thus have $s \in \updownarrow(x,\alpha)$ and $s \notin \updownarrow(y,\beta)$. Suppose now that $\exists b \in \beta$ such that $s \in \updownarrow(x \sqcap b, \alpha)$. We have $s \preceq x$, $s \preceq b$ and $s \notin \downarrow\alpha$. It follows that $s \in \updownarrow(x,\alpha)$ and $s \in \downarrow\beta$, thus $s \notin \updownarrow(y,\beta)$.

□

The following example illustrates the second and third statements of Proposition 4.9.

Example 4.10. Let $\langle S, \preceq\rangle$ be a lower semilattice and let $(x,\{a\})$ and $(y,\{b\})$, with $x,y,a,b \in S$, be two pseudo-elements as depicted in Figure 4.5. We have $\updownarrow(x,\{a\}) \cap \updownarrow(y,\{b\}) = \updownarrow(x \sqcap y, \{a,b\})$. We also have $\updownarrow(x,\{a\}) \setminus \updownarrow(y,\{b\}) = \updownarrow\{(x,\{y\}\dot\cup\{a\}),(x \sqcap b,\{a\})\} = \updownarrow\{(x,\{y\}),(b,\{a\})\}$. Note that $(x,\{y\})$ and $(b,\{a\})$ are not in canonical form. The canonical representation of $\updownarrow(x,\{y\})$ (resp. $\updownarrow(b,\{a\})$) is given by $(x,\{x \sqcap y\})$ (resp. $(b,\{b \sqcap a\})$). ◁

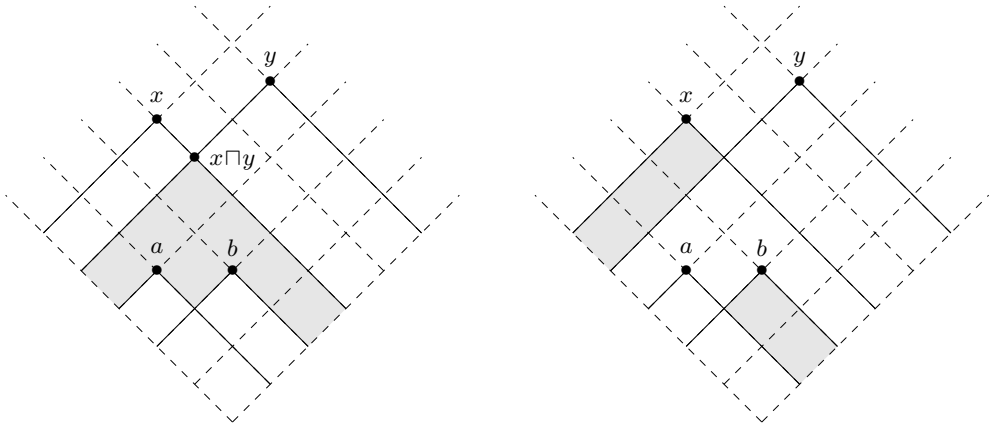

Figure 4.5: Intersection (left) and difference (right) of pseudo-closure of pseudo-elements.

4.3 Library AaPAL

We have implemented a generic library, called AaPAL (Antichain and Pseudo-Antichain Library), for the manipulation of antichains and pseudo-antichains. The library especially provides an efficient implementation of the operations described in Propositions 4.2 and 4.9. AaPAL has been implemented in C and is publicly available at `http://lit2.ulb.ac.be/aapal`. It has been used in a tool [Aca] and a prototype [STR] for which promising experimental results have been reported [BBF+12, BBFR13, BBR14a]. We come back to those implementations in the remaining chapters.

Part II

Synthesis of worst-case winning strategies

CHAPTER 5

Synthesis from LTL specifications

We open Part II by considering the problem of synthesis from LTL specifications. We first formally define the studied problem and related notions, and we give its computational complexity. We then present a so-called Safraless procedure that reduces the LTL synthesis problem to a safety game. We show how those games can be efficiently solved with antichains by exploiting a partial order on their state space. We discuss some optimizations of the proposed procedure and we present an interesting approach to efficiently handle specifications expressed by conjunctions of LTL formulas. Finally, we present the tool Acacia+ which provides an implementation of the antichain based Safraless algorithm. We consider several application scenarios of Acacia+ and we report experimental results.

The content of this chapter is based on [FJR11] and [BBF+12].

5.1 Definitions and computational complexity

In this section, we formally define the LTL realizability and synthesis problems and related notions. We also state their computational complexity.

LTL realizability and synthesis problems. The realizability problem for LTL is best seen as a game between two players. Let ϕ be an LTL formula over the set $P = I \uplus O$ partitioned into I, the set of *input signals* controlled by Player I

(the environment), and O the set of *output signals*, controlled by Player O (the system). With this partition of P, we associate the three alphabets $\Sigma_P = 2^P$, $\Sigma_O = 2^O$, and $\Sigma_I = 2^I$.

The realizability game is played in turns. Player O starts by giving $o_0 \in \Sigma_O$, Player I responds by giving $i_0 \in \Sigma_I$, then Player O gives $o_1 \in \Sigma_O$ and Player I responds by $i_1 \in \Sigma_I \dots$ This game lasts forever and the outcome of the game is the infinite word $(o_0 \cup i_0)(o_1 \cup i_1)(o_2 \cup i_2) \cdots \in \Sigma_P^\omega$.

The players play according to *strategies*. A strategy for Player O is a mapping $\lambda_O : (\Sigma_O\Sigma_I)^* \to \Sigma_O$, while a strategy for Player I is a mapping $\lambda_I : (\Sigma_O\Sigma_I)^*\Sigma_O \to \Sigma_I$. The outcome of the strategies λ_O and λ_I is the word $\mathsf{Outcome}(\lambda_O, \lambda_I) = (o_0 \cup i_0)(o_1 \cup i_1) \dots$ such that $o_0 = \lambda_O(\epsilon)$, $i_0 = \lambda_I(o_0)$ and for all $k \geq 1$, $o_k = \lambda_O(o_0 i_0 \dots o_{k-1} i_{k-1})$ and $i_k = \lambda_I(o_0 i_0 \dots o_{k-1} i_{k-1} o_k)$. We denote by $\mathsf{Outcome}(\lambda_O)$ the set of all outcomes $\mathsf{Outcome}(\lambda_O, \lambda_I)$ with λ_I any strategy of Player I. We let Π_O (resp. Π_I) be the set of strategies for Player O (resp. Player I).

Given an LTL formula ϕ, called the *specification*, over P, and a partition of P into I and O, the LTL *realizability problem* is to decide whether there exists a strategy λ_O of Player O such that $\mathsf{Outcome}(\lambda_O, \lambda_I) \models \phi$ against all strategies λ_I of Player I. If such a *winning* strategy exists, we say that the specification ϕ is *realizable*. Otherwise, we say that ϕ is *unrealizable*. The LTL *synthesis problem* asks to produce a strategy λ_O that realizes ϕ, when it is realizable.

Moore machines It is known that the LTL realizability problem is 2ExpTime-complete and that finite-memory strategies suffice in the case of a realizable specification [PR89b, Ros92]. A strategy λ_O of Player O is *finite-memory* if there exists a right-congruence $\sim$ on $(\Sigma_O\Sigma_I)^*$ of finite index such that $\lambda_O(u) = \lambda_O(u')$ for all $u \sim u'$. It is equivalent to say that it can be described by a Moore machine.

Definition 5.1 (Moore machine). A *Moore machine* is a tuple $\mathcal{M} = (M, m_0, \alpha_U, \alpha_N)$ where:

- the non-empty set M is the finite memory of $\mathcal{M}$, that is, M is the set of equivalence classes for $\sim$,
- m_0 is the initial memory state,
- $\alpha_U : M \times \Sigma_I \to M$ is the memory update function that modifies the current memory state at each $i \in \Sigma_I$ emitted by Player I, and

- $\alpha_N : M \to \Sigma_O$ is the next-move function that indicates which $o \in \Sigma_O$ is proposed by Player O given the current memory state.

The function α_U is naturally extended to words $u \in \Sigma_I^*$. The *language* of $\mathcal{M}$, denoted by $\mathcal{L}(\mathcal{M})$, is the set of words $u = (o_0 \cup i_0)(o_1 \cup i_1)\cdots \in \Sigma_P^\omega$ such that $o_0 = \alpha_N(m_0)$ and for all $k \geq 1$, $o_k = \alpha_N(\alpha_U(m_0, i_0 \dots i_{k-1}))$. The size $|\mathcal{M}|$ of a Moore machine is defined as the size $|M|$ of its memory. Therefore, with these notations, an LTL formula is realizable if and only if there exists a Moore machine $\mathcal{M}$ such that $\mathcal{L}(\mathcal{M}) \subseteq [\![\phi]\!]$.

Theorem 5.2 ([PR89b, Ros92])**.** *The* LTL *realizability problem is 2ExpTime-complete and any realizable* LTL *formula is realizable by a finite-memory strategy.*

Example 5.3. Let $I = \{q\}$, $O = \{p\}$. The formula $p\mathsf{U}q$ is not realizable. Indeed, q being an input signal, Player I can decide to leave it always false and the outcome does not satisfy ϕ. However, the formula $\Diamond q \Rightarrow (p\mathsf{U}q)$ is realizable. The assumption $\Diamond q$ states that q will hold at some point, and so, a possible winning strategy for Player O is to always assert p. ◁

5.2 Safraless algorithm

The classical procedure for solving the LTL realizability problem is based on Safra's construction to get a deterministic automaton $\mathcal{A}_\phi$ such that $\mathcal{L}(\mathcal{A}_\phi) = [\![\phi]\!]$ (see Theorem 2.9). However, this construction is intricate and notoriously difficult to implement efficiently [ATW06], even if recent progresses have been made [Sch09, TFVT10]. In this section, we recall the so-called *Safraless construction*, proposed in [SF07, FJR11], with a reduction to safety games (SGs). To this end, we need to introduce two acceptance conditions for infinite word automata.

5.2.1 New acceptance conditions

Let $\mathcal{A} = (\Sigma, Q, q_0, \Omega, \delta)$ be an infinite word automata. We consider the following acceptance conditions given by a subset $\Omega = \alpha$ with $\alpha \subseteq Q$ of final states.

Universal co-Büchi. A run $\rho \in Q^\omega$ is accepting in a *universal co-Büchi automaton (UCB)* $\mathcal{A}$ if and only if $\forall q \in \alpha, \mathsf{Visit}(\rho, q) < \infty$.

Universal K-co-Büchi. A run $\rho \in Q^\omega$ is accepting in a *universal K-co-Büchi automaton (UKCB)* $\langle \mathcal{A}, K \rangle$ if and only if $\forall q \in \alpha, \mathsf{Visit}(\rho, q) \leq K$.

The infinite word language $\mathcal{L}_{\text{ucb}}(\mathcal{A})$ (resp. $\mathcal{L}_{\text{ucb},K}(\mathcal{A})$) recognized by the UCB $\mathcal{A}$ (resp. by the UKCB $\langle \mathcal{A}, K \rangle$) is the set of words $u \in \Sigma^\omega$ such that *all* runs of $\mathcal{A}$ (resp. $\langle \mathcal{A}, K \rangle$) on u are accepting. Notice that if $K \leq K'$, then $\mathcal{L}_{\text{ucb},K}(\mathcal{A}) \subseteq \mathcal{L}_{\text{ucb},K'}(\mathcal{A})$.

Determinization of UKCB. The interest of UKCB is that they can be determinized with the subset construction extended with counters [SF07, FJR11]. Intuitively, for all states q of $\mathcal{A}$, we count (up to $\top = K+1$) the maximal number of final states which have been visited by runs ending in q. This counter is equal to -1 when no run ends in q. The final states are the subsets in which a state has its counter greater than K (accepted runs will avoid them). Formally, let $\mathcal{A}$ be a UKCB$(\Sigma_P, Q, q_0, \alpha, \delta)$ with $K \in \mathbb{N}$. Let $\mathcal{K} = \{-1, 0 \ldots, K, \top\}$ (with $\top > K$). We define $\det(\mathcal{A}, K) = (\Sigma_P, \mathcal{F}, F_0, \beta, \Delta)$ where:

- $\mathcal{F} = \{F \mid F \text{ is a mapping from } Q \text{ to } \mathcal{K}\}$
- $F_0 = q \in Q \mapsto \begin{cases} -1 & \text{if } q \neq q_0 \\ (q_0 \in \alpha) & \text{otherwise} \end{cases}$
- $\beta = \{F \in \mathcal{F} \mid \exists q, F(q) = \top\}$
- $\Delta(F, \sigma) = q \mapsto \max\{F(p) \oplus (q \in \alpha) \mid q \in \delta(p, \sigma)\}$

In this definition, $(q \in \alpha) = 1$ if q is in α, and 0 otherwise; we use the operator $\oplus : \mathcal{K} \times \{0, 1\} \to \mathcal{K}$ such that $k \oplus b = -1$ if $k = -1$, $k \oplus b = k + b$ if $\{k \neq -1, \top$ and $k + b \leq K\}$, and $k \oplus b = \top$ in all other cases. The automaton $\det(\mathcal{A}, K)$ has the following properties.

Proposition 5.4. *Let $K \in \mathbb{N}$ and $\langle \mathcal{A}, K \rangle$ be an* UKCB*. Then $\det(\mathcal{A}, K)$ is a deterministic and complete automaton such that $\mathcal{L}_{\text{ucb},0}(\det(\mathcal{A}, K)) = \mathcal{L}_{\text{ucb},K}(\mathcal{A})$.* □

Example 5.5. Let $\phi = \Box(r \Rightarrow \mathsf{X}\Diamond g)$ be an LTL formula over $P = \{r, g\}$ which states that a every request (r) must be eventually granted (g) after its emission. Figure 5.1 (left side) represents a UCB $\mathcal{A}$ over Σ_P equivalent to ϕ. Note that this UCB is not complete. Missing transitions are directed to an additional non-final sink state that we do not represent for the sake of readability. The label true

on a transition from q to q' means that there is a transition from q to q' for all $\sigma \in \Sigma_P$. If a request is never granted, then a run will visit the final state q_1 infinitely often, and this run will not be accepting. Figure 5.1 (right side) represents the deterministic and complete automaton $\det(\mathcal{A}, 1)$ obtained by the determinization procedure described above, where $F_0 = (q_0 \mapsto 0, q_1 \mapsto -1)$, $F_1 = (q_0 \mapsto 0, q_1 \mapsto 1)$ and $F_2 = (q_0 \mapsto 0, q_1 \mapsto \top)$. For the sake of readability, states that are not reachable from F_0 are not represented. One can easily see that $\mathcal{L}_{\text{ucb},0}(\det(\mathcal{A}, 1)) = \mathcal{L}_{\text{ucb},1}(\mathcal{A})$. ◁

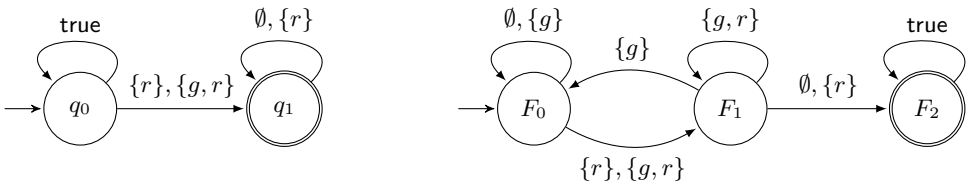

Figure 5.1: UCB $\mathcal{A}$ equivalent to $\square(r \Rightarrow \mathsf{X}\Diamond g)$ (left) and UCB $\det(\mathcal{A}, 1)$ (right).

5.2.2 Reduction to safety games

We now go through a series of results that allow us to construct, given an LTL formula ϕ, a SG from a UCB $\mathcal{A}$ equivalent to ϕ (see Theorem 5.8). The results of this section come from [FJR11].

Proposition 5.6. *Let ϕ be an LTL formula over P. Then, there exists a UCB $\mathcal{A}$ such that $\mathcal{L}_{\text{ucb}}(\mathcal{A}) = [\![\phi]\!]$. Moreover, with the related UCB $\mathcal{A}$, the formula ϕ is realizable if and only if there exists a Moore machine $\mathcal{M}$ such that $\mathcal{L}(\mathcal{M}) \subseteq \mathcal{L}_{\text{ucb}}(\mathcal{A})$.*

Proof. Let us take the negation $\neg\phi$ of ϕ. From Theorem 2.7, we have that there exists a NB $\mathcal{A}$ such that $\mathcal{L}_{\text{b}}(\mathcal{A}) = [\![\neg\phi]\!]$; moreover $\overline{\mathcal{L}_{\text{b}}(\mathcal{A})} = \mathcal{L}_{\text{ucb}}(\mathcal{A})$ as the accepting conditions are dual as well as the transition functions. In this way, we get the first part of the proposition. The second part follows from the definition of Moore machines. □

The next theorem states that the co-Büchi acceptance condition can be strengthened into a K-co-Büchi acceptance condition, providing that K is taken large enough.

Theorem 5.7. *Let ϕ be an* LTL *formula over P and $\mathcal{A}$ be a* UCB *with n states such that $\mathcal{L}_{\text{ucb}}(\mathcal{A}) = [\![\phi]\!]$. Let $\mathbb{K} = n(n^{2n+2} + 1)$. Then, ϕ is realizable if and only if there exists a Moore machine $\mathcal{M}$ such that $\mathcal{L}(\mathcal{M}) \subseteq \mathcal{L}_{\text{ucb},\mathbb{K}}(\mathcal{A})$.*

Consequently to Proposition 6.16 and Theorem 5.7, we finally get the next theorem establishing the reduction of the LTL synthesis problem to a SG. A precise description of this SG is given in the next section.

Theorem 5.8. *Let ϕ be an* LTL *formula. Then, one can construct a* SG *in which Player 1 has a winning strategy if and only if ϕ is realizable.*

5.3 Antichain based algorithms

In Section 5.2, we have shown how to reduce the LTL realizability problem to a SG. In this section we explain how to efficiently and symbolically solve this SG with antichains. The results presented in this section come from [FJR11].

5.3.1 Description of the safety game

From Theorem 5.8, for all LTL formulas ϕ, there exists a SG $\langle \mathcal{G}_{\phi,\mathbb{K}}, \alpha \rangle$ which depends on the constant $\mathbb{K}$ of Theorem 5.7, such that Player 1 has a winning strategy in $\langle \mathcal{G}_{\phi,\mathbb{K}}, \alpha \rangle$ if and only if ϕ is realizable. Let us give a precise description of this SG, but more generally for any value $K \in \mathbb{N}$ instead of $\mathbb{K}$. Notice that this SG is turn-based. Let $\mathcal{A} = (\Sigma_P, Q, q_0, \alpha, \delta)$ be a UCB such that $\mathcal{L}_{\text{ucb}}(\mathcal{A}) = [\![\phi]\!]$, and $\det(\mathcal{A}, K) = (\Sigma_P, \mathcal{F}, F_0, \beta, \Delta)$ be the related deterministic automaton. The turn-based SG $\langle \mathcal{G}_{\phi,K}, \alpha \rangle$ with $\mathcal{G}_{\phi,K} = (S = S_1 \uplus S_2, s_0, E)$ has the following structure:

- $S_1 = \{(F, i) \mid F \in \mathcal{F}, i \in \Sigma_I\}$,
- $S_2 = \{(F, o) \mid F \in \mathcal{F}, o \in \Sigma_O\}$,
- $s_0 = (F_0, j_0)$ where j_0 is an arbitrary symbol of Σ_I, and F_0 is the initial state of $\det(\mathcal{A}, K)$,
- For all $\Delta(F, o \cup i) = F'$ and $j \in \Sigma_I$, the set E contains the edges

$$((F, j), (F, o)) \text{ and } ((F, o), (F', i)), \text{ and}$$

- $\alpha = S \setminus \{(F, \sigma) \in S \mid \exists q, F(q) = \top\}$.

Remark 5.9. It is possible to reduce the size of the SG $\langle \mathcal{G}_{\phi,K}, \alpha \rangle$ such that $S_1 = \mathcal{F}$ instead of $\{(F, i) \mid F \in \mathcal{F},\ i \in \Sigma_I\}$. However, our definition simplifies the presentation of the remaining of this section. ◁

Example 5.10. Figure 5.2 presents the turn-based SG $\langle \mathcal{G}_{\phi,K} = (S, s_0, E), \alpha \rangle$ with $K = 1$ and $\phi = \square(r \Rightarrow \mathsf{X}\Diamond g)$ constructed with the procedure described above from the deterministic UCB of Figure 5.1 (right side) for the partition $I = \{r\}$ and $O = \{g\}$. For the sake of readability, only reachable states from s_0 are represented. Note that for all $F \in \mathcal{F}$, there are two distinct states $(F, \emptyset) \in S_1$ with $\emptyset \in \Sigma_O$ and $(F, \emptyset) \in S_2$ with $\emptyset \in \Sigma_I$. The set α contains all states $(F, \sigma) \in S$ such that $F \neq F_2 = (q_0 \mapsto 0, q_1 \mapsto \top)$. One can easily see that every strategy λ_1 for Player 1 such that $\lambda_1(S^*(F_1, \{r\})) = (F_1, \{g\})$ is winning. ◁

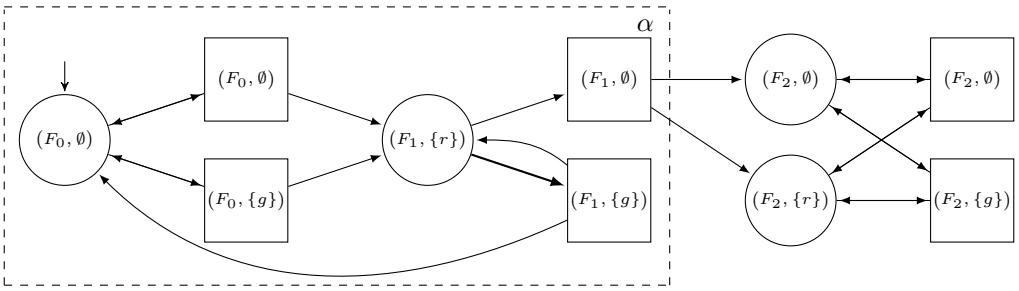

Figure 5.2: Turn-based SG $\langle \mathcal{G}_{\phi,K}, \alpha \rangle$ with $K = 1$ and $\phi = \square(r \Rightarrow \mathsf{X}\Diamond g)$.

Incremental algorithm. Notice that given $K_1, K_2 \in \mathbb{N}$ such that $K_1 \leq K_2$, if Player 1 has a winning strategy in the SG $\langle \mathcal{G}_{\phi,K_1}, \alpha \rangle$, then he has a winning strategy in the SG $\langle \mathcal{G}_{\phi,K_2}, \alpha \rangle$. The next corollary of Theorem 5.8 holds.

Corollary 5.11. *Let ϕ be an* LTL *formula and $K \in \mathbb{N}$. If Player* 1 *has a winning strategy in the* SG $\langle \mathcal{G}_{\phi,K}, \alpha \rangle$*, then ϕ is realizable.* □

This property indicates that testing whether ϕ is realizable can be done *incrementally* by solving the family of SGs $\langle \mathcal{G}_{\phi,K}, \alpha \rangle$ with increasing values of $K \geq 0$ until either Player 1 has a winning strategy in $\langle \mathcal{G}_{\phi,K}, \alpha \rangle$ for some K such that $0 \leq K \leq \mathbb{K}$, or Player 1 has no winning strategy in $\langle \mathcal{G}_{\phi,\mathbb{K}}, \alpha \rangle$.

Example 5.12. To illustrate the incremental algorithm, let us consider the following series of specifications with an increasing number of nested X temporal

operators. Those specifications stand for systems that must grant requests made by the environment after a fixed number of steps. Let

$$\begin{aligned}\phi_1 &= \square(r \Rightarrow \mathsf{X}g)\\ \phi_2 &= \square(r \Rightarrow \mathsf{XX}g)\\ \phi_3 &= \square(r \Rightarrow \mathsf{XXX}g)\\ &\vdots\end{aligned}$$

where $I = \{r\}$ and $O = \{g\}$. The minimum value of K required for Player 1 to have a winning strategy in the SG $\langle \mathcal{G}_{\phi_i,K}, \alpha\rangle$ increases with the number of nested X's. For instance, ϕ_1 can be solved with $K = 1$, ϕ_2 with $K = 2$, ϕ_3 with $K = 3\ldots$ Let us give some intuition with the example of ϕ_2 and its equivalent UCB $\mathcal{A}_{\phi_2}$ over the alphabet $\Sigma_P = \{\emptyset, \{g\}, \{r\}, \{g, r\}\}$ depicted in Figure 5.3. Note that this automaton is not complete (see state q_2). It has a sequence of 3 final states. As it is a UCB, accepting runs containing a $\{r\}$ or a $\{g, r\}$ are those leaving the final state q_2 with a $\{g\}$ or a $\{g, r\}$, that is, g must be played. In general for ϕ_i, the moment to play such a g increases with the length of the sequence of final states (i.e. with the number of nested X's), and so does the minimum value of K required for Player 1 to have a winning strategy in the SG $\langle \mathcal{G}_{\phi_i,K}, \alpha\rangle$. This also illustrates that we do not need to consider large values of K. ◁

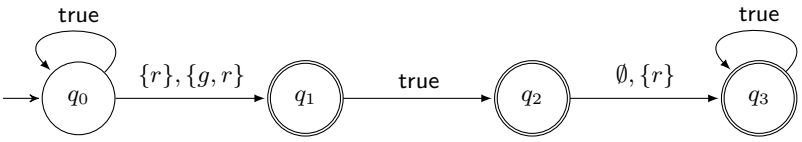

Figure 5.3: UCB equivalent to $\square(r \Rightarrow \mathsf{XX}g)$.

Unrealizable specifications. The incremental algorithm is not reasonable to test unrealizability, since it would be necessary to reach the theoretical bound $\mathbb{K} = n(n^{2n+2} + 1)$ to conclude for unrealizability. We obtain a more practical algorithm by relying on the determinacy of ω-regular games [Mar75].

Theorem 5.13. *Let ϕ be an* LTL *formula. Then, either Player* 1 *has a winning strategy λ_1 such that for all strategy λ_2 of Player* 2, $\mathsf{Outcome}(\lambda_1, \lambda_2) \models \phi$, *or*

Player 2 *has a winning strategy* λ_2 *such that, for all strategy* λ_1 *of Player* 1, $\mathsf{Outcome}(\lambda_1, \lambda_2) \models \neg\phi$.

From Theorem 5.13, it follows that when an LTL specification ϕ is not realizable for Player 1, its negation $\neg\phi$ is realizable for Player 2. In practice, to avoid the enumeration of values of K up to $\mathbb{K}$, the following algorithm can be applied. Given an LTL formula ϕ, we construct two UCBs: one that recognizes the language $[\![\phi]\!]$, and one that recognizes the language $[\![\neg\phi]\!]$. Then, we check in parallel the realizability of ϕ for Player 1 and the realizability of $\neg\phi$ for Player 2, for increasing values of $K \geq 0$. The algorithm stops as soon as a winning strategy is found for a given $K \in \mathbb{N}$, either for Player 1 in the SG $\langle \mathcal{G}_{\phi,K}, \alpha \rangle$ (which means that ϕ is realizable), or for Player 2 in the SG $\langle \mathcal{G}_{\neg\phi,K}, \alpha \rangle$ (which means that ϕ is not realizable).

5.3.2 Antichains

In the previous section, we have shown how the LTL realizability problem can be reduced to a family of SGs $\langle \mathcal{G}_{\phi,K}, \alpha \rangle$ with $0 \leq K \leq \mathbb{K}$. In this section, we show that the states of those SGs can be partially ordered, and the sets manipulated by the fixpoint algorithm for solving them can be represented by the antichain of their maximal elements. Therefore, it is not necessary to construct explicitly the SGs $\langle \mathcal{G}_{\phi,K}, \alpha \rangle$, which may be very large even for small values of K.

Partial order on game states. Consider the SG $\langle \mathcal{G}_{\phi,K}, \alpha \rangle$ with $\mathcal{G}_{\phi,K} = (S = S_1 \uplus S_2, s_0, E)$ as defined in the previous section. We define the binary relation $\preceq \subseteq S \times S$ by

$$(F', \sigma) \preceq (F, \sigma) \text{ if and only if } F' \leq F$$

where $F, F' \in \mathcal{F}$, $\sigma \in \Sigma_P$ and $F' \leq F$ if and only if $F'(q) \leq F(q)$ for all q. It is clear that $\preceq$ is a partial order. Intuitively, if Player 1 can win the SG from a state (F, σ), then he can also win from all states $(F', \sigma) \preceq (F, \sigma)$ as it is somehow more difficult to avoid $\top$ from F than from F'. Formally, $\preceq$ is a game simulation relation in the terminology of [AHKV98]. We have the following lemma establishing that the function Δ is monotonic.

Lemma 5.14. *For all* $F, F' \in \mathcal{F}$ *such that* $F' \leq F$ *and* $\sigma \in \Sigma_P$, *we have* $\Delta(F', \sigma) \leq \Delta(F, \sigma)$.

Backward algorithm with antichains. We recall the fixpoint algorithm to check whether Player 1 has a winning strategy in the SG $\langle \mathcal{G}_{\phi,K} = (S, s_0, E), \alpha \rangle$ (cf. Section 2.4.2). In the sequel, this algorithm is called the *backward* algorithm as it explores the game in a backward fashion. Given a set $L \subseteq S$, this algorithm uses notations $\mathsf{CPre}_1(L) = \{s \in S_1 \mid \exists (s, s') \in E, s' \in L\}$ and $\mathsf{CPre}_2(L) = \{s \in S_2 \mid \forall (s, s') \in E, s' \in L\}$. It computes the fixpoint $\mathsf{Win}_1(\alpha)$ of the sequence $W_0 = \alpha$, $W_{k+1} = W_k \cap \{\mathsf{CPre}_1(W_k) \cup \mathsf{CPre}_2(W_k)\}$ for all $k \geq 0$. Player 1 has a winning strategy in $\langle \mathcal{G}_{\phi,K}, \alpha \rangle$ if and only if $s_0 \in \mathsf{Win}_1(\alpha)$. This algorithm can be efficiently implemented with antichains. Indeed, we have the next lemma, the proof of which is based on Lemma 5.14.

Lemma 5.15. *If $L \subseteq S$ is a closed set, then $\mathsf{CPre}_1(L)$ and $\mathsf{CPre}_2(L)$ are also closed.*

By definition of the SG $\langle \mathcal{G}_{\phi,K}, \alpha \rangle$, the set α is closed. Therefore, by Proposition 4.2 and Lemma 5.15, the sets W_k manipulated by the backward algorithm are closed for all $k \geq 0$, and can thus be compactly represented by the antichain $\lceil W_k \rceil$ of their maximal elements. Moreover, those sets can be manipulated efficiently, as indicated by the next lemma.

Lemma 5.16. *Let $L \subseteq S$ be an antichain. Then, the computation of $\mathsf{CPre}_1(\downarrow L)$ and $\mathsf{CPre}_2(\downarrow L)$ can be limited to the antichain L.*

Finally, when Player 1 has a winning strategy in the SG $\langle \mathcal{G}_{\phi,K}, \alpha \rangle$, that is, when ϕ is realizable, it is easy to extract from the antichain $\lceil \mathsf{Win}_1(\alpha) \rceil$ a Moore machine that realizes it. Recall that $\langle \mathcal{G}_{\phi,K}, \alpha \rangle$ is constructed from the deterministic automaton $\det(\mathcal{A}, K) = (\Sigma_P, \mathcal{F}, F_0, \beta, \Delta)$. In this paragraph, for the sake of simplicity, we assume that $\langle \mathcal{G}_{\phi,K}, \alpha \rangle$ is simplified as suggested in Remark 5.9, that is, $\mathcal{G}_{\phi,K} = (S = S_1 \uplus S_2, s_0, E)$ is such that $S_1 = \mathcal{F}$.[1] By definition of CPre_1 and CPre_2, for all $F \in \lceil \mathsf{Win}_1(\alpha) \rceil \cap S_1$, there exists $o_F \in \Sigma_O$ such that for all $i \in \Sigma_I$, $\Delta(F, o_F \cup i) \in \downarrow \lceil \mathsf{Win}_1(\alpha) \rceil \cap S_1$, and this o_F can be computed. We can thus extract a Moore machine $\mathcal{M} = (M, m_0, \alpha_U, \alpha_N)$ with memory $M = \lceil \mathsf{Win}_1(\alpha) \rceil \cap S_1$ as follows. The initial state m_0 is some $F \in M$ such that $F_0 \preceq F$ (it exists if ϕ is realizable). The next-move function α_N maps any $F \in M$ to $o_F \in \Sigma_O$ such that, for all $i \in \Sigma_I$, $\Delta(F, o_F \cup i) \in \downarrow \lceil \mathsf{Win}_1(\alpha) \rceil \cap S_1$, and

[1] In this case, for all $F, F' \in S_1$, we have $F' \preceq F$ if and only if $F' \leq F$.

the update function α_U maps any pair $(F, i) \in M \times \Sigma_I$ to a state $F' \in M$ such that $\Delta(F, \alpha_N(F) \cup i) \preceq F'$ (it exists by definition of $\lceil \mathsf{Win}_1(\alpha) \rceil$ and Lemma 5.14). We thus have the next theorem, which highlights that the backward algorithm has the main advantage of producing *compact* strategies (see Section 5.7 for experimental results).

Theorem 5.17. *Let $\langle \mathcal{G}_{\phi,K}, \alpha \rangle$ with $\mathcal{G}_{\phi,K} = (S = S_1 \uplus S_2, s_0, E)$ be a* SG*. Then, there exists a Moore machine $\mathcal{M}$ with $|\mathcal{M}| \leq |\lceil \mathsf{Win}_1(\alpha) \rceil \cap S_1|$ that encodes a winning strategy for Player* 1 *in $\langle \mathcal{G}_{\phi,K}, \alpha \rangle$.*

Example 5.18. Let us consider again the turn-based SG of Figure 5.2 (page 65) to illustrate the extraction of a Moore machine from the antichain $\lceil \mathsf{Win}_1(\alpha) \rceil$. Applied to this SG, the antichain based backward algorithm computes the fixpoint $\lceil \mathsf{Win}_1(\alpha) \rceil$ such that $\lceil \mathsf{Win}_1(\alpha) \rceil \cap S_1 = \{(q_0 \mapsto 0, q_1 \mapsto 1)\} = \{F_1\}$. From this set, we construct a Moore machine $\mathcal{M} = (M, m_0, \alpha_U, \alpha_N)$ such that $M = \{F_1\}$ and $m_0 = F_1$. The next-move function α_U is defined such that $\alpha_N(F_1) = \{g\}$, since $\Delta(F_1, \{g\} \cup \emptyset) = F_0 \in \downarrow \lceil \mathsf{Win}_1(\alpha) \rceil$ and $\Delta(F_1, \{g\} \cup \{r\}) = F_1 \in \downarrow \lceil \mathsf{Win}_1(\alpha) \rceil$. Finally, the update function α_U is defined such that $\alpha_U(F_1, \emptyset) = \alpha_U(F_1, \{r\}) = F_1$. $\lhd$

Forward algorithm with antichains. The proposed algorithm for solving the SG $\langle \mathcal{G}_{\phi,K}, \alpha \rangle$ works in a backward manner. An alternative algorithm, called the *forward* algorithm, is proposed in [FJR11]. This algorithm is a variant of the OTFUR algorithm of [CDF+05]. It explores the states of the SG in a forward fashion, starting from the initial state, looking for a winning strategy for Player 1. As for the backward algorithm, it is not necessary to construct the game explicitly and antichains can again be used [FJR11]. Indeed, during the execution of this forward algorithm, the successors of a state (F, σ) are computed on demand. Compared to the backward algorithm, the forward algorithm has the following advantage: it only computes winning states (F, σ) (for Player 1) which are reachable from the initial state. Nevertheless, it computes a single winning strategy if it exists, whereas the backward algorithm computes a fixpoint from which we can easily enumerate the set of all winning strategies (in the SG).

The backward and forward algorithms presented in this section are called *monolithic* in the sequel.

5.4 Optimizations

In this section, we describe two optimizations of the antichain based algorithms for solving the LTL realizability and synthesis problems. Let ϕ be an LTL formula over $P = I \uplus O$, $\mathcal{A} = (\Sigma_P, Q, q_0, \alpha, \delta)$ be a UCB equivalent to ϕ, and $\langle \mathcal{G}_{\phi,K}, \alpha \rangle$ with $\mathcal{G}_{\phi,K} = (S = S_1 \uplus S_2, s_0, E)$ be the related SG, for some $K \in \mathbb{N}$.

Unbounded states. This first optimization [Fil11] applies on the UCB $\mathcal{A}$ with the aim of reducing the set α of final states. This optimization tends towards decreasing the minimum value of K required for Player 1 to have a winning strategy in $\langle \mathcal{G}_{\phi,K}, \alpha \rangle$. It computes an under-approximation of the set $B \subseteq Q$ of states q such that no run containing more than K final states may end up in q, for all $K \in \mathbb{N}$. States in B are called *bounded states*, while states in $Q \setminus B$ are called *unbounded states*. Formally, a state $q \in Q$ is *unbounded* if there exists a word $u \in \Sigma^\omega$ and a prefix $\rho(n) = q_0, \ldots, q_{n-1} \in Q^*$ of a run ρ on u with $q_{n-1} = q$ such that there exist $1 \leq i \leq j \leq k < n$ such that $q_i = q_k$ and $q_j \in \alpha$.

Example 5.19. Let us consider again the UCB $\mathcal{A}$ of Figure 5.3 (page 66) to illustrate the notions of (un)bounded states. State q_3 is the only unbounded state of $\mathcal{A}$. Indeed, states q_0, q_1 and q_2 are bounded since q_0 cannot be reached by a prefix of a run containing a final state, and q_1 (resp. q_2) cannot be reached by a prefix of a run containing more than 1 (resp. 2) final state(s). ◁

Before giving the algorithm to compute bounded states, we explain how to exploit them. Every final bounded state can be removed from the set α of final states without changing the language recognized by $\mathcal{A}$. Moreover, in the deterministic automaton $\det(\mathcal{A}, K) = (\Sigma_P, \mathcal{F}, F_0, \beta, \Delta)$, for all $K \in \mathbb{N}$, functions $F \in \mathcal{F}$ can map bounded states to $\{-1, 0\}$ instead of $\{-1, 0, \ldots, K, \top\}$. Indeed, we have the next lemma.

Lemma 5.20. *Let $\mathcal{A} = (\Sigma_P, Q, q_0, \alpha, \delta)$ be a UCB, and $B \subseteq Q$ be the set of bounded states. Let $\mathcal{A}' = (\Sigma_P, Q, q_0, \alpha \setminus B, \delta)$, then $\mathcal{L}_{\text{ucb}}(\mathcal{A}) = \mathcal{L}_{\text{ucb}}(\mathcal{A}')$.*

Proof. It is clear that $\mathcal{L}_{\text{ucb}}(\mathcal{A}) \subseteq \mathcal{L}_{\text{ucb}}(\mathcal{A}')$, as for all $u \in \mathcal{L}_{\text{ucb}}(\mathcal{A})$, the runs on u visit less final states in $\mathcal{A}'$ than in $\mathcal{A}$.

For the converse, let $w \in \mathcal{L}_{\text{ucb}}(\mathcal{A}')$, and let $\rho \in Q^\omega$ be a run of $\mathcal{A}'$ on w. Since $\mathcal{A}'$ is a UCB, ρ visits a finite number of final states. Suppose that ρ is

Algorithm 1 BoundedStates(UCB $\mathcal{A} = (\Sigma_P, Q, q_0, \alpha, \delta)$)

for $q \in Q$ **do**
  $c[q] := 0$
$c[q_0] := (q_0 \in \alpha)$
repeat
  $c' =$ copy of c
  for $q \in Q$ **do**
    $c'[q] := \max_{q' \in Q \text{ s.t. } \exists \sigma, q \in \delta(q', \sigma)} \min(|\alpha| + 1, c[q'] + (q \in \alpha))$
until $c = c'$
return $\{q \in Q \mid c[q] < |\alpha| + 1\}$

not accepting in $\mathcal{A}$, that is, ρ visits an infinite number of final states in $\mathcal{A}$. Therefore, there is a state $q \in \alpha$ which occurs infinitely often in ρ, that is, $q \in \mathsf{Inf}(\rho)$. If $q \notin B$, then ρ would not be accepting in $\mathcal{A}'$. Therefore, $q \in B$, which is impossible by definition of B. Indeed, since $q \in \alpha$ and $q \in \mathsf{Inf}(\rho)$, we have that there exists a prefix $\rho(n) = q_0, \ldots, q_{n-1} \in Q^*$ of ρ with $q_{n-1} = q$ such that there exist $1 \leq i \leq j \leq k < n$ such that $q_i = q_k$ and $q_j \in \alpha$ (for instance, it holds for $q_i = q_j = q_k = q$), that is, q is unbounded. □

We now give the algorithm to compute the set B of bounded states (see Algorithm 1). This algorithm computes, for each state $q \in Q$, a counter value that represents the maximum number of final states (up to $|\alpha| + 1$) that can be visited by prefixes of runs ending up in q. The bounded states are then states with counter smaller than or equal to $|\alpha|$. The correctness of Algorithm 1 is given by the next lemma.

Lemma 5.21. *Given a* UCB $\mathcal{A}$*, Algorithm 1 returns the set of bounded states in* $\mathcal{A}$*.*

Proof. If a state q is bounded, then there is no way to increase its counter value up to $|\alpha| + 1$, since otherwise, there would be a prefix of run ending up in q that visits $|\alpha| + 1$ final states. Therefore, there would be a final state occurring twice on this prefix, contradicting the fact that q is bounded. The converse is trivial. □

Critical signals. The second optimization concerns the backward algorithm. We here only give the intuition of this optimization, which is fully detailed

in [FJR13]. It consists in computing, at each iteration k of the algorithm, a subset of relevant input signals $\Sigma_I^k \subseteq \Sigma_I$, called *critical signals*, which are sufficient to compute an over-approximation of the set $\mathsf{CPre}_2(W_{k-1})$. It results in a new operator CPre_2^c such that $\mathsf{CPre}_2^c(W_{k-1}) = \{s \in S \mid \forall (s, s') \in E, s' = (F, i) \in W_{k-1}, i \in \Sigma_I^k\}$, that is, it only considers the states $(F, i) \in W_{k-1}$ with $i \in \Sigma_I^k$. By using the operator CPre_2^c instead of CPre_2, the same fixpoint $\mathsf{Win}_1(\alpha)$ is reached in a potentially larger number of iterations. However, the computation at each iteration is simplified.

5.5 Compositional approach

In [FJR11], the authors also propose two *compositional* algorithms for LTL formulas of the form $\phi = \phi_1 \wedge \cdots \wedge \phi_n$, that is, conjunctions of sub-specifications. With those algorithms, the LTL realizability and synthesis problems are solved by first solving, for each sub-specification ϕ_i, the related SG $\langle \mathcal{G}_{\phi_i, K_i}, \alpha_i \rangle$ for some $K_i \in \mathbb{N}$, and then by composing the solutions according to the *parenthesizing* of ϕ. The compositional algorithms are able to solve the LTL realizability and synthesis problems for formulas that are several pages long, while the monolithic algorithms are often limited to much smaller formulas.

The first compositional algorithm follows a compositional *backward* approach such that at each stage of the parenthesizing, the antichains $\mathsf{Win}_1(\alpha_i)$ of the sub-specifications ϕ_i are computed by the backward algorithm and the antichain of the formula ϕ itself is also computed backward from the antichains $\mathsf{Win}_1(\alpha_i)$, for all i. In this approach, *all* the winning strategies for ϕ (for fixed $K_i \in \mathbb{N}$, for all i) are computed and compactly represented by the final antichain.

The second compositional algorithm follows a compositional *forward* approach such that at each stage of the parenthesizing, antichains are computed by the backward algorithm as explained before, except at the last stage where a forward algorithm seeks for *one* winning strategy by exploring the game graph on the fly in a forward fashion.

5.6 Tool Acacia+

In this section, we present Acacia+, a tool for solving the LTL realizability and synthesis problems. Acacia+ implements the antichain based algorithms de-

scribed in the previous sections. It has been developed with emphasis on modularity, code efficiency and usability, in the hope that other researchers will take up the available code and extend it. The remaining of this chapter is based on [BBF+12].

5.6.1 Programming choices

Acacia+ is written in Python and C. The C language is used for all the low level operations, while the orchestration of those operations is done with Python. The binding between these two languages is realized by the ctypes library of Python. This separation presents the following advantages. First, C being a low level language, it allows for efficient implementations of algorithms and data structures. Second, the simplicity of Python increases scalability and modularity and it reduces the risk of errors. Unfortunately, using Python also presents some drawbacks. Indeed, interfacing Python and C leads to light performance overhead. Nevertheless, we believe that the overhead is a small price to pay in comparison with the gain of simplicity.

Our implementation does not use BDDs, as they do not seem to be well adapted in this context, and might be outperformed by the use of antichains (e.g. [DDHR06, DR07]). We thus have instead used antichains and the library AaPAL [AaP] presented in Chapter 4.

5.6.2 Tool download and user interface

Acacia+ can be downloaded at `http://lit2.ulb.ac.be/acaciaplus/`. It can be installed under a single command-line version working both on Linux and MacOsX, or used directly via a web interface. The source is licensed under the GNU General Public License. The code is open and can be used, extended and adapted by the research community. For convenience, a number of examples and benchmarks have been pre-loaded in the web interface. A screenshot of the web interface is given in Figure 5.4.

5.6.3 Execution Parameters

Acacia+ offers many execution parameters, fully detailed in the web interface helper.

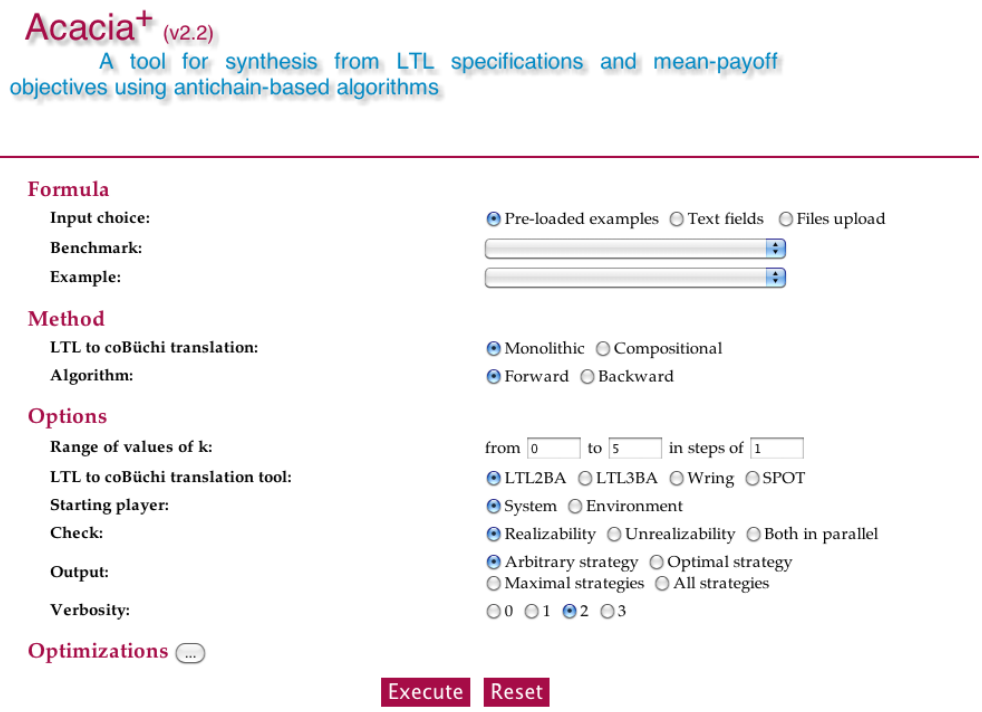

Figure 5.4: Web interface of Acacia+ (input form).

Formula. Two inputs are required: an LTL formula ϕ and a partition of the atomic signals into the sets I and O. The formula can be given as a single specification, or as a conjunction $\phi_1 \wedge \cdots \wedge \phi_n$ of several sub-specifications (for the compositional approach). Acacia+ accepts both the Wring and LTL2BA input formats.

Method. Formulas ϕ are processed in two steps. The first step constructs a UCB $\mathcal{A}_\phi$ equivalent to ϕ. The second step checks for realizability (synthesis follows when ϕ is realizable). The automaton construction can be done either monolithically (a single automaton $\mathcal{A}_\phi$ is associated with ϕ), or compositionally if the formula is given as a conjunction $\phi_1 \wedge \cdots \wedge \phi_n$ (an automaton $\mathcal{A}_{\phi_i}$ is associated with every ϕ_i). The realizability check can be either monolithic, or compositional (only if ϕ is a conjunction of ϕ_i's). When the automaton construction is compositional and the realizability step is monolithic, the latter starts with the union

of all automata $\mathcal{A}_{\phi_i}$.

The user can also choose between backward or forward algorithms for solving the underlying SG. In the case of a compositional realizability check with forward option enabled, each intermediate SG is solved backward and the whole game is solved forward. The way of parenthesizing ϕ is totally flexible: the user can specify his own parenthesizing or use predefined ones. This parenthesizing enforces the order in which the results of the intermediate SGs are combined, and may influence the performance.

Options. For the automaton construction, the user has the choice between four tools: LTL2BA [GO01], LTL3BA [BKRS12], Wring [SB00][2] or Spot [DLP04]. The user can also choose the starting player for the realizability game.[3] We recall that the implemented algorithms are incremental; the range of values of $K \in \mathbb{N}$ used in the SGs $\langle \mathcal{G}_{\phi,K}, \alpha \rangle$ can be specified.

The user can choose between either realizability check, or unrealizability check, or both in parallel. He can also ask for the synthesis of one *arbitrary* strategy, the set of *maximal* strategies or the set of *all* winning strategies[4]. In the first case, an arbitrary strategy is computed as described in Section 5.3 from $\lceil \mathsf{Win}_1(\alpha) \rceil \cap S_1$. In the second case, all strategies with memory $M = \lceil \mathsf{Win}_1(\alpha) \rceil \cap S_1$, called maximal strategies, are constructed with the procedure described in Section 5.3, but for all $F \in M$, we consider all $o_F \in \Sigma_O$ such that, for all $i \in \Sigma_I$, $\Delta(F, o_F \cup i) \in \downarrow \lceil \mathsf{Win}_1(\alpha) \rceil \cap S_1$. In the latter case, we explicitly consider the whole set $\downarrow \lceil \mathsf{Win}_1(\alpha) \rceil \cap S_1$ as memory of the strategies. In the case of synthesis of multiple strategies, a unique infinite word automaton representing all of them is output. Note that a fourth option of strategy synthesis is discussed in Section 8.2.2. Finally, Acacia+ contains several optimizations including those described in Section 5.4. All these optimizations are enabled by default, but can be turned off.

[2] Our tool uses the Wring module included in the tool Lily [JB06].

[3] In Section 5.1, the realizability game has been described such that Player O, the system, plays first. A variant is to let Player I, the environment, play first.

[4] Notice that by *all* winning strategies, we mean all winning strategies in the related SG $\langle \mathcal{G}_{\phi,K}, \alpha \rangle$, for some value $K \in \mathbb{N}$. This is thus a subset of all the winning strategies for Player O in the realizability game.

Output. The output of the execution indicates if the input formula ϕ is realizable, and in this case proposes at least one winning strategy for the system (Player O). At least one winning strategy for the environment (Player I) can also be returned when ϕ is unrealizable (only in case of a monolithic automaton construction). Those strategies are represented by infinite word automata. When they are small (≤ 20 states), those automata are also drawn in PNG and DOT using PyGraphviz. Many statistics about the execution are also output. A screenshot of the output of the web interface on a realizable specification is given in Figure 5.5. Words of explanation on how to interpret the output strategies are given in the web interface helper.

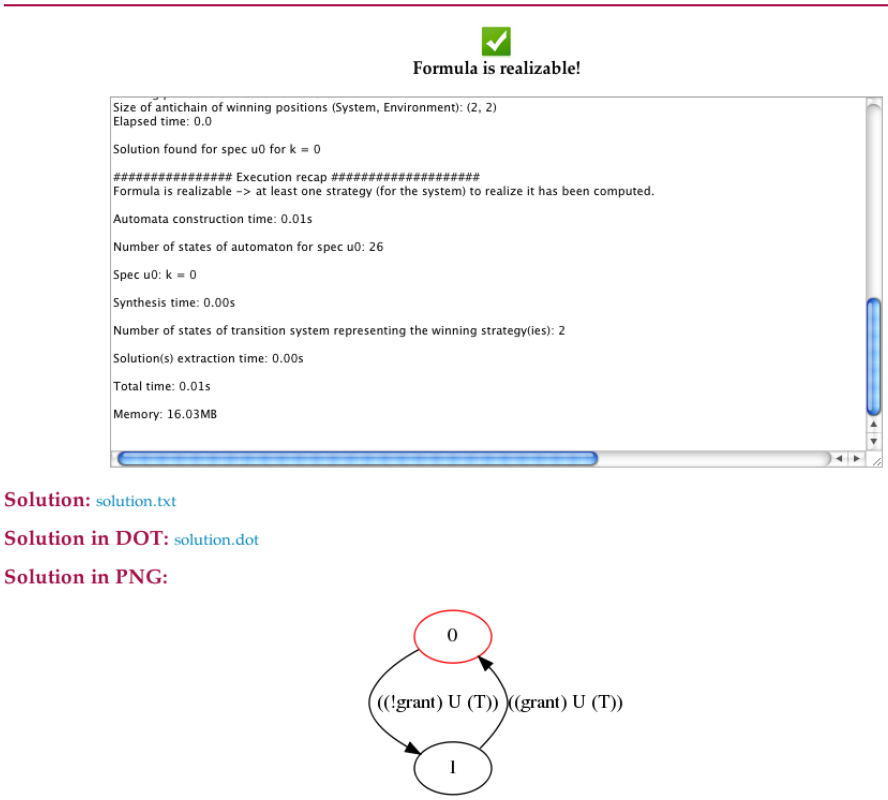

Figure 5.5: Web interface of Acacia+ (output).

5.7 Application scenarios and experiments

In this section, we describe three typical scenarios of usage of Acacia+ and we report experimental results. All experiments were carried out on a Linux platform with a 3.2GHz CPU (Intel Core i7) and 12GB of memory. The timeout (**TO**) is set to one hour. Note that Acacia+ is single-threaded and thus uses only one core. More details and examples can be found on the website.

5.7.1 System synthesis from LTL specifications

A first classical use of Acacia+ is to construct finite-state systems that enforce LTL specifications. Such specifications are usually specified by a set of LTL assumptions on the environment, and a set of LTL guarantees to be fulfilled by the system. Several benchmarks of synthesis problems are available for comparison with other tools: the test suite included in the tool Lily [JB06], a Generalized buffer controller from the IBM RuleBase tutorial [IBM], and the Load balancing system provided with the tool Unbeast [Ehl11].

We compare Acacia+ v2.2 with Lily 1.0.2 and Unbeast v.0.6 on those three benchmark. Note that for Unbeast, the environment always plays first, and thus it is weaker than when the system plays first. Some specifications are designed for the system playing first (Lily test suite and the Generalized buffer controller). To run them with Unbeast, one rewrote the specifications to make them environment starting compliant (see [FJR13]).

Lily test suite. We consider the accumulated execution times for the whole Lily test suite (Examples 1 to 23), as these examples are very small. Note that four examples of the benchmark are unrealizable (Examples 1, 2, 4 and 11). For the sake of fair comparison, we only check realizability on realizable examples and unrealizability on unrealizable ones, for all tools we benchmark. Synthesis is turned on for all tools, and contrarily to Lily and Unbeast, Acacia+ outputs a counter strategy for the environment when the formula is unrealizable. Table 5.1 shows that Acacia+ outperforms the two other tools on this benchmark. Regarding the tool used by Acacia+ for automaton construction, LTL3BA produces smaller automata with less non-deterministic choices than LTL2BA [BKRS12], which are thus more easily handled by Acacia+; Wring produces small automata

but is slower than other tools; finally, Spot offers mitigated execution times both for automaton construction and synthesis, although it is the tool that produces the smallest automata.

Table 5.1: Acacia+, Lily and Unbeast on Lily test suite. Execution times of Acacia+ are given with LTL2BA, LTL3BA, Wring and Spot for automaton construction. The column *AC* gives the time for automata construction, *syn* the time for synthesis, and *total* the total execution time. All times are given in seconds.

	AC	syn	total
Acacia+ (LTL2BA)	0.33	0.17	0.5
Acacia+ (LTL3BA)	0.36	0.15	0.51
Acacia+ (Wring)	28.33	0.1	28.43
Acacia+ (Spot)	1.46	0.32	1.78
Lily	65.23	16.05	81.28
Unbeast	?	?	15

Generalized buffer controller. Table 5.2 gives the performance of Acacia+ using LTL3BA on the Generalized buffer controller benchmark.[5] These results have been obtained with compositional automaton construction and compositional realizability check, with the forward algorithm for solving the underlying SGs. We obtain same solution sizes, and similar execution times and memory consumptions with monolithic realizability check. Unbeast (even with synthesis turned off), and Lily meet a timeout on all these examples. Let us note that Unbeast does not handle compositional specifications, which may explain its low performance on this benchmark. However, it is not due to the automaton construction, as LTL2BA (on which Unbeast is based) can construct the automaton monolithically in less than 1s, for the example with 2 clients. Note that the strategies synthesized by Acacia+ are relatively small compared to the number of states of the automata, although in theory they could be exponentially larger. Note also that the surprisingly large amount of memory consumption on small instances is due to Python libraries loaded in memory.

Load balancing system. Table 5.3 presents the execution results of Acacia+ and Unbeast on the Load balancing system benchmark. For the sake of fair com-

[5]Execution results of Acacia+ using the three other tools for automaton construction are available on the website.

Table 5.2: Acacia+ on the Generalized buffer controller parameterized by the number of receivers r. The column $\sum |\mathcal{A}_i|$ gives the sum of the number of states of each sub-automaton $\mathcal{A}_i$, $|\mathcal{M}|$ the size of the synthesized Moore machine, *mem* the total memory consumption, and all other columns have the same meaning as in Table 5.1. All memory consumptions are given in megabytes.

r	$\sum \lvert\mathcal{A}_i\rvert$	$\lvert\mathcal{M}\rvert$	AC	syn	total	mem
2	390	49	0.1	0.3	0.4	17.7
3	564	69	0.13	0.91	1.04	19.5
4	784	89	0.21	2.63	2.84	22.2
5	1070	121	0.3	7.22	7.52	25.0
6	1440	148	0.43	20.15	20.58	34.5
7	1912	167	0.81	53.81	54.62	52.8

parison, we used LTL2BA in Acacia+ to construct the automata, as Unbeast does. Moreover, we ran Acacia+ with the environment as starting player (as done with Unbeast). For the sake of completeness, we also ran Acacia+ with the system playing first. For Acacia+, those results have been obtained with compositional automaton construction and compositional realizability, with the forward algorithm for solving the SGs. Compared to Unbeast, the execution times seem similar but Acacia+ outputs much smaller strategies, unlike those output by Unbeast (they are represented by NuSMV programs for which we give the size of the files in which they are stored).

Table 5.3: Acacia+ and Unbeast on the Load balancing system parameterized by the number of clients c. The column |NuSMV| gives the size (in megabytes) of the files of the synthesized NuSMV programs by Unbeast, and all other columns have the same meaning as in Table 5.2.

.

	Acacia+								Unbeast	
	System				Environment					
c	$\sum \lvert\mathcal{A}_i\rvert$	$\lvert\mathcal{M}\rvert$	time	mem	$\sum \lvert\mathcal{A}_i\rvert$	$\lvert\mathcal{M}\rvert$	time	mem	|NuSMV|	time
2	122	11	0.08	16.62	153	11	0.09	16.61	0.6	0.2
3	268	25	0.38	18.11	231	45	0.24	17.67	5.2	2.3
4	538	59	2.78	29.26	302	155	2.70	35.24	30	10.0
5	802	272	197.12	404.92	373	645	193.96	367.07	151	254.4
6	890	285	375.84	658.61			**TO**			**TO**

As a conclusion, the overall performance of Acacia+ on these three bench-

marks is better than or similar to other tools, with the advantage of generating compact solutions. As mentioned in [Ehl11], extracting small strategies is a challenging problem.

5.7.2 Debugging of LTL specifications

Writing a correct LTL specification is error prone [KHB10]. Acacia+ can help to debug unrealizable LTL specifications as follows. As explained in Section 5.3.1, when an LTL specification ϕ is unrealizable for Player O, then its negation $\neg\phi$ is realizable for Player I. A winning strategy of Player I for $\neg\phi$ can then be used to debug the specification ϕ. Again, Acacia+ often offers the advantage to output readable compact (counter) strategies that help the specifier to correct his specification. This application scenario is illustrated in the following example.

Example 5.22. Assume that we want to synthesize a system that always grants a request, and that avoids undesirable grants, that is, it never does a grant when there is no pending request. Let $\phi_1 = \Box(r \Rightarrow \Diamond g) \wedge \neg\Diamond(\neg r \Rightarrow \mathsf{X}g)$ be a naive (and incorrect) specification for such a system, where $I = \{r\}$ and $O = \{g\}$. Applied to ϕ_1, Acacia+ concludes that ϕ_1 is unrealizable, and outputs a counter strategy for the environment. This strategy is depicted in Figure 5.6 (left side) as an infinite word automaton such that transitions are labeled with symbols $o|i$ with $o \in \Sigma_O$ and $i \in \Sigma_I$. The label $\mathsf{true}|i$ (resp. $o|\mathsf{true}$) on a transition from q to q' means that there is a transition labeled with $o|i$ from q to q' for all $o \in \Sigma_O$ (resp. for all $i \in \Sigma_I$). This strategy tells the environment to make requests all the time, making the sub-formula $\neg\Diamond(\neg r \Rightarrow \mathsf{X}g)$ always false, whatever the system does. This observation indicates that ϕ_1 was not correctly specified, and can be rewritten as $\phi_2 = \Box(r \Rightarrow \Diamond g) \wedge \neg\Diamond(\neg r \wedge \mathsf{X}g)$. This new specification is now realizable, and the winning strategy computed by Acacia+ is depicted in Figure 6.1 (right side). This automaton can be seen as a Moore machine $\mathcal{M} = (M, m_0, \alpha_U, \alpha_N)$ with $M = \{q_0, q_1\}$, $m_0 = q_0$, and for all transition from q to q' labeled by $o|i$, we have $\alpha_U(q, i) = q'$ and $\alpha_N(q) = o$. ◁

5.7.3 From LTL to deterministic automata

A third application scenario of Acacia+ consists in generating, from LTL formula φ, deterministic automata $\mathcal{A}_\varphi$ that recognize exactly the set $[\![\varphi]\!]$. We consider

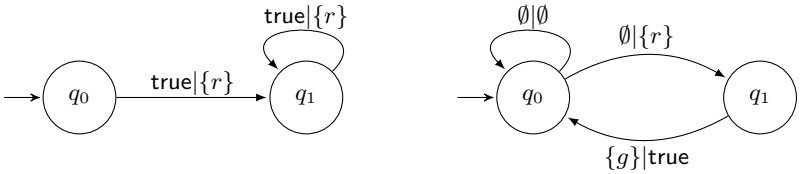

Figure 5.6: Counter strategy for the environment for ϕ_1 (left side) and winning strategy for the system for ϕ_2 (right side) computed by Acacia+.

the construction of deterministic Büchi automata (DBs) and deterministic parity automata (DPs).

From LTL to DB. As suggested to us by Rüdiger Ehlers, following an idea proposed in [KV05], LTL synthesis tools can be used to convert LTL formulas into equivalent DBs (when possible). The idea is as follows. Let φ be an LTL formula over a set of atomic propositions Σ and σ be a new proposition not in Σ. Let $I = \Sigma$ and $O = \{\sigma\}$. Then, the LTL formula $\phi = (\varphi \Leftrightarrow \Box\Diamond\sigma)$ is realizable if and only if there exists a DB $\mathcal{A}_\varphi$ such that $\mathcal{L}_\mathrm{b}(\mathcal{A}_\varphi) = [\![\varphi]\!]$. Indeed if ϕ is realizable, then the Moore machine $\mathcal{M} = (M, m_0, \alpha_U, \alpha_N)$ representing a winning strategy for Player O can be transformed into such a DB $\mathcal{A}_\varphi$ whose final states are exactly the states $m \in M$ with $\alpha_N(m) = \sigma$. Conversely, if there exists a DB $\mathcal{A}_\varphi$ such that $\mathcal{L}_\mathrm{b}(\mathcal{A}_\varphi) = [\![\varphi]\!]$, this automaton, outputting σ on final states, realizes ϕ.

Therefore one can use Acacia+ to construct DBs from LTL formulas (if possible). In [Ehl10], the author provides an automated method (together with a prototype) for the NP-complete problem of minimizing Büchi automata [Sch10]. This method is benchmarked on several automata obtained from a set of LTL formulas, for which the size of the minimal equivalent DB is given. We use those formulas to benchmark Acacia+ on the LTL to DB problem. Results are given in Table 5.4. We use LTL3BA to generate nondeterministic automata for formulas ϕ and the backward algorithm for solving the underlying SGs. We obtain very short execution times and the size of the constructed automata is very close to that of minimal DBs. The minimal size is indeed reached for 21 among 26 formulas. This shows again that Acacia+ is able to synthesize compact strategies. Note that there is no DB equivalent to the last formula $\varphi = \Diamond\Box\neg a$ (cf. Example 2.8 page 17), for which Acacia+ outputs that ϕ is unrealizable.

Table 5.4: Construction of equivalent DBs from LTL formulas φ using Acacia+ with LTL3BA. The column $|\mathcal{M}|$ gives the number of states of the produced DB equivalent to φ, $\min|\mathcal{A}_\varphi|$ the number of states of the minimal DB equivalent to φ, and *total* the total execution time (in seconds).

#	LTL formula φ	$\lvert\mathcal{M}\rvert$	$\min\lvert\mathcal{A}_\varphi\rvert$	total
1	$\Diamond(q \wedge \mathsf{X}(p\mathsf{U}r))$	**3**	3	0.02
2	$(p\mathsf{U}q\mathsf{U}r) \wedge (q\mathsf{U}r\mathsf{U}p) \wedge (r\mathsf{U}p\mathsf{U}q)$	12	3	0.04
3	$\Diamond(p \wedge \mathsf{X}(q \wedge \mathsf{X}\Diamond r))$	**4**	4	0.02
4	$\Box(p \Rightarrow (q\mathsf{U}r))$	**3**	3	0.02
5	$p\mathsf{U}(q \wedge \mathsf{X}(r\mathsf{U}s))$	**5**	5	0.04
6	$\Diamond(p \wedge \mathsf{X}\Diamond(q \wedge \mathsf{X}\Diamond(r \wedge \mathsf{X}\Diamond s)))$	**5**	5	0.05
7	$p\mathsf{U}(q \wedge \mathsf{X}(r \wedge \Diamond(s \wedge \mathsf{X}\Diamond(u \wedge \mathsf{X}\Diamond(v \wedge \mathsf{X}\Diamond w)))))$	**9**	9	1.81
8	$\Box\Diamond p \wedge \Box\Diamond q \wedge \Box\Diamond r \wedge \Box\Diamond s \wedge \Box\Diamond u$	**6**	6	0.11
9	$\Box\Diamond a \vee \Box\Diamond b \vee \Box\Diamond c$	**2**	2	0.03
10	$\Box(a \Rightarrow \Diamond b)$	**2**	2	0.02
11	$\Box(a\mathsf{U}b\mathsf{U}\neg a\mathsf{U}\neg b)$	6	2	0.08
12	$(\Box a \Rightarrow \Diamond b) \wedge (\Box\neg a \Rightarrow \Diamond\neg b)$	**4**	4	0.02
13	$\Box\neg c \wedge \Box(a \Rightarrow \Diamond b) \wedge \Box(b \Rightarrow \Diamond c)$	**2**	2	0.02
14	$\Box(a \Rightarrow \Diamond b) \wedge \Box c$	**3**	3	0.02
15	$\Box\Diamond(a \Rightarrow \mathsf{XXX}b)$	**2**	2	0.07
16	$\Box(a \Rightarrow \Diamond b) \wedge \Box(\neg a \Rightarrow \Diamond\neg b)$	**4**	4	0.05
17	$\Box\Diamond(a \Leftrightarrow \mathsf{XX}b)$	10	6	0.07
18	$\Box(a \Rightarrow \Diamond b) \wedge \Box(b \Rightarrow \Diamond c)$	**5**	5	0.06
19	$\Box(a \Rightarrow \mathsf{XXX}b)$	**9**	9	0.04
20	$\Box(a \Rightarrow \Diamond b) \wedge \Box(c \Rightarrow \Diamond d)$	7	6	0.09
21	$\Box\Diamond(a \Leftrightarrow \mathsf{XXX}b)$	22	?	0.34
22	$\Box a$	**2**	2	0.02
23	$\Box(a \Rightarrow \Box b)$	**3**	3	0.02
24	$\Diamond a$	**2**	2	0.01
25	$\Box(a \wedge (b \Rightarrow (b\mathsf{U}(c \wedge b))))$	**3**	3	0.02
26	$\Box(a \Rightarrow \Box(b \Rightarrow \Diamond c))$	**3**	3	0.03
27	$\Diamond\Box\neg a$	NA	NA	0.02

From LTL to DP. As shown in Table 5.4, there is no DB equivalent to the formula $\varphi = \Diamond\Box\neg a$. However, with a similar technique [KV05], we can construct a DP of fixed index. Indeed, let us consider the construction of a DP $\mathcal{A}_\varphi$ with two priorities 0 and 1 equivalent to φ. It suffices to synthesize a Moore machine $\mathcal{M} = (M, m_0, \alpha_U, \alpha_N)$ (if it exists) for the formula

$$\varphi \quad \Leftrightarrow \quad (\Box\Diamond p_0 \ \wedge \ \neg(\Box\Diamond p_1) \ \wedge \ \Box(p_0 \Leftrightarrow \neg p_1))$$

where $I = \{a\}$ and $O = \{p_0, p_1\}$. The last requirement ensures that priorities form a partition of M (a state in $\mathcal{A}_\varphi$ has priority i if and only if its corresponding

$m \in M$ is such that $\alpha_N(m) = p_i$, for all $i \in \{0, 1\}$). Table 5.5 gives the execution times of Acacia+ together with the size of the constructed DPs on a benchmark of LTL formulas from [KE12]. Note that the first formula of Table 5.5 is the last one of Table 5.4 for which Acacia+ outputs a DP (of index 3) of size 4 while the minimal one is of size 2. All examples are solved with the backward algorithm, except for the 19th example which is solved with the forward algorithm since it meets a timeout otherwise.

Table 5.5: Construction of equivalent DPs from LTL formulas φ using Acacia+ with LTL3BA. The column $|\mathcal{M}|$ gives the number of states of the produced DP $\mathcal{A}_\varphi$ equivalent to φ, *index* the index of $\mathcal{A}_\varphi$, and *total* the total execution time (in seconds).

#	LTL formula φ	$\lvert\mathcal{M}\rvert$	index	total
1	$\Diamond\Box\neg a$	4	3	0.06
2	$\Box(a \vee \Diamond b)$	2	2	0.03
3	$\Diamond\Box a \vee \Diamond\Box b \vee \Diamond\Box c$	5	3	0.35
4	$\Diamond(a \vee b)$	2	2	0.02
5	$\Box\Diamond(a \vee b)$	2	2	0.02
6	$\Box(a \vee b \vee c)$	2	2	0.02
7	$\Box(a \vee \Diamond(b \vee c))$	2	2	0.03
8	$\Diamond a \vee \Box b$	3	2	0.03
9	$\Box(a \vee \Diamond(b \wedge c))$	2	2	0.03
10	$\Diamond\Box a \vee \Box\Diamond b$	3	3	0.07
11	$\Box\Diamond(a \vee b) \wedge \Box\Diamond(b \vee c)$	3	2	0.04
12	$\Box\Diamond a \wedge \Diamond\Box b$	5	4	0.10
13	$(\Box\Diamond a \wedge \Diamond\Box b) \vee (\Diamond\Box\neg a \vee \neg b)$	6	3	0.59
14	$\Diamond\Box a \wedge \Box\Diamond a$	5	3	0.07
15	$\Box(\Diamond a \wedge \Diamond b)$	3	2	0.04
16	$\Diamond a \wedge \Diamond b$	4	2	0.02
17	$(\Box(b \vee \Box\Diamond a) \wedge \Box(c \vee \Box\Diamond\neg a)) \vee \Box b \vee \Box c$	6	2	0.22
18	$(\Box(b \vee \Diamond\Box a) \wedge \Box(c \vee \Diamond\Box\neg a)) \vee \Box b \vee \Box c$	4	2	0.07
19	$(\Diamond(b \wedge \Diamond\Box a) \vee \Diamond(c \wedge \Diamond\Box\neg a)) \wedge \Diamond b \wedge \Diamond c$	10	3	0.16
20	$(\Diamond(b \wedge \Box\Diamond a) \vee \Diamond(c \wedge \Box\Diamond\neg a)) \wedge \Diamond b \wedge \Diamond c$	4	2	0.04
21	$\Diamond\Box a \vee \Box\Diamond b$	3	3	0.11
22	$(\bigwedge_{i=1}^{5} \Box\Diamond a_i) \Rightarrow \Box\Diamond b$	8	3	5.45
23	$\Diamond\Box(\Diamond a \vee \Box\Diamond b \vee \Diamond\Box(a \vee b))$	2	2	0.04
24	$\Diamond\Box(\Diamond a \vee \Box\Diamond b \vee \Diamond\Box(a \vee b) \vee \Diamond\Box b)$	2	2	0.04

CHAPTER 6

Synthesis from LTL specifications with mean-payoff objectives

The LTL synthesis problem is purely qualitative: the given LTL specification is realized or not by a reactive system. LTL is not expressive enough to formalize the correctness of reactive systems with respect to some quantitative aspects. In this chapter, we extend the *qualitative* LTL synthesis setting to a *quantitative* setting. We study the problems of synthesis from *quantitative specifications* expressed in LTL extended with mean-payoff ($\mathsf{LTL}_{\mathsf{MP}}$) and energy ($\mathsf{LTL}_{\mathsf{E}}$) objectives.

We first define the $\mathsf{LTL}_{\mathsf{MP}}$ and $\mathsf{LTL}_{\mathsf{E}}$ realizability and synthesis problems, and we show that, as for the LTL realizability problem, both the $\mathsf{LTL}_{\mathsf{MP}}$ and the $\mathsf{LTL}_{\mathsf{E}}$ realizability problems are 2ExpTime-complete. We then propose a Safraless procedure to solve these problems based on a reduction to safety games, and we show how to efficiently solve those games with antichains by exploiting a partial order on their state space. We show that our results can be extended to multi-dimensional mean-payoff and energy objectives. Finally, we report some promising experimental results.

The content of this chapter is based on [BBFR13] and its extended version [BBFR12].

6.1 Definitions

In this section, we formally define the $\mathsf{LTL}_{\mathsf{MP}}$ and $\mathsf{LTL}_{\mathsf{E}}$ realizability and synthesis problems and related notions. We first consider the following example to illustrate our contributions.

Example 6.1. Let us consider the following specification of a system that should grant exclusive access to a resource to two clients. A client requests access to the resource by setting to true its request signal (r_1 for client 1 and r_2 for client 2), and the server grants those requests by setting to true the respective grant signal g_1 or g_2. We want to synthesize a server that eventually grants any client request, and that only grants one request at a time. This can be formalized with the following LTL specification ϕ where the signals in $I = \{r_1, r_2\}$ are controlled by the environment (the two clients), and the signals in $O = \{g_1, w_1, g_2, w_2\}$ are controlled by the server.

$$\begin{aligned}\phi_1 &= \square(r_1 \Rightarrow \mathsf{X}(w_1 \mathsf{U} g_1))\\ \phi_2 &= \square(r_2 \Rightarrow \mathsf{X}(w_2 \mathsf{U} g_2))\\ \phi_3 &= \square(\neg g_1 \vee \neg g_2)\\ \phi &= \phi_1 \wedge \phi_2 \wedge \phi_3\end{aligned}$$

Intuitively, ϕ_1 (resp. ϕ_2) specifies that any request of client 1 (resp. client 2) must be eventually granted, and in-between the waiting signal w_1 (resp. w_2) must be high. Formula ϕ_3 stands for mutual exclusion.

Applied on ϕ, Acacia+ concludes that ϕ is realizable, and outputs the winning strategy depicted in Figure 6.1. This strategy tells the server to alternatively assert $\{g_1, w_2\}$ and $\{g_2, w_1\}$, i.e. alternatively grant client 1 and client 2. While this strategy is formally correct, as it realizes the formula ϕ against all possible behaviors of the clients, it may not be the one that we expect. Indeed, we may prefer a solution that does not make unsolicited grants for example. Or, we may prefer a solution that gives, in case of request by both clients, some priority to client 2's request. In the later case, one elegant solution would be to associate a cost equal to 2 when w_2 is true and a cost equal to 1 when w_1 is true. This clearly will favor solutions that give priority to requests from client 2 over requests from client 1. We will develop several other examples in this chapter and describe the solutions that we obtain automatically with our algorithms. ◁

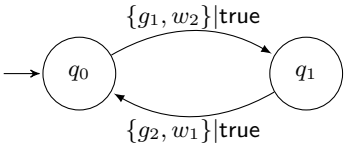

Figure 6.1: Winning strategy for the system computed by Acacia+ for the specification of Example 6.1.

$\mathsf{LTL}_{\mathsf{MP}}$ realizability and synthesis problems. Consider a finite set P partitioned as $I \uplus O$. Let $\mathsf{Lit}(P)$ be the set $\{p \mid p \in P\} \cup \{\neg p \mid p \in P\}$ of literals over P, and let $w : \mathsf{Lit}(P) \to \mathbb{Z}$ be a *weight function* where positive numbers represent rewards. This function is extended to Σ_I (resp. Σ_O) as follows: $w(i) = \Sigma_{p \in i} w(p) + \Sigma_{p \in I \setminus \{i\}} w(\neg p)$ for $i \in \Sigma_I$ (resp. $w(o) = \Sigma_{p \in o} w(p) + \Sigma_{p \in O \setminus \{o\}} w(\neg p)$ for $o \in \Sigma_O$). In this way, it can also be extended to Σ_P as $w(o \cup i) = w(o) + w(i)$ for all $o \in \Sigma_O$ and $i \in \Sigma_I$.[1] In the sequel, we denote by $\langle P, w \rangle$ the pair given by the finite set P and the weight function w over $\mathsf{Lit}(P)$; we also use the weighted alphabet $\langle \Sigma_P, w \rangle$.

Recall from the previous chapter that the LTL realizability problem is best seen as a game between Player I and Player O. The game is played similarly in the case of $\mathsf{LTL}_{\mathsf{MP}}$. Consider an LTL formula ϕ over $\langle P, w \rangle$ and an outcome $u = (o_0 \cup i_0)(o_1 \cup i_1) \cdots \in \Sigma_P^\omega$ produced by Players I and O. We associate a *value* $\mathsf{Val}(u)$ with u that captures the two objectives of Player O of both satisfying ϕ and achieving a mean-payoff objective. We define the *energy level* of the prefix $u(n)$ as $\mathsf{EL}(u(n)) = \sum_{k=0}^{n-1} w(o_k) + w(i_k)$. We then assign to u a *mean-payoff value* equal to $\mathsf{MP}(u) = \liminf_{n \to \infty} \frac{1}{n} \mathsf{EL}(u(n))$. Finally we define the value of u as:

$$\mathsf{Val}(u) = \begin{cases} \mathsf{MP}(u) \text{ if } u \models \phi \\ -\infty \text{ otherwise.} \end{cases}$$

Given an LTL formula ϕ over $\langle P, w \rangle$, a partition of P into I and O, and a threshold $\nu \in \mathbb{Q}$, the $\mathsf{LTL}_{\mathsf{MP}}$ *realizability problem* (resp. $\mathsf{LTL}_{\mathsf{MP}}$ *realizability problem under finite memory*) asks to decide whether there exists a strategy (resp.

[1] The decomposition of $w(o \cup i)$ as the sum $w(o) + w(i)$ emphasizes the partition of P as $I \uplus O$ and will be useful in some proofs.

a finite-memory strategy) λ_O of Player O such that $\mathsf{Val}(\mathsf{Outcome}(\lambda_O, \lambda_I)) \geq \nu$ against all strategies λ_I of Player I. If such a *winning* strategy exists, we say that ϕ is MP*-realizable* (resp. MP*-realizable under finite memory*). The $\mathsf{LTL}_{\mathsf{MP}}$ *synthesis problem* is to produce such a winning strategy λ_O for Player O. Therefore the aim is to achieve two objectives: *(i)* realizing ϕ, and *(ii)* having a long-run average reward greater than or equal to the given threshold.

Optimality. Let ϕ be an LTL formula over $\langle P, w \rangle$. The *optimal value* (for Player O) is defined as:

$$\nu_\phi = \sup_{\lambda_O \in \Pi_O} \inf_{\lambda_I \in \Pi_I} \mathsf{Val}(\mathsf{Outcome}(\lambda_O, \lambda_I)).$$

For a real-valued $\epsilon \geq 0$, a strategy λ_O of Player O is called ϵ*-optimal* if $\mathsf{Val}(\mathsf{Outcome}(\lambda_O, \lambda_I)) \geq \nu_\phi - \epsilon$ against all strategies λ_I of Player I. It is *optimal* if it is ϵ-optimal with $\epsilon = 0$. Notice that ν_ϕ is equal to $-\infty$ if Player O cannot realize ϕ

Example 6.2. Let us come back to Example 6.1 of a client-server system with two clients sharing a resource. The specification has been formalized by an LTL formula ϕ over the alphabet $P = I \uplus O$, with $I = \{r_1, r_2\}$, $O = \{g_1, w_1, g_2, w_2\}$. Suppose that we want to add the following constraints: client 2's requests take the priority over client 1's requests, but client 1's should still be eventually granted. Moreover, we would like to keep minimal the delay between requests and grants. This latter requirement has more the flavor of an optimality criterion and is best modeled using a weight function and a mean-payoff objective. To this end, we impose penalties to the waiting signals w_1 and w_2 controlled by Player O, with a larger penalty to w_2 than to w_1. For instance, we use the following weight function $w : \mathsf{Lit}(P) \to \mathbb{Z}$:

$$w(l) = \begin{cases} -1 \text{ if } l = w_1 \\ -2 \text{ if } l = w_2 \\ 0 \text{ otherwise.} \end{cases}$$

One optimal strategy for the server is to behave as follows: it almost always grants the resource to client 2 immediately after r_2 is set to true by client 2, and with a decreasing frequency grants requests r_1 emitted by client 1. Such a server

ensures a mean-payoff value equal to -1 against the most demanding behavior of the clients (where they are constantly requesting the shared resource). Such a strategy requires the server to use an infinite memory as it has to grant client 1 with an infinitely decreasing frequency. Note that a server that would grant client 1 in such a way without the presence of requests by client 1 would still be optimal.

It is easy to see that no finite memory server can be optimal. Indeed, if we allow the server to count only up to a fixed positive integer $k \in \mathbb{N}$ then the best that this server can do is as follows: grant immediately any request by client 2 if the last ungranted request of client 1 has been emitted less than k steps in the past, otherwise grant the request of client 1. The mean-payoff value of this solution, in the worst-case (when the two clients always emit their respective request), is equal to $-(1 + \frac{1}{k})$. So, even if finite memory cannot be optimal in this example, it is the case that given any $\epsilon > 0$, we can devise a finite-memory strategy that is ϵ-optimal. $\lhd$

LTL_E realizability and synthesis. For our proofs, we also need to consider realizability and synthesis with *energy objectives* (instead of mean-payoff objectives). With the same notations as before, the LTL_E *realizability problem* is to decide whether ϕ is E-*realizable*, that is, whether there exists a strategy λ_O of Player O and an integer $c_0 \in \mathbb{N}$ such that for all strategies λ_I of Player I, *(i)* $u = \mathsf{Outcome}(\lambda_O, \lambda_I) \models \phi$, and *(ii)* $\forall n \geq 0, c_0 + \mathsf{EL}(u(n)) \geq 0$. Instead of requiring that $\mathsf{MP}(u) \geq \nu$ for a given threshold ν as second objective, we thus ask if there exists an *initial credit* $c_0 \in \mathbb{N}$ such that the energy level of each prefix $u(n)$ remains positive. When ϕ is E-realizable, the LTL_E *synthesis problem* is to produce such a winning strategy λ_O for Player O. Finally, we define the *minimum initial credit* as the least value of initial credit for which ϕ is E-realizable. A strategy λ_O is *optimal* if it is winning for the minimum initial credit.

6.2 Computational complexity

In this section, we solve the LTL_MP realizability problem, and we establish its complexity. Our solution relies on a reduction to a mean-payoff parity game (MPPG). The same result also holds for the LTL_E realizability problem.

6.2.1 Solution to the $\mathsf{LTL}_{\mathsf{MP}}$ realizability problem

We first consider the $\mathsf{LTL}_{\mathsf{MP}}$ realizability problem. We have the next theorem.

Theorem 6.3. *The $\mathsf{LTL}_{\mathsf{MP}}$ realizability problem is 2ExpTime-complete.*

Our proof of Theorem 6.3 is based on the following proposition.

Proposition 6.4. *Let ϕ be an LTL formula over $\langle P, w_P\rangle$ such that $|\phi| = n$. Then one can construct a MPPG $\langle \mathcal{G}_\phi, w, p\rangle$ with $2^{2^{\mathcal{O}(n \log n)}}$ states and $2^{\mathcal{O}(n)}$ priorities such that the following statements are equivalent. For each threshold $\nu \in \mathbb{Q}$:*

1. *there exists a (finite-memory) strategy λ_O of Player O such that, against all strategies λ_I of Player I, $\mathsf{Val}(\mathsf{Outcome}(\lambda_O, \lambda_I)) \geq \nu$;*
2. *there exists a (finite-memory) strategy λ_1 of Player 1 such that, against all strategies λ_2 of Player 2, $\mathsf{Val}_{\mathcal{G}_\phi}(\mathsf{Outcome}_{\mathcal{G}_\phi}(\lambda_1, \lambda_2)) \geq \nu$ in the game $\langle \mathcal{G}_\phi, w, p\rangle$.*

Moreover, if λ_1 is a finite-memory strategy with size m, then λ_O can be chosen as a finite-memory strategy with size $m \cdot |S_1|$ where S_1 is the set of states of Player 1 in $\mathcal{G}_\phi$.[2]

Proof. Let ϕ be an LTL formula over $P = I \uplus O$ such that $|\phi| = n$, and let $w_P : \mathsf{Lit}(P) \to \mathbb{Z}$ be a weight function. We first construct a deterministic parity automaton $\mathcal{A}_\phi = (\Sigma_P, Q, q_0, p, \delta)$ such that $\mathcal{L}_{\mathrm{p}}(\mathcal{A}_\phi) = [\![\phi]\!]$ (see Theorem 2.10). This automaton has $2^{2^{\mathcal{O}(n \log n)}}$ states and $2^{\mathcal{O}(n)}$ priorities.

We then derive from $\mathcal{A}_\phi$ a turn-based MPPG $\langle \mathcal{G}_\phi, w, p'\rangle$ with $\mathcal{G}_\phi = (S, s_0, E)$ as follows. The initial state s_0 is equal to (q_0, j_0) for some arbitrary $j_0 \in I$. To get the turn-based aspect, the set S is partitioned as $S_1 \uplus S_2$ such that $S_1 = \{(q, i) \mid q \in Q, i \in \Sigma_I\}$ and $S_2 = \{(q, o) \mid q \in Q, o \in \Sigma_O\}$.[3] Let us describe the edges of $\mathcal{G}_\phi$. For each $q \in Q$, $o \in \Sigma_O$ and $i \in \Sigma_I$, let $q' = \delta(q, o \cup i)$. Then, E contains the two edges $((q, j), (q, o))$ and $((q, o), (q', i))$ for all $j \in \Sigma_I$. We clearly have $E \subseteq (S_1 \times S_2) \cup (S_2 \times S_1)$, that is, $\mathcal{G}$ is turn-based. Moreover, since $\mathcal{A}_\phi$ is deterministic, we have the nice property that there exists a bijection $\Theta : \Sigma_P^* \to \mathsf{Pref}(\mathcal{G}_\phi) \cap (S_1 S_2)^* S_1$ defined as follows. For each $u = (o_0 \cup i_0)(o_1 \cup i_1) \dots (o_n \cup i_n) \in \Sigma_P^*$, we consider in $\mathcal{A}_\phi$ the run $\rho(u) =$

[2] A converse of this corollary could also be stated but is of no utility in the next results.

[3] We will limit the set of states to the accessible ones.

$q_0 q_1 \dots q_{n+1} \in Q^*$ such that $q_{k+1} = \delta(q_k, o_k \cup i_k)$ for all k. We then define $\Theta(u)$ as $(q_0, j_0)(q_0, o_0)(q_1, i_0)(q_1, o_1)(q_2, i_1) \cdots (q_n, o_n)(q_{n+1}, i_n)$. Clearly, Θ is a bijection and it can be extended to a bijection $\Theta : \Sigma_P^\omega \to \mathsf{Plays}(\mathcal{G}_\phi)$.

The priority function $p' : S \to \mathbb{N}$ for $\mathcal{G}_\phi$ is defined from the priority function p of $\mathcal{A}_\phi$ by $p'(q, o) = p'(q, i) = p(q)$ for all $q \in Q, o \in O$ and $i \in I$. The weight function $w : E \to \mathbb{Z}$ for $\mathcal{G}_\phi$ is defined from the weight function w_P as follows. For all edges e ending in a state (q, o) with $o \in O$ (resp. (q, i) with $i \in I$), we define $w(e) = w_P(o)$ (resp. $w(e) = w_P(i)$). Notice that Θ preserves the energy level, the mean-payoff value and the parity objective, since for each $u \in \Sigma_P^\omega$, we have *(i)* $\mathsf{EL}(u(n)) = \mathsf{EL}_{\mathcal{G}_\phi}(\Theta(u(n)))$ for all n[4], *(ii)* $\mathsf{MP}(u) = \mathsf{MP}_{\mathcal{G}_\phi}(\Theta(u))$, and *(iii)* $u \models \phi$ if and only if $\Theta(u)$ satisfies the objective $\mathsf{Parity}_{\mathcal{G}_\phi}(p')$.

It is now easy to prove that the two statements of Proposition 6.4 are equivalent. Suppose that *1.* holds. Given the winning strategy $\lambda_O : (\Sigma_O \Sigma_I)^* \to \Sigma_O$, we are going to define a winning strategy $\lambda_1 : (S_1 S_2)^* S_1 \to S_2$ of Player 1 in $\mathcal{G}_\phi$, with the idea that λ_1 mimics λ_O thanks to the bijection Θ. More precisely, for any prefix $\gamma \in (S_1 S_2)^* S_1$ compatible with λ_1, we let $\lambda_1(\gamma) = (q, o)$ such that (q, i_n) is the last state of γ and $o = \lambda_O(o_0 i_0 o_1 i_1 \dots o_n i_n)$ with $(o_0 \cup i_0)(o_1 \cup i_1) \dots (o_n \cup i_n) = \Theta^{-1}(\gamma)$. In this way, λ_1 is winning. Indeed, $\mathsf{Val}_{\mathcal{G}_\phi}(\mathsf{Outcome}_{\mathcal{G}_\phi}(\lambda_1, \lambda_2)) \geq \nu$ for any strategy λ_2 of Player 2 because λ_O is winning and by *(ii)* and *(iii)* above. Moreover λ_O is finite-memory if and only if λ_1 is finite-memory.

Suppose now that *2.* holds. Given the (finite-memory) winning strategy λ_1, we define a (finite-memory) winning strategy λ_O with the same kind of arguments as done above. We just give the definition of λ_O from λ_1. We let $\lambda_O(o_0 i_0 o_1 i_1 \dots o_n i_n) = o$ such that $\lambda_1(\gamma) = (q, o)$ with $\gamma = \Theta((o_0 \cup i_0)(o_1 \cup i_1) \dots (o_n \cup i_n))$.

Suppose now that λ_1 is finite-memory with size m. Let $\sim_1$ be a right-congruence on $\mathsf{Pref}(\mathcal{G}_\phi)$ of index m such that $\lambda_1(\gamma \cdot s) = \lambda_1(\gamma' \cdot s)$ for all $\gamma \sim_1 \gamma'$ and $s \in S_1$. To show that λ_O is finite-memory, we have to define a right-congruence $\sim_O$ on $(\Sigma_O \Sigma_I)^*$ of finite index such that $\lambda_O(u) = \lambda_O(u')$ for all $u \sim_O u'$. Let $u = o_0 i_0 \dots o_n i_n, u' = o'_0 i'_0 \dots o'_l i'_l \in (\Sigma_O \Sigma_I)^*$, let $\Theta((o_0 \cup i_0) \dots (o_n \cup i_n)) = \gamma \cdot s$, $\Theta((o'_0 \cup i'_0) \dots (o'_l \cup i'_l)) = \gamma' \cdot s'$. Looking at the definition of λ_O, we see that $\sim_O$ has to be defined such that $u \sim_O u'$ if $\gamma \sim_1 \gamma'$ and $s = s'$. Moreover,

[4]For this equality, it is useful to recall footnote 1.

$\sim_O$ is of index $m \cdot |S_1|$. This completes the proof.[5] □

We can now prove Theorem 6.3.

Proof of Theorem 6.3. By Proposition 6.4, solving the $\mathsf{LTL}_{\mathsf{MP}}$ realizability problem is equivalent to checking whether Player 1 has a winning strategy in the MPPG $\langle \mathcal{G}_\phi = (S, s_0, E), w, p \rangle$ for the given threshold ν. By Theorem 2.17, this check can be done in time $\mathcal{O}(|E| \cdot |S|^{d+2} \cdot W)$. Since $\mathcal{G}_\phi$ has $2^{2^{\mathcal{O}(n \log n)}}$ states and $2^{\mathcal{O}(n)}$ priorities where $n = |\phi|$ (see Proposition 6.4), the $\mathsf{LTL}_{\mathsf{MP}}$ realizability problem is in $\mathcal{O}(2^{2^{\mathcal{O}(n \log n)}} \cdot W)$.

This proves the 2Exptime-easiness of $\mathsf{LTL}_{\mathsf{MP}}$ realizability problem. The 2Exptime-hardness of this problem is a consequence of the 2Exptime-hardness of LTL realizability problem (see Theorem 5.2). □

Proposition 6.4 and its proof lead to the next two interesting corollaries. The first corollary is immediate. The second one asserts that the optimal value can be approached with finite-memory strategies. Recall from Section 2.4 that given a MPPG $\langle \mathcal{G}, w, p \rangle$, $\nu_{\mathcal{G}}$ denotes the optimal value for Player 1 in $\mathcal{G}$.

Corollary 6.5. *Let ϕ be an LTL formula and $\langle \mathcal{G}_\phi, w, p \rangle$ be the associated MPPG. Then, $\nu_\phi = \nu_{\mathcal{G}_\phi}$. Moreover, when $\nu_\phi \neq -\infty$, one can construct an optimal strategy λ_1 for Player 1 from an optimal strategy λ_O for Player O, and conversely.* □

Corollary 6.6. *Let ϕ be an LTL formula. If ϕ is MP-realizable, then for all $\epsilon > 0$, Player O has an ϵ-optimal winning strategy that is finite-memory, that is,*

$$\nu_\phi = \sup_{\substack{\lambda_O \in \Pi_O \\ \lambda_O \text{ finite-memory}}} \inf_{\lambda_I \in \Pi_I} \mathsf{Val}(\mathsf{Outcome}(\lambda_O, \lambda_I)).$$

Proof. Suppose that ϕ is MP-realizable. By Corollary 6.5, $\nu_\phi = \nu_{\mathcal{G}_\phi} \neq -\infty$. Therefore, by Proposition 6.4 and Theorem 2.17, for each $\epsilon > 0$, Player 1 has a finite-memory winning strategy λ_1 in the MPPG $\langle \mathcal{G}_\phi, w, p \rangle$ for the threshold $\nu_\phi - \epsilon$. By Proposition 6.4, from λ_1, we can derive a finite-memory winning strategy λ_O for the $\mathsf{LTL}_{\mathsf{MP}}$ realizability of ϕ for this threshold, which is thus the required finite-memory ϵ-optimal winning strategy. □

[5] Notice that we could have defined a smaller game graph $\mathcal{G}_\phi$ with $S_1 = Q$ and $w((q, o), q') = \max\{w_P(i) \mid \delta(q, o \cup i) = q'\}$ for all $(q, o) \in S_2, q' \in S_1$. However, the current definition simplifies the proof.

6.2.2 Solution to the LTL_E realizability problem

The same kind of arguments can be used to show that the LTL_E realizability problem is 2Exptime-complete. Indeed, in Proposition 6.4, it is enough to use a reduction with the same game $\langle \mathcal{G}_\phi, w, p \rangle$, but with energy parity objective instead of mean-payoff parity objective, that is, $\langle \mathcal{G}_\phi, w, p \rangle$ is an energy parity game (EPG). The proof is almost identical by taking the same initial credit c_0 for both the LTL_E realizability of ϕ and the energy objective in $\mathcal{G}_\phi$, and by using Theorem 2.15 instead of Theorem 2.17.

Theorem 6.7. *The LTL_E realizability problem is 2ExpTime-complete.* □

The next proposition states that when ϕ is E-realizable, then Player O has a finite-memory strategy the size of which is related to $\mathcal{G}_\phi$. This result is stronger than Corollary 6.6 since it also holds for optimal strategies and it gives a bound on the memory size of the winning strategy.

Proposition 6.8. *Let ϕ be an LTL formula and $\langle \mathcal{G}_\phi = (S, s_0, E), w, p \rangle$ be the associated EPG. Then, ϕ is E-realizable if and only if it is E-realizable under finite memory. Moreover Player O has a finite-memory winning strategy with a memory size bounded by $4 \cdot |S|^2 \cdot d \cdot W$, where $d = |\mathsf{Img}(p)|$.*

Proof. By Proposition 6.4 where $\langle \mathcal{G}_\phi = (S, s_0, E), w, p \rangle$ is considered as an EPG, we know that Player 1 has a winning strategy λ_1 for the initial credit problem in $\mathcal{G}_\phi$. Moreover, by Theorem 2.15, we can suppose that this strategy has finite-memory M with $|M| \leq 4 \cdot |S| \cdot d \cdot W$. Finally, one can derive a finite-memory winning strategy λ_O for the LTL_E realizability of ϕ with a memory size bounded by $|M| \cdot |S|$ by Proposition 6.4. □

6.3 Safraless algorithm

In the previous section, we have proposed an algorithm for solving the LTL_MP realizability of a given LTL formula ϕ, which is based on a reduction to the MPPG $\langle \mathcal{G}_\phi, w, p \rangle$. This algorithm has two main drawbacks. First, it requires the use of Safra's construction to get a deterministic automaton $\mathcal{A}_\phi$ such that $\mathcal{L}(\mathcal{A}_\phi) = [\![\phi]\!]$. Second, strategies for the game $\langle \mathcal{G}_\phi, w, p \rangle$ may require infinite memory (for the threshold $\nu_{\mathcal{G}_\phi}$, see Theorem 2.17). This can also be the case for

the $\mathsf{LTL}_{\mathsf{MP}}$ realizability problem, as illustrated by Example 6.2. In this section, we show how to circumvent these two drawbacks.

6.3.1 Finite-memory strategies

The second drawback (infinite memory strategies) has been already partially solved by Corollary 6.6, when the threshold given for the $\mathsf{LTL}_{\mathsf{MP}}$ realizability is the optimal value ν_ϕ. Indeed it states the existence of finite-memory winning strategies for the thresholds $\nu_\phi - \epsilon$, for all $\epsilon > 0$. We here show that we can go further by translating the $\mathsf{LTL}_{\mathsf{MP}}$ realizability problem under finite memory into an $\mathsf{LTL}_{\mathsf{E}}$ realizability problem, and conversely.

Recall (see Proposition 6.4) that testing whether an LTL formula ϕ is MP-realizable for a given threshold ν is equivalent to solve the MPPG $\langle \mathcal{G}_\phi, w, p\rangle$ for the same threshold. Moreover, with the same game graph, testing whether ϕ is E-realizable is equivalent to solve the EPG $\langle \mathcal{G}_\phi, w', p\rangle$. Proposition 2.22, which establishes a tight relationship between finite-memory strategies in MPPGs and EPGs, leads to the next theorem.

Theorem 6.9. *Let ϕ be an* LTL *formula ϕ over the weighted alphabet $\langle P, w_P\rangle$, and let $\langle \mathcal{G}_\phi = (S, s_0, E), w, p\rangle$ be its associated* MPPG*. Then,*

- *the formula ϕ is* MP*-realizable under finite-memory for threshold ν if and only if ϕ over $\langle P, w_P - \nu\rangle$ is* E*-realizable;*
- *if ϕ is* MP*-realizable under finite memory, Player O has a winning strategy whose memory size is bounded by $4 \cdot |S|^2 \cdot d \cdot W$, where $d = |\mathsf{Img}(p)|$ and W is the largest absolute weight of the weight function $w - \nu$.*

Proof. If ϕ is MP-realizable under finite-memory for threshold ν, then, by Proposition 6.4, Player 1 has a finite-memory strategy λ_1 in the MPPG $\langle \mathcal{G}_\phi, w, p\rangle$. By Proposition 2.22, λ_1 is a winning strategy in the EPG $\langle \mathcal{G}_\phi, w - \nu, p\rangle$, what shows (by Proposition 6.4) that ϕ is E-realizable with weight function $w_P - \nu$ over $\mathsf{Lit}(P)$. The converse is proved similarly.

Finally, given a winning strategy for the $\mathsf{LTL}_{\mathsf{E}}$ realizability of ϕ with weight function $w - \nu$ over $\mathsf{Lit}(P)$, we can suppose by Proposition 6.8 that it is finite-memory with a memory size bounded by $4 \cdot |S|^2 \cdot d \cdot W$, where S, d and W are the parameters of the EPG $\langle \mathcal{G}_\phi, w - \nu, p\rangle$. This concludes the proof. □

The next corollary follows from Theorem 6.9 and Corollary 6.6.

Corollary 6.10. *Let ϕ be an* LTL *formula over $\langle P, w_P \rangle$. Then for all $\epsilon \in \mathbb{Q}$, with $\epsilon > 0$, the following are equivalent:*

1. *λ_O is a finite-memory ϵ-optimal winning strategy for the* $\mathsf{LTL}_{\mathsf{MP}}$ *realizability of ϕ;*
2. *λ_O is a winning strategy for the* $\mathsf{LTL}_{\mathsf{E}}$ *realizability of ϕ with weight function $w_P - \nu_\phi + \epsilon$ over* $\mathsf{Lit}(P)$. □

It is important to notice that in this corollary, the memory size of the strategy (as described in Theorem 6.9) increases as ϵ decreases. Indeed, it depends on the weight function $w - \nu_\phi + \epsilon$ used by the EPG $\langle \mathcal{G}_\phi, w - \nu_\phi + \epsilon, p \rangle$. We recall that if $\epsilon = \frac{a}{b}$, then this weight function must be multiplied by b in a way to have integer weights (see footnote 3 page 19). The largest absolute weight W is thus also multiplied by b.

In the sequel, to avoid strategies with infinite memory when solving the $\mathsf{LTL}_{\mathsf{MP}}$ realizability problem, we will restrict to the $\mathsf{LTL}_{\mathsf{MP}}$ realizability problem under finite memory. By Theorem 6.9, it is enough to study winning strategies for the $\mathsf{LTL}_{\mathsf{E}}$ realizability problem (having in mind that the weight function over $\mathsf{Lit}(P)$ has to be adapted). In the sequel, we only study this problem.

6.3.2 Safraless construction

To avoid the Safra's construction needed to obtain a deterministic automaton for the underlying LTL formula, we adapt the Safraless construction presented in Section 5.2 for the LTL synthesis problem, in a way to deal with weights and efficiently solve the $\mathsf{LTL}_{\mathsf{E}}$ synthesis problem. Instead of constructing a MPPG from a deterministic parity automaton (DP) as done in Proposition 6.4, we will propose a reduction to a safety game (SG). To this end, we will need to define the notion of energy automaton. We first consider energy safety games (ESGs) and we show how they can be reduced to SGs. This reduction will be useful in our final proof.

Energy safety games. For an ESG $\langle \mathcal{G}, w, \alpha \rangle$, the *initial credit problem* asks whether there exists an initial credit $c_0 \in \mathbb{N}$ and a winning strategy for Player 1 for the objective $\mathsf{Safety}_{\mathcal{G}}(\alpha) \cap \mathsf{PosEnergy}_{\mathcal{G}}(c_0)$. Recall from Section 2.4.2 the

fixpoint algorithm for SGs that computes the set $\mathsf{Win}_1(\alpha)$ of states from which Player 1 has a winning strategy for the safety objective. We have the next reduction of ESGs to EGs.

Proposition 6.11. *Let $\langle \mathcal{G}, \alpha \rangle$ be an ESG and $\langle \mathcal{G}', w' \rangle$ be the EG such that $\mathcal{G}'$ is the subgraph of $\mathcal{G}$ induced by $\mathsf{Win}_1(\alpha)$ and w' is the restriction of w to its edges. Then the winning strategies of Player 1 in $\langle \mathcal{G}', w' \rangle$ are the winning strategies in $\langle \mathcal{G}, w \rangle$ that are restricted to the states of $\mathsf{Win}_1(\alpha)$.*

Proof. The graph $\mathcal{G}'$ is simply the subgraph of $\mathcal{G}$ induced by $\mathsf{Win}_1(\alpha)$. Its weight function w' is equal to w restricted to the edges of $\mathcal{G}'$. The winning strategies are identical for both graphs, except that they are restricted to the states of $\mathcal{G}'$ for the game $\mathcal{G}'$. □

The next corollary follows from Theorem 2.14 and Propositions 6.11.

Corollary 6.12. *The initial credit problem for a given ESG $\langle \mathcal{G} = (S, s_0, E), w, \alpha \rangle$ can be solved in $\mathsf{NP} \cap \mathsf{coNP}$. Moreover, if Player 1 has a winning strategy in $\langle \mathcal{G}, w, \alpha \rangle$, then he can win with a memoryless strategy and with an initial credit of $C = (|S| - 1) \cdot W$.*

We now show how EGs can be reduced to SGs. Indeed, by storing in the states of the game the current energy level up to some bound $C \geq 0$, one gets a SG with safety objective α corresponding to the set of states with a positive energy level. For a sufficiently large bound C, this SG is equivalent to the initial EG [BCD+11]. Intuitively, given an EG $\langle \mathcal{G}, w \rangle$, the states of the equivalent SG are pairs (s, c) where s is a state of $\mathcal{G}$ and c is an energy level in $\mathcal{C} = \{\bot, 0, 1, \ldots, C\}$ (with $\bot < 0$). When adding a positive (resp. negative) weight to an energy level, we bound the sum to C (resp. $\bot = -1$). The safety objective is to avoid states of the form $(s, \bot)$. Formally, given an EG $\langle \mathcal{G} = (S, s_0, E), w \rangle$, we define the SG $\langle \mathcal{G}_C = (S', s'_0, E'), \alpha \rangle$ as follows:

- $S' = \{(s, c) \mid s \in S, c \in \mathcal{C}\}$,
- $s'_0 = (s_0, C)$,
- $((s, c), (s', c')) \in E'$ if $e = (s, s') \in E$ and $c \oplus w(e) = c'$, and
- $\alpha = \{(s, c) \mid s \in S, c \neq \bot\}$

In this definition, we use the operator $\oplus : \mathcal{C} \times \mathbb{Z} \to \mathcal{C}$ such that $c \oplus k = \min(C, c+k)$ if $\{c \neq \bot \text{ and } c + k \geq 0\}$, and $\bot$ otherwise.

Example 6.13. Let us consider the following example to illustrate the reduction from EGs to SGs. Figure 6.2 presents an EG (left) and its equivalent SG (right) obtained with the construction described above for $C = 1$. For the sake of readability, only reachable states from the initial state $(r, 1)$ are represented. In the SG, the only strategy for Player 1 to avoid to leave the set α is to go from $(r, 1)$ to $(s, 0)$. Its equivalent strategy in the EG is to go from r to s. One can easily see that those two strategies are winning in the respective game (for initial credit $c_0 = 1$ in the EG). ◁

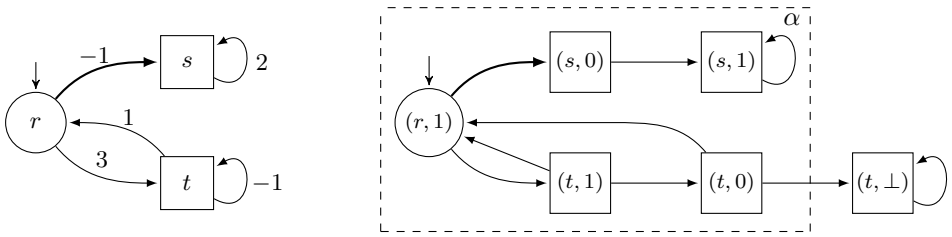

Figure 6.2: EG (left) and its equivalent SG (right).

By Proposition 6.11, it follows that ESGs can be reduced to SGs, as stated in the next theorem.

Theorem 6.14. *Let $\langle G, w, \alpha\rangle$ be an* ESG. *Then, one can construct a* SG $\langle \mathcal{G}', \alpha'\rangle$ *such that Player* 1 *has a winning strategy in $\langle \mathcal{G}, w, \alpha\rangle$ if and only if he has a winning strategy in $\langle \mathcal{G}', \alpha'\rangle$.* □

Energy automata. In Section 5.2, we introduced infinite word automata with new acceptance conditions. We here enrich those automata with weights. Let $\langle P, w\rangle$ with P a finite set of signals and let w be a weight function over $\mathsf{Lit}(P)$. We introduce the notion of energy automata over the weighted alphabet $\langle \Sigma_P, w\rangle$.

Let $\mathcal{A} = (\Sigma_P, Q, q_0, \alpha, \delta)$ be a UCB over the alphabet Σ_P. The related *energy universal co-Büchi automaton* (eUCB) $\mathcal{A}^w$ is over the weighted alphabet $\langle \Sigma_P, w\rangle$ and has the same structure as $\mathcal{A}$. Given an initial credit $c_0 \in \mathbb{N}$, a run $\rho \in Q^\omega$ on a word $u \in \Sigma_P^\omega$ is accepting in the eUCB $\mathcal{A}^w$ if *(i)* ρ satisfies the co-Büchi acceptance condition, and *(ii)* $\forall n \geq 0, c_0 + \mathsf{EL}(u(n)) \geq 0$. We denote by $\mathcal{L}_{\text{ucb}}(\mathcal{A}^w, c_0)$ the

language recognized by $\mathcal{A}^w$ with the given initial credit c_0, that is, the set of words $u \in \Sigma_P^\omega$ such that all runs of $\mathcal{A}^w$ on u are accepting. We also have the notion of *energy universal K-co-Büchi automaton* (eUKCB) $\langle \mathcal{A}^w, K \rangle$ and the related recognized language $\mathcal{L}_{\mathrm{ucb},K}(\mathcal{A}^w, c_0)$. Notice that if $K \leq K'$, then $\mathcal{L}_{\mathrm{ucb},K}(\mathcal{A}^w, c_0) \subseteq \mathcal{L}_{\mathrm{ucb},K'}(\mathcal{A}^w, c_0)$, and that if $c_0 \leq c_0'$, then $\mathcal{L}_{\mathrm{ucb},K}(\mathcal{A}^w, c_0) \subseteq \mathcal{L}_{\mathrm{ucb},K}(\mathcal{A}^w, c_0')$.

Recall from Section 5.2 that UKCBs can be determinized with the subset construction extended with counters. This construction also holds for eUKCBs, and, given an eUKCB $\langle A^w, K \rangle$, the automaton $\det(\mathcal{A}^w, K)$ has the following properties.

Proposition 6.15. *Let $K \in \mathbb{N}$ and $\langle \mathcal{A}^w, K \rangle$ be an* eUKCB*. Then, $\det(\mathcal{A}^w, K)$ is a deterministic and complete energy automaton such that, for all $c_0 \in \mathbb{N}$, $\mathcal{L}_{\mathrm{ucb},0}(\det(\mathcal{A}^w, K), c_0) = \mathcal{L}_{\mathrm{ucb},K}(\mathcal{A}^w, c_0)$.* □

Reduction to safety games. We now go through a series of results that allow us to construct a SG from an eUCB $\mathcal{A}^w$ such that, given an initial credit $c_0 \in \mathbb{N}$, $\mathcal{L}_{\mathrm{ucb}}(\mathcal{A}^w, c_0) = [\![\phi]\!] \cap \{u \in \Sigma_P^\omega \mid \forall n \geq 0, c_0 + \mathsf{EL}(u(n)) \geq 0\}$ (see Theorem 6.18). Recall from Proposition 5.6 that for all LTL formula ϕ, there exists a UCB $\mathcal{A}$ such that $\mathcal{L}_{\mathrm{ucb}}(\mathcal{A}) = [\![\phi]\!]$. From Proposition 6.8 and the definition of Moore machines, we get the next proposition.

Proposition 6.16. *Let ϕ be an* LTL *formula over $\langle P, w \rangle$, $\mathcal{A}$ be a* UCB *such that $\mathcal{L}_{\mathrm{ucb}}(\mathcal{A}) = [\![\phi]\!]$ and $\mathcal{A}^w$ be the related* eUCB*. Then, ϕ is* E*-realizable with the initial credit c_0 if and only if there exists a Moore machine $\mathcal{M}$ such that $\mathcal{L}(\mathcal{M}) \subseteq \mathcal{L}_{\mathrm{ucb}}(\mathcal{A}^w, c_0)$.* □

Theorem 6.17. *Let ϕ be an* LTL *formula over $\langle P, w_P \rangle$ and $\langle \mathcal{G}_\phi = (S, s_0, E), w, p \rangle$ be the associated* EPG *with $d = |\mathsf{Img}(p)|$. Let $\mathcal{A}$ be a* UCB *with n states such that $\mathcal{L}_{\mathrm{ucb}}(\mathcal{A}) = [\![\phi]\!]$. Let $\mathbb{K} = 4 \cdot n \cdot |S|^2 \cdot d \cdot W$ and $\mathbb{C} = \mathbb{K} \cdot W$. Then, ϕ is* E*-realizable if and only if there exists a Moore machine $\mathcal{M}$ such that $\mathcal{L}(\mathcal{M}) \subseteq \mathcal{L}_{\mathrm{ucb},\mathbb{K}}(\mathcal{A}^w, \mathbb{C})$.*

Proof. By Propositions 6.8 and 6.16, ϕ is E-realizable for some initial credit c_0 if and only if there exists a Moore machine $\mathcal{M}$ such that $\mathcal{L}(\mathcal{M}) \subseteq \mathcal{L}_{\mathrm{ucb}}(\mathcal{A}^w, c_0)$ and $|\mathcal{M}| = 4 \cdot |S|^2 \cdot d \cdot W$. Consider now the product of $\mathcal{M}$ and $\mathcal{A}^w$: in any accessible cycle of this product, there is no accepting state of $\mathcal{A}^w$ (as shown similarly for

the qualitative case in [FJR11]) and the sum of the weights must be positive. The length of a path reaching such a cycle is at most $n \cdot |\mathcal{M}|$, therefore, one gets $\mathcal{L}(\mathcal{M}) \subseteq \mathcal{L}_{\text{ucb},n\cdot|\mathcal{M}|}(\mathcal{A}^w, n \cdot |\mathcal{M}| \cdot W)$. □

Theorem 6.18. *Let ϕ be an* LTL *formula. Then, one can construct a* SG *in which Player* 1 *has a winning strategy if and only if ϕ is* E*-realizable.*

Proof. Given an LTL formula ϕ, let us describe the structure of the announced SG $\langle \mathcal{G}_{\phi,\mathbb{K},\mathbb{C}}, \alpha \rangle$. The involved constants $\mathbb{K}$ and $\mathbb{C}$ are those of Theorem 6.17. By Theorem 6.14, it is enough to show the statement with an ESG instead of a SG. The construction of this ESG is very similar to the construction of a MPPG from a DP as given in the proof of Proposition 6.4. The main difference is that we will here use a UKCB instead of a parity automaton.

Let ϕ be an LTL formula and $\mathcal{A}$ be a UCB such that $\mathcal{L}_{\text{ucb}}(\mathcal{A}) = [\![\phi]\!]$. By Theorem 6.17 and Proposition 6.15, ϕ is E-realizable if and only if there exists a Moore machine $\mathcal{M}$ such that $\mathcal{L}(\mathcal{M}) \subseteq \mathcal{L}_{\text{ucb},0}(\det(\mathcal{A}^w, \mathbb{K}), \mathbb{C})$. Exactly as in Proposition 6.4, we derive from $\det(\mathcal{A}^w, \mathbb{K}) = (\langle \Sigma_P, w \rangle, \mathcal{F}, F_0, \beta, \Delta)$ a turn-based ESG $\langle \mathcal{G}_{\phi,\mathbb{K}}, w, \alpha \rangle$ as follows. The construction of the graph and the definition of w are the same, and $\alpha = \mathcal{F} \setminus \beta$. Similarly we have a bijection $\Theta : \Sigma_P^* \to \mathsf{Pref}(\mathcal{G}_{\phi,\mathbb{K}}) \cap (S_1 S_2)^* S_1$ that can be extended to a bijection $\Theta : \Sigma_P^\omega \to \mathsf{Plays}(\mathcal{G}_{\phi,\mathbb{K}})$. One can verify that for each $u \in \Sigma_P^\omega$, we have that *(i)* $\mathsf{EL}(u(n)) = \mathsf{EL}_{\mathcal{G}_{\phi,\mathbb{K}}}(\Theta(u(n)))$ for all $n \geq 0$, and *(ii)* $\sum_{q \in \beta} \mathsf{Visit}(\rho, q) = 0$ for all runs ρ on u if and only if $\Theta(u)$ satisfies the objective $\mathsf{Safety}_{\mathcal{G}_{\phi,\mathbb{K}}}(\alpha)$. It follows that $u \in \mathcal{L}_{\text{ucb},0}(\det(\mathcal{A}^w, \mathbb{K}), \mathbb{C})$ if and only if $\Theta(u)$ satisfies the combined objective $\mathsf{Safety}_{\mathcal{G}_{\phi,\mathbb{K}}}(\alpha) \cap \mathsf{PosEnergy}_{\mathcal{G}_{\phi,\mathbb{K}}}(\mathbb{C})$ (*).

Suppose that ϕ is E-realizable. By Theorem 6.17, there exists a finite-memory strategy λ_O represented by a Moore machine $\mathcal{M}$ as given before. As in the proof of Proposition 6.4, we use Θ to derive a strategy λ_1 of Player 1 that mimics λ_O. As $\mathcal{L}(\mathcal{M}) \subseteq \mathcal{L}_{\text{ucb},0}(\det(\mathcal{A}^w, \mathbb{K}), \mathbb{C})$, by (*), it follows that λ_1 is winning in the ESG $\langle \mathcal{G}_{\phi,\mathbb{K}}, w, \alpha \rangle$ with the initial credit $\mathbb{C}$.

Conversely, suppose that λ_1 is a winning strategy in $\langle \mathcal{G}_{\phi,\mathbb{K}}, w, \alpha \rangle$ with the initial credit $\mathbb{C}$. We again use Θ to derive a strategy $\lambda_O : (\Sigma_O \Sigma_I)^* \to \Sigma_O$ that mimics λ_1. As λ_1 is winning, by (*), we have $\mathsf{Outcome}(\lambda_O) \subseteq \mathcal{L}_{\text{ucb},0}(\det(\mathcal{A}^w, \mathbb{K}), \mathbb{C})$. It follows that ϕ is E-realizable. □

A careful analysis of the complexity of the proposed Safraless procedure shows that it is in 2ExpTime.

6.4 Antichain based algorithms

In Section 6.3, we have shown how to reduce the LTL_E realizability problem to a SG. In this section we explain how to efficiently and symbolically solve this SG with antichains.

6.4.1 Description of the safety game

In the proof of Theorem 6.18, we have shown how to construct a SG $\langle \mathcal{G}_{\phi,\mathbb{K},\mathbb{C}}, \alpha \rangle$ from an LTL formula ϕ such that ϕ is E-realizable if and only if Player 1 has a winning strategy in this game. Let us give the precise construction of this game, but more generally for any values $K, C \in \mathbb{N}$. Let $\mathcal{A} = (\langle \Sigma_P, w \rangle, Q, q_0, \alpha, \delta)$ be a UCB such that $\mathcal{L}_{\text{ucb}}(\mathcal{A}) = [\![\phi]\!]$, and $\det(\mathcal{A}^w, K) = (\langle \Sigma_P, w \rangle, \mathcal{F}, F_0, \beta, \Delta)$ be the related energy deterministic automaton (recall the construction of $\det(\mathcal{A}^w, K)$ from Section 5.2.1). Let $\mathcal{C} = \{\bot, 0, 1, \ldots, C\}$. The turn-based safety game $\langle \mathcal{G}_{\phi,K,C}, \alpha \rangle$ with $\mathcal{G}_{\phi,K,C} = (S = S_1 \uplus S_2, s_0, E)$ has the following structure:

- $S_1 = \{(F, i, c) \mid F \in \mathcal{F}, i \in \Sigma_I, c \in \mathcal{C}\}$,
- $S_2 = \{(F, o, c) \mid F \in \mathcal{F}, o \in \Sigma_O, c \in \mathcal{C}\}$,
- $s_0 = (F_0, j_0, C)$ with j_0 be an arbitrary symbol of Σ_I, and F_0 be the initial state of $\det(\mathcal{A}^w, K)$,
- For all $\Delta(F, o \cup i) = F'$, $j \in \Sigma_I$ and $c \in \mathcal{C}$, the set E contains the edges

$$((F, j, c), (F, o, c')) \text{ and } ((F, o, c'), (F', i, c''))$$

  where $c' = c \oplus w(o)$ and $c'' = c' \oplus w(i)$, and
- $\alpha = S \setminus \{(F, \sigma, c) \mid \exists q, F(q) = \top \text{ or } c = \bot\}$.

Remark 6.19. It is possible to reduce the size of the SG $\langle \mathcal{G}_{\phi,K,C}, \alpha \rangle$ such that $S_1 = \{(F, c) \mid F \in \mathcal{F}, c \in \mathcal{C}\}$ instead of $\{(F, i, c) \mid F \in \mathcal{F}, i \in \Sigma_I, c \in \mathcal{C}\}$. We refer to the proof of Proposition 6.4 and footnote 5 for the justification. However, our definition simplifies the presentation of the remaining of this section. ◁

Example 6.20. Let $\phi = \square(r \Rightarrow \mathsf{X}\lozenge g)$ be an LTL formula over $P = I \uplus O$ with $I = \{r\}$ and $O = \{g\}$, let $w : \mathsf{Lit}(P) \to \mathbb{Z}$ be a weight function such that $w(r) = 1$, $w(g) = -1$ and $w(\neg r) = w(\neg g) = 0$, and let $\nu = 0$ be a threshold. This LTL_MP specification states that every request must be eventually granted (LTL

formula ϕ) with the additional requirement that unsolicited grants are prohibited (weight function w). This kind of specification is discussed in more details in Section 6.6.2. The UCB $\mathcal{A}$ equivalent to ϕ and the deterministic UCB $\det(\mathcal{A}, 1)$ are depicted in Figure 6.3, where $F_0 = (q_0 \mapsto 0, q_1 \mapsto -1)$, $F_1 = (q_0 \mapsto 0, q_1 \mapsto 1)$ and $F_2 = (q_0 \mapsto 0, q_1 \mapsto \top)$.

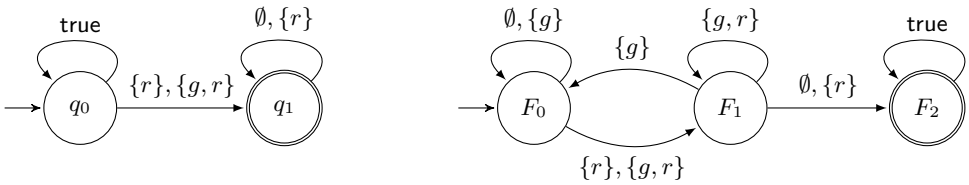

Figure 6.3: UCB $\mathcal{A}$ equivalent to $\square(r \Rightarrow \mathsf{X}\Diamond g)$ (left) and UCB $\det(\mathcal{A}, 1)$ (right).

Figure 6.4 presents the turn-based SG $\langle \mathcal{G}_{\phi,K,C} = (S, s_0, E), \alpha \rangle$ with $K = 1$ and $C = 1$ constructed with the procedure described above. For the sake of readability, only reachable states from s_0 are represented. Moreover, we do not represent successors of states $s \notin \alpha$. The set α contains all states $(F, \sigma, c) \in S$ such that $F \neq F_2 = (q_0 \mapsto 0, q_1 \mapsto \top)$ and $c \neq \bot$. One can easily see that a winning strategy λ_1 for Player 1 is such that $\lambda_1(S^*(F_0, \emptyset, c)) = (F_0, \emptyset, c)$ for all $c \in \mathcal{C}$, $\lambda_1(S^*(F_1, \{r\}, 1)) = (F_1, \{g\}, 0)$, and λ_1 is defined arbitrarily elsewhere.

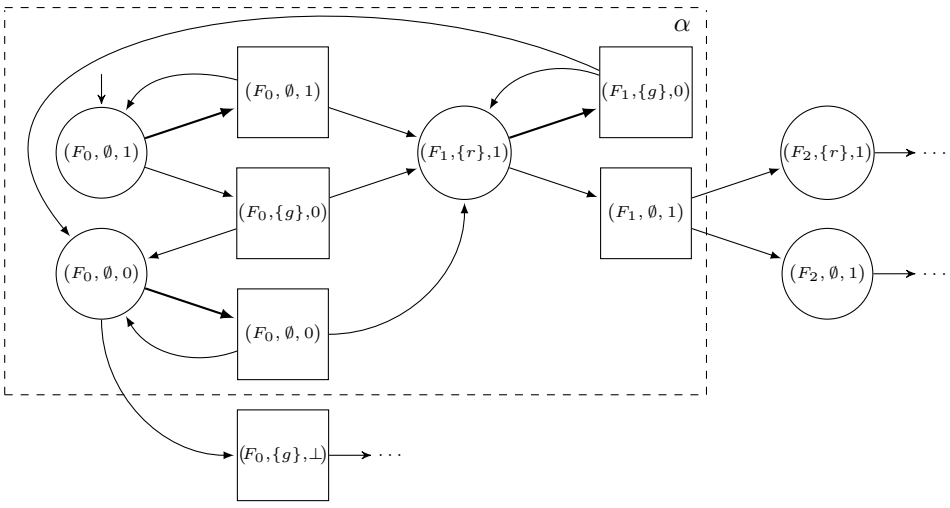

Figure 6.4: Turn-based SG $\langle \mathcal{G}_{\phi,K,C}, \alpha \rangle$ with $K = C = 1$.

◁

Incremental algorithm. Notice that given $K_1, K_2, C_1, C_2 \in \mathbb{N}$ such that $K_1 \leq K_2$ and $C_1 \leq C_2$, if Player 1 has a winning strategy in the SG $\langle \mathcal{G}_{\phi,K_1,C_1}, \alpha \rangle$, then he has a winning strategy in the SG $\langle \mathcal{G}_{\phi,K_2,C_2}, \alpha \rangle$. The next corollary of Theorem 6.18 thus holds.

Corollary 6.21. *Let ϕ be an* LTL *formula, and $K, C \in \mathbb{N}$. If Player 1 has a winning strategy in the* SG $\langle \mathcal{G}_{\phi,K,C}, \alpha \rangle$*, then ϕ is* E*-realizable.* □

This property indicates that testing whether ϕ is E-realizable can be done *incrementally* by solving the family of SGs $\langle \mathcal{G}_{\phi,K,C}, \alpha \rangle$ with increasing values of $K, C \geq 0$ until either Player 1 has a winning strategy in $\langle \mathcal{G}_{\phi,K,C}, \alpha \rangle$ for some K, C such that $0 \leq K \leq \mathbb{K}$, $0 \leq C \leq \mathbb{C}$, or Player 1 has no winning strategy in $\langle \mathcal{G}_{\phi,\mathbb{K},\mathbb{C}}, \alpha \rangle$.

6.4.2 Antichains

The LTL_E realizability problem can be reduced to a family of SGs $\langle \mathcal{G}_{\phi,K,C}, \alpha \rangle$ with $0 \leq K \leq \mathbb{K}$, $0 \leq C \leq \mathbb{C}$. We here show how to make more efficient the fixpoint algorithm to check whether Player 1 has a winning strategy in $\langle \mathcal{G}_{\phi,K,C}, \alpha \rangle$, by avoiding to explicitly construct this SG. This is possible because the states of $\mathcal{G}_{\phi,K,C}$ can be partially ordered, and the sets manipulated by the fixpoint algorithm can be compactly represented by the antichain of their maximal elements.

Partial order on game states. Consider the SG $\langle \mathcal{G}_{\phi,K,C}, \alpha \rangle$ with $\mathcal{G}_{\phi,K,C} = (S = S_1 \uplus S_2, s_0, E)$ as defined in the previous section. We define the binary relation $\preceq \subseteq S \times S$ by

$$(F', \sigma, c') \preceq (F, \sigma, c) \text{ if and only if } (i)\ F' \leq F \text{ and } (ii)\ c' \geq c$$

where $F, F' \in \mathcal{F}$, $\sigma \in \Sigma_P$, $c, c' \in \mathcal{C}$, and $F' \leq F$ if and only if $F'(q) \leq F(q)$ for all q. It is clear that $\preceq$ is a partial order. Intuitively, if Player 1 can win the SG from (F, σ, c), then he can also win from all $(F', \sigma, c') \preceq (F, \sigma, c)$ as (i) it is more difficult to avoid $\top$ from F than from F', and (ii) the energy level is higher with c' than with c. Formally, $\preceq$ is a game simulation relation in the terminology of [AHKV98]. Finally, we have that the partially ordered

set $\langle S, \preceq \rangle$ is a lower semilattice with greatest lower bound $\sqcap$ such that, for all $(F_1, \sigma, c_1), (F_2, \sigma, c_2) \in S$,

$$(F_1, \sigma, c_1) \sqcap (F_2, \sigma, c_2) \stackrel{\text{def}}{=} (q \mapsto \min(F_1(q), F_2(q)), \sigma, \max(c_1, c_2)).$$

The next lemma will be useful later.

Lemma 6.22. *For all $c, c' \in \mathcal{C}$ such that $c' \geq c$ and $k \in \mathbb{Z}$, we have $c' \oplus k \geq c \oplus k$.*

Proof. We first show that if $c' \oplus k = \bot$, then $c \oplus k = \bot$. It holds since, from $c' \geq c$, we have that $c' = \bot$ implies $c = \bot$ and $c' + k < 0$ implies $c + k < 0$. We now show that $c' \oplus k = \min(C, c' + k) \geq c \oplus k$. It trivially holds if $c \oplus k = \bot$, and otherwise, $\min(C, c' + k) \geq \min(C, c + k) = c \oplus k$, which concludes the proof. □

Backward algorithm with antichains. As done in Section 5.3 for LTL realizability and synthesis, we here show how to efficiently implement the fixpoint algorithm for computing the set $\mathsf{Win}_1(\alpha)$ in the SG $\langle \mathcal{G}_{\phi,K,C} = (S, s_0, E), \alpha \rangle$. Recall that due to its underlying nature, we name it the backward algorithm. We start with the next lemma.

Lemma 6.23. *If $L \subseteq S$ is a closed set, then $\mathsf{CPre}_1(L)$ and $\mathsf{CPre}_2(L)$ are also closed.*

Proof. To get the required property, it is enough to prove that if $(s, s') \in E$ and $r \preceq s$, then there exists $(r, r') \in E$ with $r' \preceq s'$.

Let us first suppose that $(s, s') \in S_1 \times S_2$. Thus, by definition of $\mathcal{G}_{\phi,K,C}$, we have $s = (F, j, c)$ and $s' = (F, o, c \oplus w(o))$ for some $F \in \mathcal{F}$, $c \in \mathcal{C}$, $j \in \Sigma_I$ and $o \in \Sigma_O$. Let $r = (G, j, d) \preceq (F, j, c)$, i.e. $G \leq F$ and $d \geq c$. We define $r' = (G, o, d \oplus w(o))$. Then $(r, r') \in E$, $G \leq F$ and $d \oplus w(o) \geq c \oplus w(o)$ by Lemma 6.22. It follows that $r' \preceq s'$.

Let us now suppose that $(s, s') \in S_2 \times S_1$. We have $s = (F, o, c)$ and $s' = (\Delta(F, o \cup i), i, c \oplus w(i))$. Let $r = (G, o, d) \preceq (F, o, c)$, and let us define $r' = (\Delta(G, o \cup i), i, d \oplus w(i))$. By Lemmas 5.14 and 6.22, we get $\Delta(G, o \cup i) \leq \Delta(F, o \cup i)$ and $d \oplus w(i) \geq c \oplus w(i)$. Therefore, $r' \preceq s'$. □

Notice that in the SG $\langle \mathcal{G}_{\phi,K,C}, \alpha \rangle$, the set α is closed by definition. Therefore, by the previous lemma and Proposition 4.2, the sets W_k computed by the fixpoint

algorithm are closed for all $k \geq 0$, and can thus be compactly represented by their antichain $\lceil W_k \rceil$. We now show how to manipulate those sets efficiently. For this purpose, we consider in more details the following sets of predecessors for each $o \in \Sigma_O$ and $i \in \Sigma_I$:

$$\mathsf{Pre}_o(L) = \{s \in S_1 \mid (s, s') \in E \text{ and } s' = (F, o, c) \in L, \text{ for some } F \in \mathcal{F}, c \in \mathcal{C}\}$$
$$\mathsf{Pre}_i(L) = \{s \in S_2 \mid (s, s') \in E \text{ and } s' = (F, i, c) \in L, \text{ for some } F \in \mathcal{F}, c \in \mathcal{C}\}$$

Notice that $\mathsf{CPre}_1(L) = \cup_{o \in \Sigma_O} \mathsf{Pre}_o(L)$ and $\mathsf{CPre}_2(L) = \cap_{i \in \Sigma_I} \mathsf{Pre}_i(L)$. Given $(F, o, c) \in S_2$ and $(F, i, c) \in S_1$, we define:

$$\Omega(F, o, c) = \begin{cases} \{(F, i, c') \mid i \in \Sigma_I, c' = \min\{d \in \mathcal{C} \mid d \oplus w(o) \geq c\}\} & \text{if } c' \text{ exists} \\ \emptyset \quad \text{otherwise.} \end{cases}$$

$$\Omega(F, i, c) = \begin{cases} \{(F', o, c') \mid o \in \Sigma_O, F' = \max\{G \in \mathcal{F} \mid \Delta(G, o \cup i) \leq F\}, \\ \qquad\qquad c' = \min\{d \in \mathcal{C} \mid d \oplus w(i) \geq c\}\} & \text{if } c' \text{ exists} \\ \emptyset \quad \text{otherwise.} \end{cases}$$

When defining the set $\Omega(F, \sigma, c)$, we focus on the worst predecessors with respect to the partial order $\preceq$. In this definition, c' may not exist since the set $\{d \in \mathcal{C} \mid d \oplus w(\sigma) \geq c\}$ can be empty. However the set $\{G \in \mathcal{F} \mid \Delta(G, o \cup i) \leq F\}$ always contains $G : q \mapsto -1$. Moreover, even if $\preceq$ is a partial order, $\max\{G \in \mathcal{F} \mid \Delta(G, o \cup i) \leq F\}$ is unique. Indeed, if $\Delta(G_1, o \cup i) \leq F$ and $\Delta(G_2, o \cup i) \leq F$, then $\Delta(G, o \cup i) \leq F$ with $G : q \mapsto \max(G_1(q), G_2(q))$.

Proposition 6.24. *For all $F \in \mathcal{F}$, $\sigma \in \Sigma_P$ and $c \in \mathcal{C}$, $\mathsf{Pre}_\sigma(\downarrow(F, \sigma, c)) = \downarrow\Omega(F, \sigma, c)$.*

Proof. We only give the proof for $\sigma = i \in \Sigma_I$ since the case $\sigma \in \Sigma_O$ is a particular case. We prove the two following inclusions.

1. $\mathsf{Pre}_i(\downarrow(F, i, c)) \subseteq \downarrow\Omega(F, i, c)$:
   Let $s' = (G, i, d) \preceq (F, i, c)$ and $s = (G', o, d')$ such that $(s, s') \in E$. We have to show that $s \preceq \Omega(F, i, c)$. As $(s, s') \in E$, we have $\Delta(G', o \cup i) = G \leq F$ and $d' \oplus w(i) = d \geq c$. It follows that $(G', o, d') \preceq \Omega(F, i, c)$ by definition of $\Omega(F, i, c)$.
2. $\downarrow\Omega(F, i, c) \subseteq \mathsf{Pre}_i(\downarrow(F, i, c))$:
   Let $(F', o, c') \in \Omega(F, i, c)$ and $s = (G', o, d') \preceq (F', o, c')$. We have to show

that there exists $(s, s') \in E$ with $s' \preceq (F, i, c)$. By definition of $\Omega(F, i, c)$, we have $\Delta(F', o \cup i) \leq F$ and $c' \oplus w(i) \geq c$. As $G' \leq F'$ and $d' \geq c'$, it follows that $\Delta(G', o \cup i) \leq \Delta(F', o \cup i) \leq F$ and $d' \oplus w(i) \geq c' \oplus w(i) \geq c$ by Lemmas 5.14 and 6.22. Therefore with $s' = (\Delta(G', o \cup i), i, d' \oplus w(i))$, we have $(s, s') \in E$ and $s' \preceq (F, i, c)$. Thus $(G', o, d') \in \mathsf{Pre}_i(\downarrow(F, i, c))$.

□

It follows from Propositions 4.2 and 6.24 that the computation steps of the backward algorithm can be limited to antichains.

Corollary 6.25. *Let $L \subseteq S$ be an antichain. Then,*

$$\mathsf{CPre}_1(L) = \bigcup_{o \in \Sigma_O} \bigcup_{(F,o,c) \in L} \downarrow\Omega(F, o, c), \text{ and}$$

$$\mathsf{CPre}_2(L) = \bigcap_{i \in \Sigma_I} \bigcup_{(F,i,c) \in L} \downarrow\Omega(F, i, c).$$

Optimization. The definition of $\Omega(F, i, c)$ requires to compute $\max\{G \in \mathcal{F} \mid \Delta(G, o \cup i) \leq F\}$. This computation can be done more efficiently using the operator $\ominus : \mathcal{K} \times \{0, 1\} \to \mathcal{K}$ defined as follows: $k \ominus b = \top$ if $k = \top$, $k \ominus b = -1$ if $k \neq \top, k - b \leq -1$, and $k \ominus b = k - b$ in all other cases. Indeed, using Lemma 4 of [FJR11], one can see that

$$\max\{G \in \mathcal{F} \mid \Delta(G, o \cup i) \leq F\} = q \mapsto \min\{F(q') \ominus (q' \in \alpha') \mid q' \in \delta(q, o \cup i)\}.$$

LTL_E synthesis. If Player 1 has a winning strategy in the SG $\langle \mathcal{G}_{\phi,K,C}, \alpha \rangle$, that is, the given formula ϕ is E-realizable, then it is easy to construct a Moore machine $\mathcal{M}$ that realizes it. As described in Section 5.3, $\mathcal{M}$ can be constructed from the antichain $\lceil \mathsf{Win}_1(\alpha) \rceil$ computed by the backward algorithm, with the advantage of having a small size bounded by the size of $\lceil \mathsf{Win}_1(\alpha) \rceil \cap S_1$ (see Section 6.6 for experimental results).

Forward algorithm with antichains. The forward algorithm mentioned in Section 5.3 for the LTL realizability can be adapted for solving the SG $\langle \mathcal{G}_{\phi,K,C}, \alpha \rangle$. Again, it is not necessary to construct the SG explicitly, and antichains can be used.

6.5 Extension to multi-dimensional weights

In this section, we consider the natural extension of the $\mathsf{LTL}_{\mathsf{MP}}$ and $\mathsf{LTL}_{\mathsf{E}}$ realizability problems to multi-dimensional weights.

Definitions. Given a set P of atomic propositions, we define a weight function $w : \mathsf{Lit}(P) \to \mathbb{Z}^m$, for some dimension $m \geq 1$. The concepts of energy level EL, mean-payoff value MP, and value Val are defined similarly to the one-dimensional case. Given an LTL formula ϕ over $\langle P, w\rangle$ and a threshold $\nu \in \mathbb{Q}^m$, the *multi-dimensional* $\mathsf{LTL}_{\mathsf{MP}}$ *realizability problem (under finite memory)* asks to decide whether there exists a (finite-memory) strategy λ_O of Player O such that $\mathsf{Val}(\mathsf{Outcome}(\lambda_O, \lambda_I)) \geq \nu$ against all strategies λ_I of Player I.[6] The *multi-dimensional* $\mathsf{LTL}_{\mathsf{E}}$ *realizability problem* asks to decide whether there exists a strategy λ_O of Player O and an initial credit $c_0 \in \mathbb{N}^m$ such that for all strategies λ_I of Player I, *(i)* $u = \mathsf{Outcome}(\lambda_O, \lambda_I) \models \phi$, *(ii)* $\forall n \geq 0,\ c_0 + \mathsf{EL}(u(n)) \geq (0, \ldots, 0)$.

Computational complexity. The 2ExpTime-completeness of the $\mathsf{LTL}_{\mathsf{MP}}$ and $\mathsf{LTL}_{\mathsf{E}}$ realizability problems in one dimension have been stated in Theorems 6.3 and 6.7. In the multi-dimensional case, we have the next result.

Theorem 6.26. *The multi-dimensional* $\mathsf{LTL}_{\mathsf{MP}}$ *realizability under finite memory and the multi-dimensional* $\mathsf{LTL}_{\mathsf{E}}$ *realizability problems are in co-N2ExpTime.*

Proof. We proceed as in the proof of the Theorems 6.3 and 6.7 by reducing the $\mathsf{LTL}_{\mathsf{MP}}$ (resp. $\mathsf{LTL}_{\mathsf{E}}$) realizability problem of formula ϕ to a MMPPG (resp. MEPG) $\langle G_\phi, w, p, m\rangle$. By Proposition 2.22, it is enough to study the multi-dimensional $\mathsf{LTL}_{\mathsf{E}}$ realizability problem. We reduce the MEPG $\langle \mathcal{G}_\phi, w, p, m\rangle$ to a MEG $\langle \mathcal{G}_\phi, w', m'\rangle$ as described in Proposition 2.19. Careful computations show that the multi-dimensional $\mathsf{LTL}_{\mathsf{E}}$ realizability problem is in co-N2ExpTime, by using Theorems 2.10 and 2.20. □

Theorem 6.26 states the complexity of the multi-dimensional $\mathsf{LTL}_{\mathsf{MP}}$ realizability problem under finite memory. Notice that it is reasonable to ask for finite-memory (instead of any) winning strategies. Indeed, the previous proof

[6] With $a \geq b$, we mean $a(i) \geq b(i)$ for all i, $1 \leq i \leq m$.

indicates a reduction to a MMPG; winning strategies for Player 1 in such games require infinite memory in general; however, if Player 1 has a winning strategy for threshold ν, then he has a finite-memory one for threshold $\nu - \epsilon$ for all $\epsilon > 0$ [VCD+12].

Safraless algorithm. As done for the one-dimensional case, we can similarly show that the multi-dimensional $\mathsf{LTL}_{\mathsf{E}}$ realizability problem can be reduced to a SG for which there exist symbolic antichain based algorithms. The multi-dimensional $\mathsf{LTL}_{\mathsf{MP}}$ realizability problem under finite memory can be solved similarly thanks to Proposition 2.22.

Theorem 6.27. *Let ϕ be an* LTL *formula. Then, one can construct a* SG *in which Player* 1 *has a winning strategy if and only if ϕ is* E*-realizable.*

Proof. The proof is very similar to the one of Theorem 6.18. We only indicate the differences.

First, let $\langle \mathcal{G}_\phi, w, p \rangle$ be the MEPG associated with ϕ as described in the proof of Theorem 6.26. By Theorem 2.20 and Proposition 6.8 adapted to this multi-dimensional game, we know that if ϕ is E-realizable, then Player O has a finite-memory strategy with a memory size $|M|$ that is at most exponential in the size of the game.

Second, we need to work with multi-energy automata over a weighted alphabet $\langle \Sigma_P, w \rangle$, such that w is a function over $\mathsf{Lit}(P)$ that assigns vectors of weights of dimension m instead of single weights.

Third, from a UCB $\mathcal{A}$ with n states such that $\mathcal{L}_{\mathrm{b}}(\mathcal{A}) = [\![\phi]\!]$, we construct, similarly to the one-dimensional case, a SG $\langle \mathcal{G}_{\phi,\mathbb{K},(\mathbb{C},\ldots,\mathbb{C})}, \alpha \rangle$, the states of which store a counter for each state of $\mathcal{A}$ and an energy level for each dimension. The constants $\mathbb{K}$ and $\mathbb{C}$ are defined differently from the one-dimensional case: $\mathbb{K} = n \cdot |M|$, and $\mathbb{C}$ is equal to the constant C of Theorem 2.20. □

Antichain based algorithms. Similarly to the one-dimensional case, testing whether an LTL formula ϕ is E-realizable can be done incrementally by solving a family of SGs related to the SG given in Theorem 6.27. These games can be symbolically solved by the antichain based backward and forward algorithms described in Section 6.4.

6.6 Experiments

In the previous sections, in one or several dimensions, we have shown how to reduce the LTL_E and LTL_MP under finite memory realizability and synthesis problems to a family of SGs depending on some values K and C, and how to derive symbolic and incremental antichain based algorithms. This approach has been implemented in the tool Acacia+ [Aca] presented in Section 5.6. We now present some experimental results. All experiments were carried out on a Linux platform with a 3.2GHz CPU (Intel Core i7) and 12GB of memory. Recall that Acacia+ is single-threaded and thus uses only one core.

6.6.1 Approaching the optimal value

Let us come back to Example 6.2 (page 88), where we have given a specification ϕ together with a one-dimensional mean-payoff objective. For the optimal value ν_ϕ, we have shown that no finite-memory strategy exists, but finite-memory ϵ-optimal strategies exist for all $\epsilon > 0$. In Table 6.1, we present the experiments done for some values of $\nu_\phi - \epsilon$.

Table 6.1: Acacia+ v2.2 on the specification of Example 6.2 (page 88) with increasing threshold values. The column ν gives the threshold, K and C the minimum values required to obtain a winning strategy, $|\mathcal{M}|$ the size of the synthesized Moore machine, *time* the total execution time (in seconds) and *mem* the total memory consumption (in megabytes). Note that the execution times are given for the forward algorithm applied to the SG with values K and C (and not with smaller ones).

ν	K	C	$\lvert\mathcal{M}\rvert$	time	mem
-1.2	4	7	5	0.01	16.04
-1.02	49	149	50	0.02	15.95
-1.002	499	1499	500	0.33	17.69
-1.001	999	2999	1000	0.86	19.13
-1.0002	4999	14999	5000	12.64	32.49
-1.0001	9999	29999	10000	47.79	50.88
-1.00005	19999	99999	20000	255.14	101.45

The synthesized strategies for the system behave as follows: grant client 2 $(|\mathcal{M}|-1)$ times, then grant once client 1, and start over. Thus, the system almost always plays $\{g_2, w_1\}$, except every $|\mathcal{M}|$ steps where he has to play $\{g_1, w_2\}$. For instance, the strategy computed by Acacia+ for $\nu = -1.2$ is depicted in

Figure 6.5. Obviously, these strategies are the smallest ones that ensure the corresponding threshold values. They can also be compactly represented by a two-state automaton with a counter that counts up to $|\mathcal{M}|$.

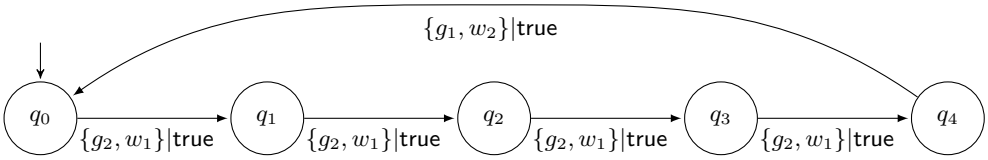

Figure 6.5: Winning strategy for the system computed by Acacia+ for the specification of Example 6.2 for $\nu = -1.2$, with the forward algorithm.

With $\nu = -1.001$ of Table 6.1, let us emphasize the interest of using antichains in our algorithms. The underlying state space manipulated by our symbolic algorithm is huge: since $K = 999$, $C = 2999$ and the number states in the automaton is 8, the number of states in the SG is around 10^{27}. However the fixpoint computed with the backward algorithm is represented by an antichain of size 2004 only.

6.6.2 No unsolicited grants

The major drawback of the strategies presented in Table 6.1 is that many unsolicited grants might be sent since the server grants the resource access to the clients in a round-robin fashion (with a longer access for client 2 than for client 1) without taking care of actual requests made by the clients. It is possible to express in LTL the fact that no unsolicited grants occur, but it is cumbersome. Alternatively, the $\mathsf{LTL}_{\mathsf{MP}}$ specification can be easily rewritten with a multi-dimensional mean-payoff objective to avoid those unsolicited grants, as shown in Example 6.28.

Example 6.28. We consider the client-server system of Examples 6.1 and 6.2 (pages 86 and 88) with the additional requirement that the server does not send unsolicited grants. This property can be naturally expressed by keeping the initial LTL specification ϕ and proposing a multi-dimensional mean-payoff objective as follows. A new dimension is added by client, such that a request (resp. grant) signal of client i has a reward (resp. cost) of 1 on its new dimension. More precisely, given ϕ and P as in Example 6.2, we define $w : \mathsf{Lit}(P) \to \mathbb{Z}^3$ as the

weight function such that $w(r_1) = (0, 1, 0)$, $w(r_2) = (0, 0, 1)$, $w(g_1) = (0, -1, 0)$, $w(g_2) = (0, 0, -1)$, $w(w_1) = (-1, 0, 0)$, $w(w_2) = (-2, 0, 0)$ and $w(l) = (0, 0, 0)$, for all $l \in \mathsf{Lit}(P) \setminus \{r_1, r_2, g_1, g_2, w_1, w_2\}$.

For threshold $\nu = (-1, 0, 0)$, there is no hope to have a finite-memory strategy (see Example 6.2). For threshold $\nu = (-1.2, 0, 0)$ and values $K = 4$, $C = (7, 1, 1)$, Acacia+ outputs a finite-memory winning strategy computed by the backward algorithm, as depicted in Figure 6.6. In this figure, the strategy is represented by an infinite word automaton where the red state is the initial state, and the transitions are labeled with symbols $o|i$ where o (resp. i) is a truth assignment of O (resp. I). Those truth assignments are interpreted as follows. For all $p \in O$, if o contains p (resp. $!p$), then $p \in o$ (resp. $p \notin o$), and for all $p \in I$, if i contains p (resp. $!p$), then $p \in i$ (resp. $p \notin i$). Some truth assignments on transitions are incomplete. For instance, the i part of the label of the transition from state 5 to state 3 is r_1. This means that there is a transition from 5 to 3 for the corresponding o part for truth assignments $r_1, !r_2$ and r_1, r_2 of I. Notice that the labels of all outgoing transitions of a state share the same o part (since we deal with a unique strategy). One can verify that no unsolicited grant is done if the server plays according to this strategy. Moreover, this is the smallest strategy to ensure a threshold of $(-1.2, 0, 0)$ against the most demanding behavior of the clients, i.e. when they both make requests all the time (see states 3 to 7), and that avoid unsolicited grants against any other behaviors of the clients (see states 0 to 2). ◁

From Example 6.28, we derive a benchmark of multi-dimensional examples parameterized by the number of clients making requests to the server. Some experimental results of Acacia+ on this benchmark are summarized in Table 6.2.

Table 6.2: Acacia+ v2.2 on the Shared resource arbiter benchmark parameterized by the number of clients, with the forward algorithm. The column c gives the number of clients and all other columns have the same meaning as in Table 6.1.

c	ν	K	C	$\lvert\mathcal{M}\rvert$	time	mem
2	$(-1.2, 0, 0)$	4	$(7, 1, 1)$	11	0.02	16.35
3	$(-2.2, 0, 0, 0)$	9	$(19, 1, 1, 1)$	27	0.11	16.13
4	$(-3.2, 0, 0, 0, 0)$	14	$(12, 1, 1, 1, 1)$	65	1.61	18.20
5	$(-4.2, 0, 0, 0, 0, 0)$	19	$(29, 1, 1, 1, 1, 1)$	240	51.55	45.34
6	$(-5.2, 0, 0, 0, 0, 0, 0)$	24	$(17, 1, 1, 1, 1, 1, 1)$	1716	3790.91	510.37

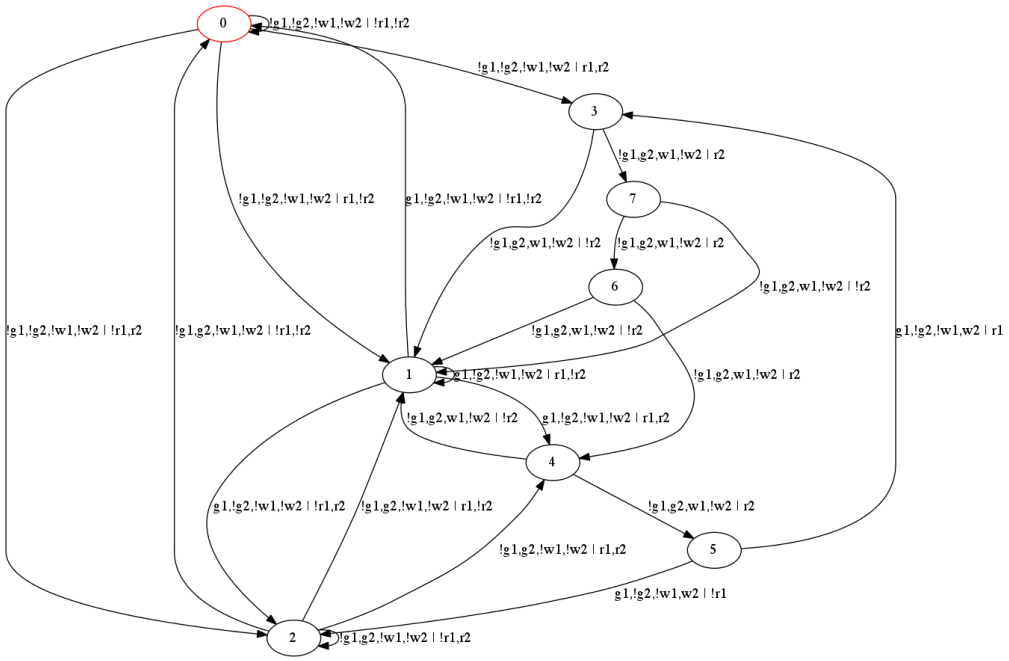

Figure 6.6: Winning strategy for the system output by Acacia+ for the specification of Example 6.28 with threshold $\nu = (-1.2, 0, 0)$ and values $K = 4$, $C = (7, 1, 1)$, using the backward algorithm.

6.6.3 Approaching the Pareto curve

As last experiment, we consider the 2-client $\mathsf{LTL}_{\mathsf{MP}}$ specification of Example 6.28 where we split the first dimension of the weight function into two dimensions, such that $w(w_1) = (-1, 0, 0, 0)$ and $w(w_2) = (0, -2, 0, 0)$. With this new specification, since we have several dimensions, there might be several optimal values for the pairwise order, corresponding to trade-offs between the two objectives that are (i) to quickly grant client 1 and (ii) to quickly grant client 2. In this experiment, we are interested in approaching, by hand, the *Pareto curve*, which consists of all those optimal values, i.e. to find finite-memory strategies that are incomparable with respect to the ensured thresholds, these thresholds being as large as possible. We give some such thresholds in Table 6.3, along with minimum values of K and C and strategies size. It is difficult to automatize the

construction of the Pareto curve. Indeed, Acacia+ cannot test (in reasonable time) whether a formula is MP-unrealizable for a given threshold, since it has to reach the huge theoretical bounds on K and C. Recall from Section 5.3.1 that in the case of LTL realizability, we have an efficient algorithm for checking if an LTL formula is unrealizable. This is not the case for $\mathsf{LTL}_{\mathsf{MP}}$ realizability. This raises two interesting questions that we let as future work: how to decide efficiently that a formula is MP-unrealizable for a given threshold, and how to compute points of the Pareto curve efficiently.

Table 6.3: Acacia+ v2.2 to approach Pareto values. All columns have the same meaning as in Table 6.1.

ν	K	C	$\lvert\mathcal{M}\rvert$
$(-0.001, -2, 0, 0)$	999	$(1999, 1, 1, 1)$	2001
$(-0.15, -1.7, 0, 0)$	55	$(41, 55, 1, 1)$	42
$(-0.25, -1.5, 0, 0)$	3	$(7, 9, 1, 1)$	9
$(-0.5, -1, 0, 0)$	1	$(3, 3, 1, 1)$	5
$(-0.75, -0.5, 0, 0)$	3	$(9, 7, 1, 1)$	9
$(-0.85, -0.3, 0, 0)$	42	$(55, 41, 1, 1)$	9
$(-1, -0.01, 0, 0)$	199	$(1, 399, 1, 1)$	401

Part III

Synthesis of good-for-expectation strategies

CHAPTER 7

Symblicit algorithms for mean-payoff and shortest path in monotonic Markov decision processes

We open Part III devoted to the framework of good-for-expectation strategy synthesis.

Recently, Wimmer et al. [WBB+10] have proposed a BDD based *symblicit* algorithm for the expected mean-payoff (EMP) problem in MDPs, that is, an algorithm that efficiently combines symbolic and explicit representations. In this chapter, we present pseudo-antichain based symblicit algorithms for both the EMP and stochastic shortest path (SSP) problems. Due to the use of antichains, our algorithms apply on a natural class of MDPs, called *monotonic* MDPs.

We first formally define the notion of monotonic MDP. We then recall strategy iteration algorithms for solving the SSP and EMP problems, and we present the symblicit approach proposed in [WBB+10]. Finally, we describe our pseudo-antichain based symblicit algorithms. Two applications of those algorithms, together with experimental results, are considered in the next chapter.

The content of this chapter is based on [BBR14b, BBR14a].

7.1 Monotonic Markov decision processes

In this section, we introduce the notion of monotonic MDP. We first consider the following example to intuitively illustrate this notion in the SSP context.

Example 7.1. Let us consider the following example, inspired from [RN95], where a monkey tries to reach an hanging bunch of bananas. There are several items strewn in the room that the monkey can get and use, individually or simultaneously. There is a *box* on which it can climb to get closer to the bananas, a *stone* that can be thrown at the bananas, a *stick* to try to take the bananas down, and obviously the *bananas* that the monkey wants to eventually obtain. Initially, the monkey possesses no item. The monkey can make actions whose effects are to add and/or to remove items from its inventory. We add stochastic aspects to the problem. For example, using the *stick*, the monkey has probability $\frac{1}{5}$ to obtain the *bananas*, while combining the *box* and the *stick* increases this probability to $\frac{1}{2}$. Additionally, we associate a (positive) weight with each action, representing the time spent executing the action. For example, picking up the *stone* has a weight of 1, while getting the *box* has a weight of 5. The objective of the monkey is then to minimize the expected time for reaching the *bananas*.

This kind of specification naturally defines an MDP. The set S of states of the MDP is the set of all the possible combinations of items. Initially the monkey is in the state with no item. The available actions at each state $s \in S$ depend on the items of s. For example, when the monkey possesses the *box* and the *stick*, it can decide to try to reach the *bananas* by using one of these two items, or the combination of both of them. If it decides to use the *stick* only, it will reach the state $s \cup \{bananas\}$ with probability $\frac{1}{5}$ whereas it will stay at state s with probability $\frac{4}{5}$. This MDP is monotonic in the following sense. First, the set S is a closed set equipped with the partial order $\supseteq$. Second, the action of trying to reach the *bananas* with the *stick* is also available if the monkey possesses the *stick* together with other items. Moreover, if it succeeds (with probability $\frac{1}{5}$), it will reach a state with the *bananas* and all the items it already had at its disposal. In other words, for all states $s' \in S$ such that $s' \supseteq s = \{stick\}$, we have that $\Sigma_s \subseteq \Sigma_{s'}$, and $t' \supseteq t = \{bananas, stick\}$ with t' the state reached from s' with probability $\frac{1}{5}$. Finally, note that the set of goal states $G = \{s \in S \mid bananas \in s\}$ is closed. ◁

New definition of MDPs. To properly define the notion of monotonic MDPs, we need a slightly different, but equivalent, definition of MDPs than the classical one given in Section 2.5. This new definition is based on a set T of stochastic actions. Recall from Chapter 2 that, for all sets X, Y, we denote by $\mathcal{F}_{\text{tot}}(X, Y)$ the set of total functions from X to Y, and by $\mathsf{Dist}(X)$ the set of probability distributions on X. In this new definition, an MDP $\mathcal{M}$ is a tuple $(S, \Sigma, T, \mathcal{E}, \mathcal{D})$ where S is a finite set of states, Σ and T are two finite sets of actions such that $\Sigma \cap T = \emptyset$, $\mathcal{E} : S \times \Sigma \to \mathcal{F}_{\text{tot}}(T, S)$ is a partial successor function, and $\mathcal{D} : S \times \Sigma \to \mathsf{Dist}(T)$ is a partial stochastic function such that $\mathsf{Dom}(\mathcal{E}) = \mathsf{Dom}(\mathcal{D})$. Figure 7.1 intuitively illustrates the relationship between the two definitions.

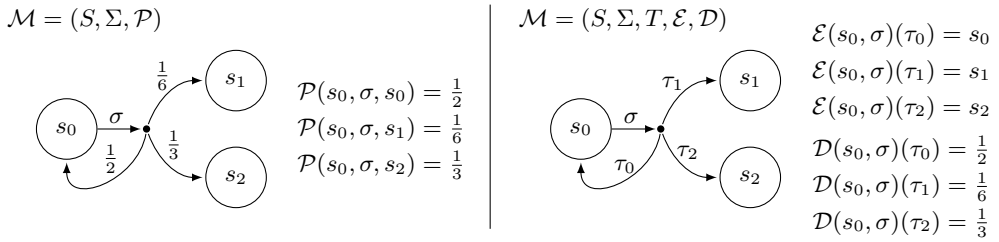

Figure 7.1: Illustration of the new definition of MDPs for a state $s_0 \in S$ and an action $\sigma \in \Sigma_{s_0}$.

Let us explain this relationship more precisely. Let an MDP $(S, \Sigma, T, \mathcal{E}, \mathcal{D})$ as given in the new definition. We can then derive from $\mathcal{E}$ and $\mathcal{D}$ the partial transition function $\mathcal{P} : S \times \Sigma \to \mathsf{Dist}(S)$ such that for all $s, s' \in S$ and $\sigma \in \Sigma_s$,

$$\mathcal{P}(s, \sigma)(s') = \sum_{\substack{\tau \in T \\ \mathcal{E}(s,\sigma)(\tau) = s'}} \mathcal{D}(s, \sigma)(\tau).$$

Conversely, let an MDP $(S, \Sigma, \mathcal{P})$ as in the classical definition. Then we can choose a set T of stochastic actions of size $|S|$ and adequate functions $\mathcal{E}, \mathcal{D}$ to get the second definition $(S, \Sigma, T, \mathcal{E}, \mathcal{D})$ for this MDP (see Figure 7.1).

In this new definition of MDPs, for all states $s \in S$ and all pair of actions $(\sigma, \tau) \in \Sigma \times T$, there is at most one $s' \in S$ such that $\mathcal{E}(s, \sigma)(\tau) = s'$. We thus say that $\mathcal{M}$ is *deterministic*. We also say that $\mathcal{M}$ is T*-complete* since for all pair $(s, \sigma) \in \mathsf{Dom}(\mathcal{E})$, $\mathcal{E}(s, \sigma)$ is a total function mapping each $\tau \in T$ to a state $s \in S$.

Notice that the notion of MC induced by a strategy can also be described in

this new formalism as follows. Given an MDP $(S, \Sigma, T, \mathcal{E}, \mathcal{D})$ and a memoryless strategy λ, we have the induced MC $(S, T, \mathcal{E}_\lambda, \mathcal{D}_\lambda)$ such that $\mathcal{E}_\lambda : S \to \mathsf{Dist}(T)$ is the successor function with $\mathcal{E}_\lambda(s) = \mathcal{E}(s, \lambda(s))$ for all $s \in S$, and $\mathcal{D}_\lambda : S \to \mathsf{Dist}(T)$ is the stochastic function with $\mathcal{D}_\lambda(s) = \mathcal{D}(s, \lambda(s))$ for all $s \in S$.

Depending on the context, we will use both definitions $\mathcal{M} = (S, \Sigma, T, \mathcal{E}, \mathcal{D})$ and $\mathcal{M} = (S, \Sigma, \mathcal{P})$ for MDPs, assuming that $\mathcal{P}$ is always obtained from some set T and partial functions $\mathcal{E}$ and $\mathcal{D}$.

Monotonic MDP**s.** We can now formally define the notion of monotonic MDP.

Definition 7.2 (Monotonic MDP**).** A *monotonic MDP* is an MDP $\mathcal{M}_\preceq = (S, \Sigma, T, \mathcal{E}, \mathcal{D})$ such that:

1. the set S is equipped with a partial order $\preceq$ such that $\langle S, \preceq \rangle$ is a lower semilattice, and
2. the partial order $\preceq$ is *compatible* with $\mathcal{E}$, i.e. for all $s, s' \in S$, if $s \preceq s'$, then for all $\sigma \in \Sigma$, $\tau \in T$, and $t' \in S$ such that $\mathcal{E}(s', \sigma)(\tau) = t'$, there exists $t \in S$ such that $\mathcal{E}(s, \sigma)(\tau) = t$ and $t \preceq t'$.

Note that since $\langle S, \preceq \rangle$ is a lower semilattice, we have that S is closed for $\preceq$. With this definition, and in particular by compatibility of $\preceq$, we have the next proposition.

Proposition 7.3. *The following statements hold for a monotonic* MDP $\mathcal{M}_\preceq$*:*

- *For all $s, s' \in S$, if $s \preceq s'$ then $\Sigma_{s'} \subseteq \Sigma_s$*
- *For all $\sigma \in \Sigma$, S_σ is closed.*

Remark 7.4. All MDPs can be seen monotonic. Indeed, let $(S, \Sigma, T, \mathcal{E}, \mathcal{D})$ be a given MDP and let $\preceq$ be a partial order such that all states in S are pairwise incomparable with respect to $\preceq$. Moreover, let $t \notin S$ be an additional state such that $t \preceq s$, for all $s \in S$, and such that t is not reachable from any state in S and for all $\sigma \in \Sigma, \tau \in T$, $\mathcal{E}(t, \sigma)(\tau) = t$. Then, $\langle S \cup \{t\}, \preceq \rangle$ is a lower semilattice with greatest lower bound $\sqcap$ such that for all $s, s' \in S \cup \{t\}$, $s \sqcap s' = t$. Moreover, $\preceq$ is trivially compatible with $\mathcal{E}$. However, such a partial order would not lead to an efficient algorithm using pseudo-antichains. Indeed, the states in S being pairwise incomparable, they would be explicitly represented, that is, the PA-representation would not be compact. By monotonic MDPs, we rather mean

MDPs that are built on state spaces *already* equipped with a *natural partial order*. For instance, this is the case for the two classes of MDPs studied in Chapter 8. The same kind of approach has been proposed in [FS01]. ◁

7.2 Strategy iteration algorithms

In this section, we present strategy iteration algorithms for computing optimal strategies for the SSP and EMP problems. A *strategy iteration* algorithm [How60] consists in generating a sequence of monotonically improving strategies (along with their associated value functions) until converging to an optimal one. Each iteration is composed of two phases: the *strategy evaluation phase* in which the value function of the current strategy is computed, and the *strategy improvement phase* in which the strategy is improved (if possible) at each state, by using the computed value function. The algorithm stops after a finite number of iterations, as soon as no more improvement can be made, and returns the computed optimal strategy. We follow the presentation of those algorithms as given in [WBB+10].

7.2.1 Stochastic shortest path

We start with the strategy iteration algorithm for the SSP problem [How60, BT96]. Let $\mathcal{M} = (S, \Sigma, \mathcal{P})$ be an MDP, $w : S \times \Sigma \to \mathbb{R}_{<0}$ be a strictly negative weight function, and $G \subseteq S$ be a set of goal states. Recall from Section 2.5 that the solution to the SSP problem is to first compute the set of proper states which are the states having at least one proper strategy.

Computing proper states. An algorithm is proposed in [dA99] for computing in quadratic time the set $S^{\mathsf{p}} = \{s \in S \mid \Lambda_s^{\mathsf{p}} \neq \emptyset\}$ of proper states. To present this algorithm, given two subsets $X, Y \subseteq S$, we define the predicate $\mathsf{APre}(Y, X)$ such that for all $s \in S$,

$$s \models \mathsf{APre}(Y, X) \Leftrightarrow \exists \sigma \in \Sigma_s, (\mathsf{succ}(s, \sigma) \subseteq Y \wedge \mathsf{succ}(s, \sigma) \cap X \neq \emptyset).$$

Then, we can compute the set S^{p} of proper states by the following μ-calculus expression[1]:

$$S^P = \nu Y \cdot \mu X \cdot (\mathsf{APre}(Y, X) \vee \mathsf{G}),$$

[1]Notations ν and μ are classical notations for greatest fixpoint and least fixpoint operators in μ-calculus expressions [Koz83].

where we denote by G a predicate that holds exactly for the states in G. The algorithm works as follows. Initially, we have $Y_0 = S$. At the end of the first iteration, we have $Y_1 = S \setminus C_0$, where C_0 is the set of states that reach G with probability 0. At the end of the second iteration, we have $Y_2 = Y_1 \setminus C_1$, where C_1 is the set of states that cannot reach G without risking to enter C_0 (i.e. states in C_1 have a strictly positive probability of entering C_0). More generally, at the end of iteration $k > 0$, we have $Y_k = Y_{k-1} \setminus C_{k-1}$, where C_{k-1} is the set of states that cannot reach G without risking to enter $\bigcup_{i=0}^{k-2} C_i$. The correctness and complexity results are proved in [dA99].

Given an MDP $\mathcal{M} = (S, \Sigma, \mathcal{P})$ with a weight function w and a set $G \subseteq S$, one can restrict $\mathcal{M}$ and w to the set S^{P} of proper states. We obtain a new MDP $\mathcal{M}^{\mathsf{P}} = (S^{\mathsf{P}}, \Sigma, \mathcal{P}^{\mathsf{P}})$ with weight function w^{P} such that $\mathcal{P}^{\mathsf{P}}$ and w^{P} are the restriction of $\mathcal{P}$ and w to S^{P}. Moreover, for all states $s \in S^{\mathsf{P}}$, we let $\Sigma_s^{\mathsf{P}} = \{\sigma \in \Sigma_s \mid \mathsf{succ}(s, \sigma) \subseteq S^{\mathsf{P}}\}$ be the set of enabled actions in s. Note that by construction of S^{P}, we have $\Sigma_s^{\mathsf{P}} \neq \emptyset$ for all $s \in S^{\mathsf{P}}$, showing that $\mathcal{M}^{\mathsf{P}}$ is Σ-*non-blocking*. To avoid a change of notation, in the sequel of this subsection, we make the assumption that each state of $\mathcal{M}$ is proper.

Example 7.5. Let us consider the following example to illustrate the algorithm of [dA99] for computing proper states. Let $\mathcal{M} = (S, \Sigma, \mathcal{P})$ be the MDP depicted in Figure 7.2 with $G = \{s_3\}$. Initially, we have $Y_0 = S$ and the following sequence is computed: $X_0 = G = \{s_3\}$, $X_1 = \{s_2, s_3\}$ and $X_2 = \{s_0, s_2, s_3\}$ which is the least fix point. We thus have $Y_1 = X_2 = \{s_0, s_2, s_3\}$ and a new least fix point is obtained with $X_1 = \{s_2, s_3\}$. Thus, $Y_2 = \{s_2, s_3\}$, which is the greatest fix point, that is, the set S^{P} of proper states in $\mathcal{M}$. States s_0 and s_1 are not proper since s_1 reaches G with probability 0, and s_0 reaches s_1 with a strictly positive probability with σ_1 and reaches G with probability 0 with σ_0. ◁

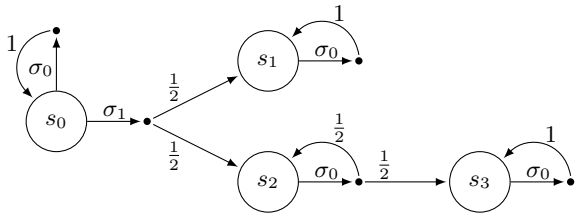

Figure 7.2: MDP to illustrate the computation of the set of proper states.

Strategy iteration algorithm. The strategy iteration algorithm for the SSP problem, named SSP_SI, is given in Algorithm 2[2]. This algorithm is applied under the typical assumption that all cycles in the underlying graph of $\mathcal{M}$ have strictly negative weight [BT96]. This assumption holds in our case by definition of the weight function w. The algorithm starts with an arbitrary proper strategy λ_0, that can be easily computed with the algorithm of [dA99][3], and improves it until an optimal strategy is found. The expected truncated sum v_n of the current strategy λ_n is computed by solving the system of linear equations in line 3, and used to improve the strategy (if possible) at each state. Note that the strategy λ_n is improved at a state s to an action $\sigma \in \Sigma_s$ only if the new expected truncated sum is strictly larger than the expected truncated sum of the action $\lambda_n(s)$, i.e. only if $\lambda_n(s) \notin \arg\max_{\sigma \in \Sigma_s}(w(s,\sigma) + \sum_{s' \in S} \mathcal{P}(s,\sigma,s') \cdot v_n(s'))$. If no improvement is possible for any state, an optimal strategy is found and the algorithm terminates in line 8. Otherwise, it restarts by solving the new equation system, tries to improve the strategy using the new values computed...

Algorithm 2 SSP_SI(MDP $\mathcal{M}$, Strictly negative weight function w, Goal states G)

1: $n := 0, \lambda_n :=$ InitialProperStrategy$(\mathcal{M}, G)$
2: **repeat**
3: Obtain v_n by solving
$$w_{\lambda_n} + (\mathcal{P}_{\lambda_n} - \mathbf{I})v_n = 0$$
4: $\widehat{\Sigma_s} := \arg\max_{\sigma \in \Sigma_s}(w(s,\sigma) + \sum_{s' \in S} \mathcal{P}(s,\sigma,s') \cdot v_n(s')), \forall s \in S$
5: Choose λ_{n+1} s.t. $\lambda_{n+1}(s) \in \widehat{\Sigma_s}, \forall s \in S$, setting $\lambda_{n+1}(s) := \lambda_n(s)$ if possible.
6: $n := n + 1$
7: **until** $\lambda_n = \lambda_{n-1}$
8: return (λ_{n-1}, v_{n-1})

Example 7.6. Let us consider the MDP $\mathcal{M} = (S, \Sigma, \mathcal{P})$ depicted in Figure 7.3 to illustrate the strategy iteration algorithm for the SSP problem. Let $G = \{s_1\}$ be the set of goal states and let $w : S \times \Sigma \to \mathbb{R}_{<0}$ be the strictly negative weight function such that $w(s_0, \sigma_0) = w(s_0, \sigma_1) = w(s_1, \sigma_0) = -1$ and $w(s_0, \sigma_2) = -3$.

[2]If the expected truncated sum has to be minimized, the weight function is restricted to the strictly positive real values and arg max is replaced by arg min in line 4.

[3]We leave out details on how to compute a proper strategy for the moment since we give the description of a pseudo-antichain based algorithm for this task in Section 7.4.3.

Let $\lambda_0 : S \to \Sigma$ be an initial proper strategy such that $\lambda_0(s_0) = \sigma_1$.[4] Note that the strategy λ' such that $\lambda'(s_0) = \sigma_0$ is not proper (see Example 2.28). This strategy can thus not be considered as initial strategy λ_0. By solving the linear system, we obtain values $v_0(s_0) = -10$ and $v_0(s_1) = 0$. We have that $\widehat{\Sigma_{s_0}} = \{\sigma_2\}$ since $w(s_0, \sigma_2) + \sum_{s' \in S} \mathcal{P}(s_0, \sigma_2, s') \cdot v_0(s') = -3 - 2 + 0 = -5 > -10 = v_0(s_0)$. We thus define λ_1 such that $\lambda_1(s_0) = \sigma_2$. During the next iteration of the algorithm, we obtain values $v_1(s_0) = -3.75$ and $v_1(s_1) = 0$ by solving the linear system, and we have $\widehat{\Sigma_{s_0}} = \{\sigma_2\}$. The algorithm thus terminates since $\lambda_1 = \lambda_2$ is the optimal strategy. The expected truncated sum up to G of λ_1 is $\mathbb{E}_{\lambda_1}^{\mathsf{TS}_G}(s_0) = -3.75$ and $\mathbb{E}_{\lambda_1}^{\mathsf{TS}_G}(s_1) = 0$.

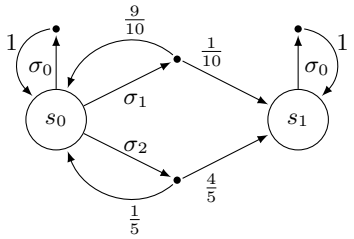

Figure 7.3: MDP to illustrate the strategy iteration algorithm for the SSP. ◁

7.2.2 Expected mean-payoff

We now consider the strategy iteration algorithm for the EMP problem [Vei66, Put94], named EMP_SI (see Algorithm 3[5]). More details can be found in [Put94]. The algorithm starts with an arbitrary strategy λ_0 (here any initial strategy is appropriate). By solving the equation system of line 3, we obtain the gain value g_n and bias value b_n of the current strategy λ_n. The gain corresponds to the expected mean-payoff, while the bias can be interpreted as the expected total difference between the cost and the expected mean-payoff. The computed gain value is then used to locally improve the strategy (lines 4-5). If such an improvement is not possible for any state, the bias value is used to locally improve the strategy (lines 6-7). By improving the strategy with the bias value, only

[4] In this example, we omit to properly define our strategies for s_1 since $|\Sigma_{s_1}| = 1$.

[5] If the expected mean-payoff has to be minimized, one has to replace arg max by arg min in lines 4 and 7.

actions that also optimize the gain can be considered (see set $\widehat{\Sigma_s}$). Finally, the algorithm stops at line 10 as soon as none of those improvements can be made for any state, and returns the optimal strategy λ_{n-1} along with its associated expected mean-payoff.

Algorithm 3 EMP_SI(MDP $\mathcal{M}$, Weight function w)

1: $n := 0, \lambda_n :=$ InitialStrategy($\mathcal{M}$)
2: **repeat**
3: Obtain g_n and b_n by solving
$$\begin{cases} (\mathcal{P}_{\lambda_n} - \mathbf{I})g_n = 0 \\ w_{\lambda_n} - g_n + (\mathcal{P}_{\lambda_n} - \mathbf{I})b_n = 0 \\ \mathcal{P}^*_{\lambda_n} b_n = 0 \end{cases}$$
4: $\widehat{\Sigma_s} := \arg\max_{\sigma \in \Sigma_s} \sum_{s' \in S} \mathcal{P}(s, \sigma, s') \cdot g_n(s'), \forall s \in S$
5: Choose λ_{n+1} s.t. $\lambda_{n+1}(s) \in \widehat{\Sigma_s}, \forall s \in S$, setting $\lambda_{n+1}(s) := \lambda_n(s)$ if possible.
6: **if** $\lambda_{n+1} = \lambda_n$ **then**
7: Choose λ_{n+1} s.t. $\lambda_{n+1}(s) \in \arg\max_{\sigma \in \widehat{\Sigma_s}}(w(s, \sigma) + \sum_{s' \in S} \mathcal{P}(s, \sigma, s') \cdot b_n(s')), \forall s \in S$, setting $\lambda_{n+1}(s) = \lambda_n(s)$ if possible.
8: $n := n + 1$
9: **until** $\lambda_n = \lambda_{n-1}$
10: return (λ_{n-1}, g_{n-1})

7.3 Symblicit approach

In this section, we recall the symblicit algorithm proposed in [WBB+10] for solving the EMP problem on MDPs, which is based on the strategy iteration algorithm presented in the previous section. Our description is more general to suit also for the SSP problem. We first talk about bisimulation lumping, a technique used by the symblicit algorithm to reduce the state space of the models it works on.

7.3.1 Bisimulation lumping

The *bisimulation lumping* technique [KS60, LS91, Buc94] applies to Markov chains (MCs). It consists in gathering certain states of an MC which behave equivalently according to the class of properties under consideration. For the

expected truncated sum and the expected mean-payoff, the following definition of equivalence of two states can be used. Let $(S, \mathcal{P})$ be an MC and $w : S \to \mathbb{R}$ be a weight function on S. Let $\sim$ be an equivalence relation on S and $S_\sim$ be the induced partition. We say that $\sim$ is a *bisimulation* if for all $s, t \in S$ such that $s \sim t$, we have $w(s) = w(t)$ and $\mathcal{P}(s, C) = \mathcal{P}(t, C)$ for all block $C \in S_\sim$, where $\mathcal{P}(s, C) = \sum_{s' \in C} \mathcal{P}(s, s')$.

Let $(S, \mathcal{P})$ be an MC with weight function w, and $\sim$ be a bisimulation on S. The *bisimulation quotient* is the MC $(S_\sim, \mathcal{P}_\sim)$ such that $\mathcal{P}_\sim(C, C') = \mathcal{P}(s, C')$, where $s \in C$ and $C, C' \in S_\sim$. The weight function $w_\sim : S_\sim \to \mathbb{R}$ is transferred to the quotient such that $w_\sim(C) = w(s)$, where $s \in C$ and $C \in S_\sim$. The quotient is thus a minimized model equivalent to the original one for our purpose, since it satisfies properties like expected truncated sum and expected mean-payoff as the original model [BKHW05]. Usually, we are interested in the unique *largest* bisimulation, denoted $\sim_L$, which leads to the smallest bisimulation quotient $(S_{\sim_L}, \mathcal{P}_{\sim_L})$.

Algorithm Lump [DHS03] (see Algorithm 4) describes how to compute the partition induced by the largest bisimulation. This algorithm is based on Paige and Tarjan's algorithm for computing bisimilarity of labeled transition systems [PT87].

Algorithm 4 Lump(MC $\mathcal{M}_\lambda$, Weight function w)

```
1:  P := InitialPartition(M_λ, w)
2:  L := P
3:  while L ≠ ∅ do
4:      C := Pop(L)
5:      P_new := ∅
6:      for all B ∈ P do
7:          {B_1, ..., B_k} := SplitBlock(B, C)
8:          P_new := P_new ∪ {B_1, ..., B_k}
9:          B_l := some block in {B_1, ..., B_k}
10:         L := L ∪ {B_1, ..., B_k} \ B_l
11:     P := P_new
12: return P
```

For a given MC $\mathcal{M}_\lambda = (S, \mathcal{P})$ with weight function w, Algorithm Lump first computes the initial partition P such that for all $s, t \in S$, s and t belongs to

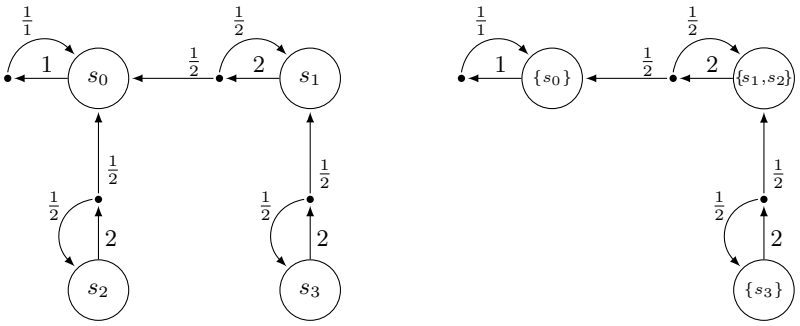

Figure 7.4: MC to illustrate the bisimulation lumping technique (left) and its bisimulation quotient (right).

the same block of P if and only if $w(s) = w(t)$. The algorithm holds a list L of potential splitters of P, where a *splitter* of P is a set $C \subseteq S$ such that $\exists B \in P, \exists s, s' \in B$ such that $\mathcal{P}(s, C) \neq \mathcal{P}(s', C)$. Initially, this list L contains the blocks of the initial partition P. Then, while L is non empty, the algorithm takes a splitter C from L and refines each block of the partition according to C. Algorithm SplitBlock splits a block B into non empty sub-blocks $B_1, \cdots, B_k$ according to the probability of reaching the splitter C, i.e. for all $s, s' \in B$, we have $s, s' \in B_i$ for some i if and only if $\mathcal{P}(s, C) = \mathcal{P}(s', C)$. The block B is then replaced in P by the computed sub-blocks $B_1, \cdots, B_k$. Finally, we add to L the sub-blocks $B_1, \cdots, B_k$, but one which can be omitted since its power of splitting other blocks is maintained by the remaining sub-blocks [DHS03]. In general, we prefer to omit the largest sub-block since it might be the most costly to process as potential splitter. The algorithm terminates when the list L is empty, which means that the partition is refined with respect to all potential splitters, i.e. P is the partition induced by the largest bisimulation $\sim_L$.

Example 7.7. Let us consider the MC $(S, \mathcal{P})$ depicted in Figure 7.4 (left) to illustrate the bisimulation lumping technique. For convenience, the weight $w(s)$ associated with each state s is written on the outgoing edge of s. The initial partition contains two blocks $B_1 = \{s_0\}$ and $B_2 = \{s_1, s_2, s_3\}$, which are added to the list L of splitters (one of them can be omitted). Block B_2 is then split into two blocks $B_2^1 = \{s_1, s_2\}$ and $B_2^2 = \{s_3\}$ according to the probability of its states to reach B_1 (or B_2). States in B_2^1 (resp. in B_2^2) reach B_1 with probability $\frac{1}{2}$ (resp. with probability 0). Since block B_2^1 cannot be split according to any splitter, the

partition $S_{\sim_L}$ induced by the largest bisimulation is $S_{\sim_L} = \{\{s_0\}, \{s_1, s_2\}, \{s_3\}\}$. The bisimulation quotient $(S_{\sim_L}, \mathcal{P}_{\sim_L})$ is depicted in Figure 7.4 (right). ◁

7.3.2 Symblicit algorithm

The algorithmic basis of the symblicit approach is the strategy iteration algorithm (see Algorithm 2 for the SSP problem and Algorithm 3 for the EMP problem). In addition, once a strategy λ_n is fixed for the MDP, Algorithm LUMP is applied on the induced MC in order to reduce its size and to produce its bisimulation quotient. The system of linear equations is then solved for the quotient, and the computed value functions are used to improve the strategy for each individual state of the MDP.

Algorithm 5 SYMBLICIT(MDP $\mathcal{M}^{\mathcal{S}}$, [Strictly negative] weight function $w^{\mathcal{S}}$[, Goal $G^{\mathcal{S}}$])

1: $n := 0, \lambda_n^{\mathcal{S}} :=$ INITIALSTRATEGY($\mathcal{M}^{\mathcal{S}}$[, $G^{\mathcal{S}}$])
2: **repeat**
3: $\quad(\mathcal{M}_{\lambda_n}^{\mathcal{S}}, w_{\lambda_n}^{\mathcal{S}}) :=$ INDUCEDMCANDWEIGHT($\mathcal{M}^{\mathcal{S}}, w^{\mathcal{S}}, \lambda_n^{\mathcal{S}}$)
4: $\quad(\mathcal{M}_{\lambda_n,\sim_L}^{\mathcal{S}}, w_{\lambda_n,\sim_L}^{\mathcal{S}}) :=$ LUMP($\mathcal{M}_{\lambda_n}^{\mathcal{S}}, w_{\lambda_n}^{\mathcal{S}}$)
5: $\quad(\mathcal{M}_{\lambda_n,\sim_L}, w_{\lambda_n,\sim_L}) :=$ EXPLICIT($\mathcal{M}_{\lambda_n,\sim_L}^{\mathcal{S}}, w_{\lambda_n,\sim_L}^{\mathcal{S}}$)
6: $\quad x_n :=$ SOLVELINEARSYSTEM($\mathcal{M}_{\lambda_n,\sim_L}, w_{\lambda_n,\sim_L}$)
7: $\quad x_n^{\mathcal{S}} :=$ SYMBOLIC(x_n)
8: $\quad\lambda_{n+1}^{\mathcal{S}} :=$ IMPROVESTRATEGY($\mathcal{M}^{\mathcal{S}}, \lambda_n^{\mathcal{S}}, x_n^{\mathcal{S}}$)
9: $\quad n := n + 1$
10: **until** $\lambda_n^{\mathcal{S}} = \lambda_{n-1}^{\mathcal{S}}$
11: return $(\lambda_{n-1}^{\mathcal{S}}, x_{n-1}^{\mathcal{S}})$

The symblicit algorithm is described in Algorithm SYMBLICIT (see Algorithm 5). Note that in line 1, the initial strategy λ_0 is selected arbitrarily for the EMP, while it has to be a proper strategy in case of SSP. It combines symbolic and explicit representations of data manipulated by the underlying algorithm. Symbolic representations are denoted by $\mathcal{S}$ in superscript. The algorithm works as follows. The MDP $\mathcal{M}^{\mathcal{S}}$, the weight function $w^{\mathcal{S}}$, the strategies $\lambda_n^{\mathcal{S}}$, the induced MCs $\mathcal{M}_{\lambda_n}^{\mathcal{S}}$ with weight functions $w_{\lambda_n}^{\mathcal{S}}$, and the set $G^{\mathcal{S}}$ of goal states for the SSP, are symbolically represented. Therefore, the lumping procedure is applied on symbolic MCs and produces a symbolic representation of the bisimulation quotient $\mathcal{M}_{\lambda_n,\sim_L}^{\mathcal{S}}$ and associated weight function $w_{\lambda_n,\sim_L}^{\mathcal{S}}$ (line 4). However, since

solving linear systems is more efficient using an explicit representation of the transition matrix, the computed bisimulation quotient is converted to a sparse matrix representation (line 5). The quotient being in general much smaller than the original model, there is no memory issue by storing it explicitly. The linear system is thus solved on the explicit quotient. The computed value functions x_n (corresponding to v_n for the SSP, and g_n and b_n for the EMP) are then converted into symbolic representations x_n^S, and transferred back to the original MDP (line 7). Finally, the update of the strategy is performed symbolically.

The correctness of Algorithm 5 (especially the use of the bisimulation lumping technique) is proved in [WBB+10] for the EMP problem (see Lemma 7.8). Note that the SSP problem is a special case of EMP where the linear system is limited to the second equation of Algorithm 3 with $g_n(s) = 0$ for all $s \in S$. The justification in the case of the SSP problem is thus a direct corollary of Lemma 7.8.

Lemma 7.8 ([WBB+10]). *Let $\mathcal{M} = (S, \mathcal{P})$ be an* MC *and let $\mathcal{M}_{\sim_L} = (S_{\sim_L}, \mathcal{P}_{\sim_L})$ be its quotient induced by $\sim_L$. Let $g_{\sim_L} : S_{\sim_L} \to \mathbb{R}$ and $b_{\sim_L} : S_{\sim_L} \to \mathbb{R}$ be such that the equations of Algorithm 3 are fulfilled in $\mathcal{M}_{\sim_L}$. Then, the gain and bias values $g : S \to \mathbb{R}$ and $b : S \to \mathbb{R}$ with $g(s) = g_{\sim_L}(C)$ and $b(s) = b_{\sim_L}(C)$ where $s \in C$ and $C \in S_{\sim_L}$ also fulfill the given equations.*

In [WBB+10], the intermediate symbolic representations use (MT)BDDs. In the next section, we show how pseudo-antichains can be used (instead of (MT)BDDs) to solve the SSP and EMP problems for monotonic MDPs under well-chosen assumptions.

7.4 Pseudo-antichain based algorithms

In this section, we propose a pseudo-antichain based version of the symblicit algorithm described in Section 7.3 for solving the SSP and EMP problems for monotonic MDPs. In our approach, equivalence relations and induced partitions are symbolically represented so that each block is PA-represented. The efficiency of this approach is thus directly linked to the number of blocks to represent, which explains why our algorithm always works with coarsest equivalence relations. It is also linked to the size of the pseudo-antichains representing the blocks of the partitions.

7.4.1 Operator $\mathsf{Pre}_{\sigma,\tau}$

We begin by presenting an operator denoted $\mathsf{Pre}_{\sigma,\tau}$ that is very useful for our algorithms. Let $\mathcal{M}_{\preceq} = (S, \Sigma, T, \mathcal{E}, \mathcal{D})$ be a monotonic MDP equipped with a weight function $w : S \times \Sigma \to \mathbb{R}$. Given $L \subseteq S$, $\sigma \in \Sigma$ and $\tau \in T$, we denote by $\mathsf{Pre}_{\sigma,\tau}(L)$ the set of states that reach L by σ, τ in $\mathcal{M}_{\preceq}$, that is

$$\mathsf{Pre}_{\sigma,\tau}(L) = \{s \in S \mid \mathcal{E}(s,\sigma)(\tau) \in L\}.$$

The elements of $\mathsf{Pre}_{\sigma,\tau}(L)$ are called *predecessors* of L for σ, τ in $\mathcal{M}_{\preceq}$. The following lemma is a direct consequence of the compatibility of $\preceq$.

Lemma 7.9. *For all closed set $L \subseteq S$, and all actions $\sigma \in \Sigma, \tau \in T$, $\mathsf{Pre}_{\sigma,\tau}(L)$ is closed.*

The next lemma indicates the behavior of the $\mathsf{Pre}_{\sigma,\tau}$ operator under boolean operations. The second and last properties follow from the fact that $\mathcal{M}_{\preceq}$ is deterministic.

Lemma 7.10. *Let $L_1, L_2 \subseteq S$, $\sigma \in \Sigma$ and $\tau \in T$. Then,*

- $\mathsf{Pre}_{\sigma,\tau}(L_1 \cup L_2) = \mathsf{Pre}_{\sigma,\tau}(L_1) \cup \mathsf{Pre}_{\sigma,\tau}(L_2)$
- $\mathsf{Pre}_{\sigma,\tau}(L_1 \cap L_2) = \mathsf{Pre}_{\sigma,\tau}(L_1) \cap \mathsf{Pre}_{\sigma,\tau}(L_2)$
- $\mathsf{Pre}_{\sigma,\tau}(L_1 \setminus L_2) = \mathsf{Pre}_{\sigma,\tau}(L_1) \setminus \mathsf{Pre}_{\sigma,\tau}(L_2)$

Proof. The first property is immediate. We only prove the second property, since the arguments are similar for the last one. Let $s \in \mathsf{Pre}_{\sigma,\tau}(L_1 \cap L_2)$, i.e. $\exists s' \in L_1 \cap L_2$ such that $\mathcal{E}(s,\sigma)(\tau) = s'$. We thus have $s \in \mathsf{Pre}_{\sigma,\tau}(L_1)$ and $s \in \mathsf{Pre}_{\sigma,\tau}(L_2)$. Conversely let $s \in \mathsf{Pre}_{\sigma,\tau}(L_1) \cap \mathsf{Pre}_{\sigma,\tau}(L_2)$, i.e. $\exists s_1 \in L_1$ such that $\mathcal{E}(s,\sigma)(\tau) = s_1$, and $\exists s_2 \in L_2$ such that $\mathcal{E}(s,\sigma)(\tau) = s_2$. As $\mathcal{M}_{\preceq}$ is deterministic, we have $s_1 = s_2$ and thus $s \in \mathsf{Pre}_{\sigma,\tau}(L_1 \cap L_2)$. □

The next proposition indicates how to compute pseudo-antichains with respect to the $\mathsf{Pre}_{\sigma,\tau}$ operator.

Proposition 7.11. *Let (x, α) be a pseudo-element with $x \in S$ and $\alpha \subseteq S$. Let $A = \{(x_i, \alpha_i) \mid i \in I\}$ be a pseudo-antichain with $x_i \in S$ and $\alpha_i \subseteq S$ for all $i \in I$. Then, for all $\sigma \in \Sigma$ and $\tau \in T$,*

- $\mathsf{Pre}_{\sigma,\tau}(\updownarrow(x,\alpha)) = \bigcup_{x' \in \lceil \mathsf{Pre}_{\sigma,\tau}(\downarrow\{x\}) \rceil} \updownarrow(x', \lceil \mathsf{Pre}_{\sigma,\tau}(\downarrow\alpha) \rceil)$
- $\mathsf{Pre}_{\sigma,\tau}(\updownarrow A) = \bigcup_{i \in I} \mathsf{Pre}_{\sigma,\tau}(\updownarrow(x_i, \alpha_i))$

Proof. For the first statement, we have $\mathsf{Pre}_{\sigma,\tau}(\updownarrow(x,\alpha)) = \mathsf{Pre}_{\sigma,\tau}(\downarrow\{x\} \setminus \downarrow\alpha) = \mathsf{Pre}_{\sigma,\tau}(\downarrow\{x\}) \setminus \mathsf{Pre}_{\sigma,\tau}(\downarrow\alpha)$ by definition of the pseudo-closure and by Lemma 7.10. The sets $\mathsf{Pre}_{\sigma,\tau}(\downarrow\{x\})$ and $\mathsf{Pre}_{\sigma,\tau}(\downarrow\alpha)$ are closed by Lemma 7.9 and thus respectively represented by the antichains $\lceil \mathsf{Pre}_{\sigma,\tau}(\downarrow\{x\}) \rceil$ and $\lceil \mathsf{Pre}_{\sigma,\tau}(\downarrow\alpha) \rceil$. By Lemma 4.8 we get the first statement.

The second statement is a direct consequence of Lemma 7.10. □

From Proposition 7.11, we can efficiently compute pseudo-antichains with respect to the $\mathsf{Pre}_{\sigma,\tau}$ operator if we have an efficient algorithm to compute antichains with respect to the $\mathsf{Pre}_{\sigma,\tau}$ operator. We make the following assumption that we can compute the predecessors of a closed set by only considering the antichain of its maximal elements. Together with Proposition 7.11, it implies that the computation of $\mathsf{Pre}_{\sigma,\tau}(\updownarrow A)$, for all pseudo-antichains A, does not need to treat the whole pseudo-closure $\updownarrow A$.

Assumption 7.12. There exists an algorithm taking any state $x \in S$ as input and returning $\lceil \mathsf{Pre}_{\sigma,\tau}(\downarrow\{x\}) \rceil$ as output.

Remark 7.13. Assumption 7.12 is a realistic and natural assumption when considering partially ordered state spaces. For instance, it holds for the two classes of MDPs considered in Chapter 8 for which the given algorithm is straightforward. Assumptions in the same flavor are made in [FS01] (see Definition 3.2). ◁

7.4.2 Symbolic representations

Before giving a pseudo-antichain based algorithm for the symblicit approach of Section 7.3 (see Algorithm 5), we detail in this section the kind of symbolic representations based on pseudo-antichains that we are going to use. Recall from Section 4.2 that PA-representations are not unique. For efficiency reasons, it will be necessary to work with PA-representations that are as *compact* as possible as suggested in the sequel (and by considering pseudo-elements in canonical form and pseudo-antichains in simplified form, as explained in Section 4.2).

Representation of the stochastic models. We begin with symbolic representations for the monotonic MDP $\mathcal{M}_{\preceq} = (S, \Sigma, T, \mathcal{E}, \mathcal{D})$ and for the MC $\mathcal{M}_{\preceq,\lambda} = (S, \Sigma, \mathcal{E}_\lambda, \mathcal{D}_\lambda)$ induced by a strategy λ. For algorithmic purposes, in addition to Assumption 7.12, we make the following assumption[6] on $\mathcal{M}_{\preceq}$.

Assumption 7.14. There exists an algorithm taking as input any state $s \in S$ and actions $\sigma \in \Sigma_s, \tau \in T$, and returning as output $\mathcal{E}(s,\sigma)(\tau)$ and $\mathcal{D}(s,\sigma)(\tau)$.

By definition of $\mathcal{M}_{\preceq}$, the set S of states is closed for $\preceq$ and can thus be canonically represented by the antichain $\lceil S \rceil$, and thus represented by the pseudo-antichain $\{(x, \emptyset) \mid x \in \lceil S \rceil\}$. In this way, it follows by Assumption 7.14 that we have a PA-representation of $\mathcal{M}_{\preceq}$, in the sense that S is PA-represented, and we can compute $\mathcal{E}(s,\sigma)(\tau)$ and $\mathcal{D}(s,\sigma)(\tau)$ on demand, that is, there is no need of storing $\mathcal{M}_{\preceq}$ explicitly.

Let $\lambda : S \to \Sigma$ be a strategy on $\mathcal{M}_{\preceq}$ and $\mathcal{M}_{\preceq,\lambda}$ be the induced MC with weight function w_λ. We denote by $\sim_\lambda$ the equivalence relation on S such that $s \sim_\lambda s'$ if and only if $\lambda(s) = \lambda(s')$. We denote by $S_{\sim_\lambda}$ the induced partition of S. Given a block $B \in S_{\sim_\lambda}$, we denote by $\lambda(B)$ the unique action $\lambda(s)$, for all $s \in B$. As any set can be represented by a pseudo-antichain, each block of $S_{\sim_\lambda}$ is PA-represented. Therefore by Assumption 7.14, we have a PA-representation of $\mathcal{M}_{\preceq,\lambda}$.

Representation of a subset of goal states. Recall that a subset $G \subseteq S$ of goal states is required for the SSP problem. Our algorithm will manipulate G when computing the set of proper states. A natural assumption is to require that G is *closed* (like S), as it is the case for the two classes of monotonic MDPs studied in Chapter 8. Under this assumption, we have a compact representation of G as the one proposed above for S. Otherwise, we take for G any PA-representation.

Representation for $\mathcal{D}$ and w. For the needs of our algorithm, we introduce symbolic representations for $\mathcal{D}_\lambda$ and w_λ. Similarly to $\sim_\lambda$, let $\sim_{\mathcal{D},\lambda}$ be the equivalence relation on S such that $s \sim_{\mathcal{D},\lambda} s'$ if and only if $\mathcal{D}_\lambda(s) = \mathcal{D}_\lambda(s')$. We denote by $S_{\sim_{\mathcal{D},\lambda}}$ the induced partition of S. Given a block $B \in S_{\sim_{\mathcal{D},\lambda}}$, we denote

[6] Remark 7.13 also holds for Assumption 7.14.

by $\mathcal{D}_\lambda(B)$ the unique probability distribution $\mathcal{D}_\lambda(s)$, for all $s \in B$. We use similar notations for the equivalence relation $\sim_{w,\lambda}$ on S such that $s \sim_{w,\lambda} s'$ if and only if $w_\lambda(s) = w_\lambda(s')$. As any set can be represented by a pseudo-antichain, each block of $S_{\sim_{\mathcal{D},\lambda}}$ and $S_{\sim_{w,\lambda}}$ is PA-represented.

We will also need to use the next two equivalence relations. For each $\sigma \in \Sigma$, we introduce the equivalence relation $\sim_{\mathcal{D},\sigma}$ on S such that $s \sim_{\mathcal{D},\sigma} s'$ if and only if $\mathcal{D}(s,\sigma) = \mathcal{D}(s',\sigma)$. Similarly, we introduce relation $\sim_{w,\sigma}$ such that $s \sim_{w,\sigma} s'$ if and only if $w(s,\sigma) = w(s',\sigma)$. Recall that $\mathcal{D}$ and w are partial functions, there may thus exist one block in their corresponding relation gathering all states s such that $\sigma \notin \Sigma_s$. Each block of the induced partitions $S_{\sim_{\mathcal{D},\sigma}}$ and $S_{\sim_{w,\sigma}}$ is PA-represented.

For the two classes of MDPs studied in Chapter 8, both functions $\mathcal{D}$ and w are independent of S. It follows that the previous equivalence relations have only one or two blocks, leading to compact symbolic representations of these relations.

Now that the operator $\mathsf{Pre}_{\sigma,\tau}$ and the used symbolic representations have been introduced, we come back to the steps of the symblicit approach of Section 7.3 (see Algorithm 5) and show how to derive a pseudo-antichain based algorithm. We will use Propositions 4.2, 4.9 and 7.11, and Assumptions 7.12 and 7.14, for which we know that boolean and $\mathsf{Pre}_{\sigma,\tau}$ operations can be performed efficiently on pseudo-closures of pseudo-antichains, by limiting the computations to the related pseudo-antichains. Whenever possible, we will work with partitions with few blocks whose PA-representation is compact. This aim will be reached for the two classes of monotonic MDPs studied in Chapter 8.

7.4.3 Initial strategy

Algorithm 5 needs an initial strategy λ_0 (line 1). This strategy can be selected arbitrarily among the set of strategies for the EMP problem, while it has to be a proper strategy for the SSP problem. We detail how to choose the initial strategy in these two quantitative settings.

Expected mean-payoff. For the EMP problem, we propose an arbitrary initial strategy λ_0 with a compact PA-representation for the induced MC $M_{\preceq,\lambda_0}$.

We know that S is PA-represented by $\{(x, \emptyset) \mid x \in \lceil S \rceil\}$, and that for all $s, s' \in S$ such that $s \preceq s'$, we have $\Sigma_{s'} \subseteq \Sigma_s$ (Proposition 7.3). This means that for $x \in \lceil S \rceil$ and $\sigma \in \Sigma_x$ we could choose $\lambda_0(s) = \sigma$ for all $s \in \downarrow\{x\}$. However we must be careful with states that belong to $\downarrow\{x\} \cap \downarrow\{x'\}$ with $x, x' \in \lceil S \rceil$, $x \neq x'$. Therefore, let us impose an arbitrary total ordering on $\lceil S \rceil = \{x_1, \ldots, x_n\}$, i.e. $x_1 < x_2 < \ldots < x_n$. We then define λ_0 arbitrarily on $\lceil S \rceil$ such that $\lambda_0(x_i) = \sigma_i$ for some $\sigma_i \in \Sigma_{x_i}$, and we extend it to all $s \in S$ by $\lambda_0(s) = \lambda_0(x)$ with $x = \min_i\{x_i \mid s \preceq x_i\}$. This makes sense looking at the previous remarks. Notice that given $\sigma \in \Sigma$, the block B of the partition $S_{\sim_{\lambda_0}}$ such that $\lambda_0(B) = \sigma$ is PA-represented by $\bigcup_i \{(x_i, \alpha_i) \mid \alpha_i = \{x_1, \ldots, x_{i-1}\}, \lambda_0(x_i) = \sigma\}$.

Proper states. Before explaining how to compute an initial proper strategy λ_0 for the SSP problem, we need to propose a pseudo-antichain based version of the algorithm of [dA99] for computing the set S^{P} of proper states. Recall from Section 7.2.1 that this algorithm is required for solving the SSP problem.

Let $\mathcal{M}_\preceq$ be a monotonic MDP and G be a set of goal states. Recall that S^{P} is computed as $S^{\mathsf{P}} = \nu Y \cdot \mu X \cdot (\mathsf{APre}(Y, X) \vee \mathsf{G})$, such that for all states s,

$$s \models \mathsf{APre}(Y, X) \Leftrightarrow \exists \sigma \in \Sigma_s, (\mathsf{succ}(s, \sigma) \subseteq Y \wedge \mathsf{succ}(s, \sigma) \cap X \neq \emptyset).$$

Our purpose is to define the set of states satisfying $\mathsf{APre}(Y, X)$ thanks to the operator $\mathsf{Pre}_{\sigma,\tau}$. The difficulty is to limit the computations to strictly positive probabilities as required by the operator succ. To this end, given the equivalence relation $\sim_{\mathcal{D},\sigma}$ defined in Section 7.4.2, for each $\sigma \in \Sigma$ and $\tau \in T$, we define $\mathcal{D}^{>0}_{\sigma,\tau}$ being the set of blocks $\{D \in S_{\sim_{\mathcal{D},\sigma}} \mid \mathcal{D}(s, \sigma)(\tau) > 0 \text{ with } s \in D\}$. For each $D \in \mathcal{D}^{>0}_{\sigma,\tau}$, notice that $\sigma \in \Sigma_s$ for all $s \in D$ (since $\mathcal{D}(s, \sigma)$ is defined). Given two sets $X, Y \subseteq S$, the set of states satisfying $\mathsf{APre}(Y, X)$ is equal to:

$$R(Y, X) = \bigcup_{\sigma \in \Sigma} \bigcup_{D \in S_{\sim_{\mathcal{D},\sigma}}} \Big(\bigcap_{\substack{\tau \in T \\ D \in \mathcal{D}^{>0}_{\sigma,\tau}}} (\mathsf{Pre}_{\sigma,\tau}(Y) \cap D) \cap \bigcup_{\substack{\tau \in T \\ D \in \mathcal{D}^{>0}_{\sigma,\tau}}} (\mathsf{Pre}_{\sigma,\tau}(X) \cap D) \Big)$$

Lemma 7.15. *For all $X, Y \subseteq S$ and $s \in S$, $s \models \mathsf{APre}(Y, X) \Leftrightarrow s \in R(Y, X)$.*

Proof. Let $s \models \mathsf{APre}(Y, X)$. Then there exists $\sigma \in \Sigma_s$ such that $\mathsf{succ}(s, \sigma) \subseteq Y$ and $\mathsf{succ}(s, \sigma) \cap X \neq \emptyset$. Let $D \in S_{\sim_{\mathcal{D},\sigma}}$ be such that $s \in D$. Let us prove that $s \in \bigcap_{\tau \in T, D \in \mathcal{D}^{>0}_{\sigma,\tau}} \mathsf{Pre}_{\sigma,\tau}(Y)$ and $s \in \bigcup_{\tau \in T, D \in \mathcal{D}^{>0}_{\sigma,\tau}} \mathsf{Pre}_{\sigma,\tau}(X)$. It will follows that

$s \in R(Y, X)$. As $\mathsf{succ}(s, \sigma) \cap X \neq \emptyset$, there exists $x \in X$ such that $\mathcal{P}(s, \sigma, x) > 0$, that is, $\mathcal{E}(s, \sigma)(\tau) = x$ and $\mathcal{D}(s, \sigma)(\tau) > 0$ for some $\tau \in T$. Thus $s \in \mathsf{Pre}_{\sigma,\tau}(X)$ and $D \in \mathcal{D}^{>0}_{\sigma,\tau}$. As $\mathsf{succ}(s, \sigma) \subseteq Y$, then for all s' such that $\mathcal{P}(s, \sigma, s') > 0$, we have $s' \in Y$, or equivalently, for all $\tau \in T$ such that $\mathcal{D}(s, \sigma)(\tau) > 0$, we have $\mathcal{E}(s, \sigma)(\tau) = s' \in Y$. Therefore, $s \in \mathsf{Pre}_{\sigma,\tau}(Y)$ for all τ such that $D \in \mathcal{D}^{>0}_{\sigma,\tau}$.

Suppose now that $s \in R(Y, X)$. Then, there exists $\sigma \in \Sigma$ and $D \in S_{\sim_{\mathcal{D},\sigma}}$ such that $s \in \bigcap_{\tau \in T, D \in \mathcal{D}^{>0}_{\sigma,\tau}} \mathsf{Pre}_{\sigma,\tau}(Y)$ and $s \in \bigcup_{\tau \in T, D \in \mathcal{D}^{>0}_{\sigma,\tau}} \mathsf{Pre}_{\sigma,\tau}(X)$. With the same arguments as above, we deduce that $\mathsf{succ}(s, \sigma) \subseteq Y$ and $\mathsf{succ}(s, \sigma) \cap X \neq \emptyset$. Notice that we have $\sigma \in \Sigma_s$. It follows that $s \models \mathsf{APre}(Y, X)$. □

In the case of the MDPs treated in Chapter 8, G is closed and $\mathcal{D}^{>0}_{\sigma,\tau}$ is composed of at most one block D that is closed. It follows that all the intermediate sets manipulated by the algorithm computing S^{p} are closed. We thus have an efficient algorithm since it can be based on antichains only.

Stochastic shortest path. For the SSP problem, the initial strategy λ_0 must be proper. In the previous paragraph, we have presented an algorithm for computing the set $S^{\mathsf{p}} = \nu Y \cdot \mu X \cdot (\mathsf{APre}(Y, X) \vee \mathsf{G})$ of proper states. One can directly extract a PA-representation of a proper strategy from the execution of this algorithm, as follows. During the greatest fix point computation, a sequence $Y_0, Y_1, \ldots, Y_k$ is computed such that $Y_0 = S$ and $Y_{k-1} = Y_k = S^{\mathsf{p}}$. During the computation of Y_k, a least fix point computation is performed, leading to a sequence $X_0, X_1, \ldots, X_l$ such that $X_0 = G$ and $X_i = X_{i-1} \cup \mathsf{APre}(S^{\mathsf{p}}, X_{i-1})$ for all i, $1 \leq i \leq l$, and $X_l = Y_k$. We define a strategy λ_0 incrementally with each set X_i. Initially, $\lambda_0(s)$ is any $\sigma \in \Sigma_s$ for each $s \in X_0$ (since G is reached). When X_i has been computed, then for each $s \in X_i \setminus X_{i-1}$, $\lambda_0(s)$ is any $\sigma \in \Sigma_s$ such that $\mathsf{succ}(s, \sigma) \subseteq S^{\mathsf{p}}$ and $\mathsf{succ}(s, \sigma) \cap X_{i-1} \neq \emptyset$. Doing so, each proper state is eventually associated with an action by λ_0. Note that this strategy is proper. Indeed, to simplify the argument, suppose G is absorbing, that is, for all $s \in G$ and $\sigma \in \Sigma_s$, $\sum_{s' \in G} \mathcal{P}(s, \sigma, s') = 1$. Then by construction of λ_0, the bottom strongly connected components of the MC induced by λ_0 are all in G, and a classical result on MCs (see for instance [BK08]) states that any infinite path leads to one of those components with probability 1. Finally, as all sets X_i, Y_j manipulated by the algorithm are PA-represented, we obtain a partition $S_{\sim_{\lambda_0}}$ such that each block $B \in S_{\sim_{\lambda_0}}$ is PA-represented.

Algorithm 6 SPLIT(B, C, λ)

1: $P[0] := B$
2: **for** i in $[1, m]$ **do**
3: $\quad P_{\text{new}} :=$ INITTABLE(P, τ_i)
4: $\quad$ **for all** $(p, block)$ in P **do**
5: $\quad\quad P_{\text{new}}[p] := P_{\text{new}}[p] \cup (block \setminus \mathsf{Pre}_\lambda(C, \tau_i))$
6: $\quad\quad$ **for all** $D \in S_{\sim_{\mathcal{D},\lambda}}$ **do**
7: $\quad\quad\quad P_{\text{new}}[p + \mathcal{D}_\lambda(D)(\tau_i)] := P_{\text{new}}[p + \mathcal{D}_\lambda(D)(\tau_i)] \cup (block \cap D \cap \mathsf{Pre}_\lambda(C, \tau_i))$
8: $\quad P :=$ REMOVEEMPTYBLOCKS(P_{new})
9: return P

7.4.4 Bisimulation lumping

We now consider the step of Algorithm 5 where Algorithm LUMP is called to compute the largest bisimulation $\sim_L$ of an MC $\mathcal{M}_{\preceq,\lambda}$ induced by a strategy λ on $\mathcal{M}_{\preceq}$ (line 4 with $\lambda = \lambda_n$). We here detail a pseudo-antichain based version of Algorithm LUMP (see Algorithm 4) when $\mathcal{M}_{\preceq,\lambda}$ is PA-represented. Recall that if $\mathcal{M}_{\preceq} = (S, \Sigma, T, \mathcal{E}, \mathcal{D})$, then $\mathcal{M}_{\preceq,\lambda} = (S, T, \mathcal{E}_\lambda, \mathcal{D}_\lambda)$. Equivalently, using the usual definition of an MC, $\mathcal{M}_{\preceq,\lambda} = (S, \mathcal{P}_\lambda)$ with $\mathcal{P}_\lambda$ derived from $\mathcal{E}_\lambda$ and $\mathcal{D}_\lambda$ (see Section 7.1). Remember also the two equivalence relations $\sim_{\mathcal{D},\lambda}$ and $\sim_{w,\lambda}$ defined in Section 7.4.2.

The initial partition P computed by Algorithm 4 (line 1) is such that for all $s, s' \in S$, s and s' belong to the same block of P if and only if $w_\lambda(s) = w_\lambda(s')$. The initial partition P is thus $S_{\sim_{w,\lambda}}$.

Algorithm 4 needs to split blocks of partition P (line 7). This can be performed with Algorithm SPLIT that we are going to describe (see Algorithm 6). Given two blocks $B, C \subseteq S$, this algorithm splits B into a partition P composed of sub-blocks $B_1, \ldots, B_k$ according to the probability of reaching C in $\mathcal{M}_{\preceq,\lambda}$, i.e. for all $s, s' \in B$, we have $s, s' \in B_l$ for some l if and only if $\mathcal{P}_\lambda(s, C) = \mathcal{P}_\lambda(s', C)$.

Suppose that $T = \{\tau_1, \ldots, \tau_m\}$. This algorithm computes intermediate partitions P of B such that at step i, B is split according to the probability of reaching C in $\mathcal{M}_{\preceq,\lambda}$ when T is restricted to $\{\tau_1, \ldots, \tau_i\}$. To perform this task, it needs a new operator Pre_λ based on $\mathcal{M}_{\preceq}$ and λ. Given $L \subseteq S$ and $\tau \in T$, we define

$$\mathsf{Pre}_\lambda(L, \tau) = \{s \in S \mid \mathcal{E}_\lambda(s)(\tau) \in L\}$$

as the set of states from which L is reached by τ in $\mathcal{M}_{\preceq}$ under the selection made by λ. Notice that when T is restricted to $\{\tau_1, \ldots, \tau_i\}$ with $i < m$, it may happen that $\mathcal{D}_\lambda(s)$ is no longer a probability distribution for some $s \in S$ (when $\sum_{\tau \in \{\tau_1,\ldots,\tau_i\}} \mathcal{D}_\lambda(s)(\tau) < 1$).

Initially, T is restricted to $\emptyset$, and the partition P is composed of one block B (line 1). At step i with $i \geq 1$, each block B_l of the partition computed at step $i-1$ is split into several sub-blocks according to its intersection with $\mathsf{Pre}_\lambda(C, \tau_i)$ and each $D \in S_{\sim_{\mathcal{D},\lambda}}$. We take into account intersections with $D \in S_{\sim_{\mathcal{D},\lambda}}$ in a way to know which probability distribution $\mathcal{D}_\lambda(D)$ is associated with the states we are considering. Suppose that at step $i-1$ the probability for any state of block B_l of reaching C is p. Then at step i, it is equal to $p + \mathcal{D}_\lambda(D)(\tau_i)$ if this state belongs to $D \cap \mathsf{Pre}_\lambda(C, \tau_i)$, with $D \in S_{\sim_{\mathcal{D},\lambda}}$, and to p if it does not belong to $\mathsf{Pre}_\lambda(C, \tau_i)$ (lines 5-7). See Figure 7.5 for intuition. Notice that some newly created sub-blocks could have the same probability, they are therefore merged.

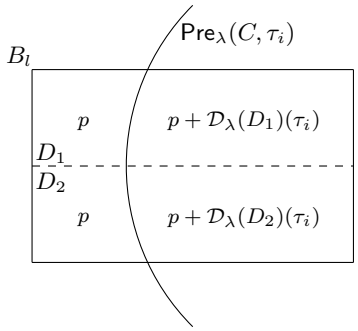

Figure 7.5: Step i of Algorithm 6 on a block B_l.

The intermediate partitions P (or P_{new}) manipulated by the algorithm are represented by hash tables: each entry (p, block) is stored as $P[p] = \mathit{block}$ such that block is the set of states that reach C with probability p. The use of hash tables permits to efficiently gather sub-blocks of states having the same probability of reaching C, and thus to keep minimal the number of blocks in the partition. Algorithm InitTable is used to initialize a new partition P_{new} from a previous partition P and symbol τ_i: the new hash table is initialized with $P_{\text{new}}[p] := \emptyset$ and $P_{\text{new}}[p + \mathcal{D}_\lambda(D)(\tau_i)] := \emptyset$, for all $D \in S_{\sim_{\mathcal{D},\lambda}}$ and all (p, block) in P. Algorithm RemoveEmptyBlocks(P) removes from the hash table P each pair (p, block) such that $\mathit{block} = \emptyset$.

Theorem 7.16. *Let λ be a strategy on $\mathcal{M}_{\preceq}$ and $\mathcal{M}_{\preceq,\lambda} = (S, \mathcal{P}_\lambda)$ be the induced* MC*. Let $B, C \subseteq S$ be two blocks. Then the output of* Split(B, C, λ) *is a partition $\{B_1, \ldots, B_k\}$ of B such that for all $s, s' \in B$, $s, s' \in B_l$ for some l if and only if $\mathcal{P}_\lambda(s, C) = \mathcal{P}_\lambda(s', C)$.*

Proof. The correctness of Algorithm Split is based on the following invariant. At step i, $0 \leq i \leq m$, with T restricted to $\{\tau_1, \ldots, \tau_i\}$, we have:

- P is a partition of B
- $\forall s \in B$, if $s \in P[p]$, then $\mathcal{P}_\lambda(s, C) = p$

Let us first prove that P is a partition. Note that the use of algorithm RemoveEmptyBlocks ensures that P never contains an empty block. Recall that $S_{\sim_{\mathcal{D},\lambda}}$ is a partition of S.

Initially, when $i = 0$, P is composed of the unique block B. Let $i \geq 1$ and suppose that $P = \{B_1, \ldots, B_k\}$ is a partition of B at step $i - 1$, and let us prove that P_{new} is a partition of B (at line 8 of the algorithm). Each $B_l \in P$ is partitioned as $\{B_l \setminus \mathsf{Pre}_\lambda(C, \tau_i)\} \cup \{B_l \cap D \cap \mathsf{Pre}_\lambda(C, \tau_i) \mid D \in S_{\sim_{\mathcal{D},\lambda}}\}$. This leads to the finer partition $P' = \{B_l \setminus \mathsf{Pre}_\lambda(C, \tau_i) \mid B_l \in P\} \cup \{B_l \cap D \cap \mathsf{Pre}_\lambda(C, \tau_i) \mid B_l \in P, D \in S_{\sim_{\mathcal{D},\lambda}}\}$ of P. Some blocks of P' are gathered by the algorithm to get P_{new} which is thus a partition.

Let us now prove that at each step i, with T restricted to $\{\tau_1, \ldots, \tau_i\}$, we have: $\forall s \in B$, if $s \in P[p]$, then $\mathcal{P}_\lambda(s, C) = p$.

Initially, when $i = 0$, T is restricted to $\emptyset$, and thus $\mathcal{P}_\lambda(s, C) = 0$ for all $s \in B$. Let $i \geq 1$ and suppose we have that $\forall s \in B$, if $s \in P[p]$ for some p, then $\mathcal{P}_\lambda(s, C) = p$, when T is restricted to $\{\tau_1, \ldots, \tau_{i-1}\}$. Let us prove that if $s \in P_{\text{new}}[p]$ for some p, then $\mathcal{P}_\lambda(s, C) = p$, when T is restricted to $\{\tau_1, \ldots, \tau_i\}$. Let $s \in B$ be such that $s \in P_{\text{new}}[p]$, we identify two cases: either (i) $s \in P[p]$ and $s \notin \mathsf{Pre}_\lambda(C, \tau_i)$, or (ii) $s \in P[p - \mathcal{D}_\lambda(D)(\tau_i)]$ and $s \in D \cap \mathsf{Pre}_\lambda(C, \tau_i)$, for some $D \in S_{\sim_{\mathcal{D},\lambda}}$. In case (i), by induction hypothesis, we know that $\mathcal{P}_\lambda(s, C) = p$, when $T = \{\tau_1, \ldots, \tau_{i-1}\}$, and since $s \notin \mathsf{Pre}_\lambda(C, \tau_i)$, adding τ_i to T does not change the probability of reaching C from s, i.e. $\mathcal{P}_\lambda(s, C) = p$ when $T = \{\tau_1, \ldots, \tau_i\}$. In case (ii), by induction hypothesis, we know that $\mathcal{P}_\lambda(s, C) = p - \mathcal{D}_\lambda(D)(\tau_i)$, when $T = \{\tau_1, \ldots, \tau_{i-1}\}$. Moreover, since $s \in D \cap \mathsf{Pre}_\lambda(C, \tau_i)$, we have $\mathcal{P}_\lambda(s, C) = \mathcal{D}_\lambda(D)(\tau_i)$, when $T = \{\tau_i\}$. It follows that $\mathcal{P}_\lambda(s, C) = p - \mathcal{D}_\lambda(D)(\tau_i) + \mathcal{D}_\lambda(D)(\tau_i) = p$, when $T = \{\tau_1, \ldots, \tau_i\}$.

Finally, after step $i = m$, we get the statement of Theorem 7.16 since P is the final partition of B and for all $s \in B$, the probabilities p being pairwise distinct, we have $s \in P[p]$ if and only if $\mathcal{P}_\lambda(s, C) = p$. □

Notice that we have a pseudo-antichain version of Algorithm Lump as soon as the given blocks B and C are PA-represented. Indeed, this algorithm uses boolean operations and Pre_λ operator. This operator can be computed as follows:

$$\mathsf{Pre}_\lambda(C, \tau) = \bigcup \left\{ \mathsf{Pre}_{\sigma,\tau}(C) \ \cap B \mid \sigma \in \Sigma, B \in S_{\sim_\lambda}, \lambda(B) = \sigma \right\}.$$

Intuitively, let us fix $\sigma \in \Sigma$ and $B \in S_{\sim_\lambda}$ such that $\lambda(B) = \sigma$. Then $\mathsf{Pre}_{\sigma,\tau}(C) \cap B$ is the set of states of $S \cap B$ that reach C in the MDP $\mathcal{M}_{\preceq}$ with σ followed by τ. Finally the union gives the set of states that reach C with τ under the selection made by λ. All these operations can be performed thanks to Propositions 4.9 and 7.11, and Assumptions 7.12 and 7.14.

7.4.5 Solving linear systems

Assume that we have computed the largest bisimulation $\sim_L$ for the MC $\mathcal{M}_{\preceq,\lambda} = (S, \mathcal{P}_\lambda)$. Let us now detail lines 5-7 of Algorithm 5 concerning the system of linear equations that has to be solved. We build the the bisimulation quotient $(S_{\sim_L}, \mathcal{P}_{\lambda,\sim_L})$. We then explicitly solve the linear system of Algorithm 2 for the SSP problem (resp. Algorithm 3 for the EMP problem) (line 3). We thus obtain the expected truncated sum v (resp. the gain value g and bias value b) of the strategy λ, for each block $B \in S_{\sim_L}$. By definition of $\sim_L$, we have that for all $s, s' \in S$, if $s \sim_L s'$, then $v(s) = v(s')$ (resp. $g(s) = g(s')$ and $b(s) = b(s')$). Given a block $B \in S_{\sim_L}$, we denote by $v(B)$ the unique expected truncated sum $v(s)$ (resp. by $g(B)$ the unique gain value $g(s)$ and by $b(B)$ the unique bias value $b(s)$), for all $s \in B$.

7.4.6 Improving strategies

Given an MDP $\mathcal{M}_{\preceq}$ with weight function w and the MC $\mathcal{M}_{\preceq,\lambda}$ induced by a strategy λ, we finally present a pseudo-antichain based algorithm to improve the strategy λ for the SSP problem, with the expected truncated sum v obtained by solving the linear system (see line 8 of Algorithm 5, and Algorithm 2). The

improvement of a strategy for the EMP problem, with the gain g or the bias b values (see Algorithm 3), is similar and is thus not detailed.

Recall that for all $s \in S$, we compute the set $\widehat{\Sigma_s}$ of actions $\sigma \in \Sigma_s$ that maximize the expression $l_\sigma(s) = w(s,\sigma) + \sum_{s' \in S} \mathcal{P}(s,\sigma,s') \cdot v(s')$, and then we improve the strategy based on the computed $\widehat{\Sigma_s}$. We give hereafter an approach based on pseudo-antichains which requires the next two steps. The first step consists in computing, for all $\sigma \in \Sigma$, an equivalence relation $\sim_{l_\sigma}$ such that the value $l_\sigma(s)$ is constant on each block of the relation. The second step uses the relations $\sim_{l_\sigma}$, with $\sigma \in \Sigma$, to improve the strategy.

Computing value l_σ. Let $\sigma \in \Sigma$ be a fixed action. We are looking for an equivalence relation $\sim_{l_\sigma}$ on the set S_σ of states where action σ is enabled, such that

$$\forall s, s' \in S_\sigma : s \sim_{l_\sigma} s' \Rightarrow l_\sigma(s) = l_\sigma(s').$$

Given $\sim_L$ the largest bisimulation for $\mathcal{M}_{\preceq,\lambda}$ and the induced partition $S_{\sim_L}$, we have for each $s \in S_\sigma$

$$l_\sigma(s) = w(s,\sigma) + \sum_{C \in S_{\sim_L}} \mathcal{P}(s,\sigma,C) \cdot v(C)$$

since the value v is constant on each block C. Therefore to get relation $\sim_{l_\sigma}$, it is enough to have $s \sim_{l_\sigma} s' \Rightarrow w(s,\sigma) = w(s',\sigma)$ and $\mathcal{P}(s,\sigma,C) = \mathcal{P}(s',\sigma,C)$, $\forall C \in S_{\sim_L}$. We proceed by defining the following equivalence relations on S_σ. For the weight part, we use relation $\sim_{w,\sigma}$ defined in Section 7.4.2. For the probabilities part, for each block C of $S_{\sim_L}$, we define the relation $\sim_{\mathcal{P},\sigma,C}$ such that $s \sim_{\mathcal{P},\sigma,C} s'$ if and only if $\mathcal{P}(s,\sigma,C) = \mathcal{P}(s',\sigma,C)$. The required relation $\sim_{l_\sigma}$ on S_σ is then defined as the relation

$$\sim_{l_\sigma} \;=\; \sim_{w,\sigma} \cap \bigcap_{C \in S_{\sim_L}} \sim_{\mathcal{P},\sigma,C} \;=\; \sim_{w,\sigma} \cap \sim_{\mathcal{P},\sigma}$$

Let us explain how to compute $\sim_{l_\sigma}$ with a pseudo-antichain based approach. Firstly, $\mathcal{M}_\preceq$ being T-complete, the set S_σ is obtained as $S_\sigma = \mathsf{Pre}_{\sigma,\tau}(S)$ where τ is an arbitrary action of T. Secondly, each relation $\sim_{\mathcal{P},\sigma,C}$ is the output obtained by a call to $\textsc{Split}(S_\sigma, C, \lambda)$ where λ is defined on S_σ by $\lambda(s) = \sigma$ for all $s \in S_\sigma$[7]

[7] As Algorithm Split only works on S_σ, it is not a problem if λ is not defined on $S \setminus S_\sigma$.

(see Algorithm 6). Thirdly, we detail a way to compute $\sim_{\mathcal{P},\sigma}$ from $\sim_{\mathcal{P},\sigma,C}$, for all $C \in S_{\sim_L}$. Let $S_{\sim_{\mathcal{P},\sigma,C}} = \{B_{C,1}, B_{C,2}, \ldots, B_{C,k_C}\}$ be the partition of S_σ induced by $\sim_{\mathcal{P},\sigma,C}$. For each $B_{C,i} \in S_{\sim_{\mathcal{P},\sigma,C}}$, we denote by $\mathcal{P}(B_{C,i}, \sigma, C)$ the unique value $\mathcal{P}(s, \sigma, C)$, for all $s \in B_{C,i}$. Then, computing a block D of $\sim_{\mathcal{P},\sigma}$ consists in picking, for all $C \in S_{\sim_L}$, one block D_C among $B_{C,1}, B_{C,2}, \ldots, B_{C,k_C}$, such that the intersection $D = \bigcap_{C \in S_{\sim_L}} D_C$ is non empty. Recall that, by definition of the MDP $\mathcal{M}_{\preceq}$, we have $\sum_{s' \in S} \mathcal{P}(s, \sigma, s') = 1$. Therefore, if D is non empty, then $\sum_{C \in S_{\sim_L}} \mathcal{P}(D_C, \sigma, C) = 1$. Finally, $\sim_{l_\sigma}$ is obtained as the intersection between $\sim_{w,\sigma}$ and $\sim_{\mathcal{P},\sigma}$.

Relation $\sim_{l_\sigma}$ induces a partition of S_σ that we denote $(S_\sigma)_{\sim_{l_\sigma}}$. For each block $D \in (S_\sigma)_{\sim_{l_\sigma}}$, we denote by $l_\sigma(D)$ the unique value $l_\sigma(s)$, for all $s \in D$.

Improving the strategy. We now propose a pseudo-antichain based algorithm for improving strategy λ by using relations $\sim_L$, $\sim_\lambda$, and $\sim_{l_\sigma}$, $\forall \sigma \in \Sigma$ (see Algorithm 7). We first compute for all $\sigma \in \Sigma$, the equivalence relation $\sim_{l_\sigma \wedge L} = \sim_{l_\sigma} \cap \sim_L$ on S_σ. Given $B \in (S_\sigma)_{\sim_{l_\sigma \wedge L}}$, we denote by $l_\sigma(B)$ the unique value $l_\sigma(s)$ and by $v(B)$ the unique value $v(s)$, for all $s \in B$. Let $\sigma \in \Sigma$, we denote by $(S_\sigma)^{>}_{\sim_{l_\sigma \wedge L}} \subseteq (S_\sigma)_{\sim_{l_\sigma \wedge L}}$ the set of blocks C for which the value $v(C)$ is improved by setting $\lambda(C) = \sigma$, that is

$$(S_\sigma)^{>}_{\sim_{l_\sigma \wedge L}} = \{C \in (S_\sigma)_{\sim_{l_\sigma \wedge L}} \mid l_\sigma(C) > v(C)\}.$$

We then compute an ordered global list $\mathcal{L}$ made of the blocks of all sets $(S_\sigma)^{>}_{\sim_{l_\sigma \wedge L}}$, for all $\sigma \in \Sigma$. It is ordered according to the increasing value $l_\sigma(C)$. In this way, when traversing $\mathcal{L}$, we have more and more promising blocks to increase v.

From input $\mathcal{L}$ and $\sim_\lambda$, Algorithm 7 outputs an equivalence relation $\sim_{\lambda'}$ for a new strategy λ' that improves λ. Given $C \in \mathcal{L}$, suppose that C comes from the relation $\sim_{l_\sigma \wedge L}$ (i.e. σ is considered). Then for each $B \in S_{\sim_\lambda}$ such that $B \cap C \neq \emptyset$ (line 4), we improve the strategy by setting $\lambda'(B \cap C) = \sigma$, while the strategy λ' is kept unchanged for $B \setminus C$. Algorithm 7 outputs a partition $S_{\sim_{\lambda'}}$ such that $s \sim_{\lambda'} s' \Rightarrow \lambda'(s) = \lambda'(s')$ for the improved strategy λ'. If necessary, for efficiency reasons, we can compute a coarser relation for the new strategy λ' by gathering blocks B_1, B_2 of $S_{\sim_{\lambda'}}$, for all B_1, B_2 such that $\lambda'(B_1) = \lambda'(B_2)$.

The correctness of Algorithm 7 is due to the list $\mathcal{L}$, which is sorted according to the increasing value $l_\sigma(C)$. It ensures that the strategy is updated at each

Algorithm 7 ImproveStrategy($\mathcal{L}, S_{\sim\lambda}$)

```
1: for C ∈ 𝓛 do
2:     S_{∼λ'} := ∅
3:     for B ∈ S_{∼λ} do
4:         if B ∩ C ≠ ∅ then
5:             S_{∼λ'} := S_{∼λ'} ∪ {B ∩ C, B \ C}
6:         else
7:             S_{∼λ'} := S_{∼λ'} ∪ B
8:     S_{∼λ} := S_{∼λ'}
9: return S_{∼λ'}
```

state s to an action $\sigma \in \widehat{\Sigma_s}$, i.e. an action σ that maximizes the expression $w(s,\sigma) + \sum_{s' \in S} \mathcal{P}(s,\sigma,s') \cdot v_n(s')$ (cf. line 4 of Algorithm 2).

CHAPTER 8

Applications of the symblicit algorithms

In this chapter, we consider two practical applications of the pseudo-antichain based symblicit algorithm of the previous chapter, one for the SSP problem and the other for the EMP problem. Those applications come from automated planning and $\mathsf{LTL}_{\mathsf{MP}}$ synthesis. In both cases, we first show the reduction to monotonic MDPs equipped with a natural partial order and that satisfy Assumptions 7.12 and 7.14, and we then present some experimental results.

All the experiments presented in this chapter have been done on a Linux platform with a 3.2GHz CPU (Intel Core i7) and 12GB of memory. Note that our implementations are single-threaded and thus use only one core. For all those experiments, the timeout is set to 10 hours and is denoted by **TO**. Finally, we restrict the memory usage to 4GB and when an execution runs out of memory, we denote it by **MO**. Note that restricting the memory usage to 4GB is enough to have a good picture of the behavior of each implementation with respect to the memory consumption.

The content of this chapter is based on [BBR14b, BBR14a].

8.1 Stochastic shortest path on STRIPS

We consider the following application of the pseudo-antichain based symblicit algorithm for the SSP problem. In the field of planning, a class of problems called *planning from STRIPSs* [FN72] operate with states represented by valuations of propositional variables. Informally, a STRIPS is defined by an *initial* state representing the initial configuration of the system and a set of *operators* that transform a state into another state. The problem of planning from STRIPSs then asks, given a valuation of propositional variables representing a set of *goal* states, to find a sequence of operators that lead from the initial state to a goal one. Let us first formally define the notion of STRIPS and show that each STRIPS can be made monotonic. We will then add stochastic aspects and show how to construct a monotonic MDP from a monotonic stochastic STRIPS.

8.1.1 Definitions

STRIPSs. A *STRIPS* [FN72] is a tuple $(P, I, (M, N), O)$ where P is a finite set of *conditions* (i.e. propositional variables), $I \subseteq P$ is a subset of conditions that are initially true (all others are assumed to be false), (M, N), with $M, N \subseteq P$ and $M \cap N = \emptyset$, specifies which conditions are true and false, respectively, in order for a state to be considered as a goal state, and O is a finite set of *operators*. An operator $o \in O$ is a pair $((\gamma, \theta), (\alpha, \delta))$ such that (γ, θ) is the *guard* of o, that is, $\gamma \subseteq P$ (resp. $\theta \subseteq P$) is the set of conditions that must be true (resp. false) for o to be executable, and (α, δ) is the *effect* of o, that is, $\alpha \subseteq P$ (resp. $\delta \subseteq P$) is the set of conditions that are made true (resp. false) by the execution of o. For all $((\gamma, \theta), (\alpha, \delta)) \in O$, we have that $\gamma \cap \theta = \emptyset$ and $\alpha \cap \delta = \emptyset$.

From a STRIPS, we derive an automaton as follows. The set of states is 2^P, that is, a state is represented by the set of conditions that are true in it. The initial state is I. The set of goal states are states Q such that $Q \supseteq M$ and $Q \cap N = \emptyset$. There is a transition from state Q to state Q' labeled by the operator $o = ((\gamma, \theta), (\alpha, \delta))$ if $Q \supseteq \gamma$, $Q \cap \theta = \emptyset$ (the guard is satisfied) and $Q' = (Q \cup \alpha) \setminus \delta$ (the effect is applied). A standard problem is to ask whether or not there exists a run from the initial state to a goal state.

Monotonic STRIPSs. A *monotonic STRIPS (MS)* is a tuple (P, I, M, O) where P and I are defined as for STRIPSs, $M \subseteq P$ specifies which conditions must be true in a goal state, and O is a finite set of operators. In the MS definition, an operator $o \in O$ is a pair $(\gamma, (\alpha, \delta))$ where $\gamma \subseteq P$ is the guard of o, that is, the set of conditions that must be true for o to be executable, and (α, δ) is the effect of o as in the STRIPS definition. MSs thus differ from STRIPSs in the sense that guards only apply on conditions that are true in states, and goal states are only specified by true conditions. The monotonicity will appear more clearly when we will derive hereafter monotonic MDPs from MSs.

Each STRIPS $\mathcal{S} = (P, I, (M, N), O)$ can be made monotonic by duplicating the set of conditions, in the following way. We denote by $\overline{P}$ the set $\{\overline{p} \mid p \in P\}$ containing a new condition $\overline{p}$ for each $p \in P$ such that $\overline{p}$ represents the negation of the propositional variable p. We construct from $\mathcal{S}$ an MS $\mathcal{S}' = (P', I', M', O')$ such that $P' = P \cup \overline{P}$, $I' = I \cup \overline{P \setminus I} \subseteq P'$, $M' = M \cup \overline{N} \subseteq P'$ and $O' = \{(\gamma \cup \overline{\theta}, (\alpha \cup \overline{\delta}, \delta \cup \overline{\alpha})) \mid ((\gamma, \theta), (\alpha, \delta)) \in O\}$. It is easy to check that $\mathcal{S}$ and $\mathcal{S}'$ are equivalent (a state Q in $\mathcal{S}$ has its counterpart $Q \cup \overline{P \setminus Q}$ in $\mathcal{S}'$). In the following, we thus only consider MSs.

Example 8.1. To illustrate the notion of MS, let us consider the following example of the monkey trying to reach a bunch of bananas (cf. Example 7.1, page 116). Let (P, I, M, O) be an MS such that:

- $P = \{box, stick, bananas\}$,
- $I = \emptyset$,
- $M = \{bananas\}$, and
- $O = \{takebox, takestick, takebananas\}$ where:
  - $takebox = (\emptyset, (\{box\}, \emptyset))$,
  - $takestick = (\emptyset, (\{stick\}, \emptyset))$, and
  - $takebananas = (\{box, stick\}, (\{bananas\}, \emptyset))$.

In this MS, a condition $p \in P$ is true when the monkey possesses the item corresponding to p. At the beginning, the monkey possesses no item, i.e. I is the empty set, and its goal is to get the *bananas*, i.e. to reach a state $s \supseteq \{bananas\}$. This can be done by first executing the operators *takebox* and *takestick* to respectively get the *box* and the *stick*, and then executing *takebananas*, the guard of which is $\{box, stick\}$. ◁

Monotonic stochastic STRIPSs. MSs can be extended with stochastic aspects as follows [BL00]. Each operator $o = (\gamma, \pi) \in O$ now consists of a guard γ as before, and an effect given as a probability distribution $\pi : 2^P \times 2^P \to [0, 1]$ on the set of pairs (α, δ). An MS extended with such stochastic aspects is called a *monotonic stochastic STRIPS (MSS)*.

Additionally, we associate with an MSS (P, I, M, O) a weight function $w_O : O \to \mathbb{R}_{>0}$ that associates a strictly positive weight with each operator. The problem of planning from MSSs is then to minimize the expected truncated sum up to the set of goal states from the initial state, i.e. this is a version of the SSP problem (see footnote 8, Chapter 2).

Example 8.2. We extend the MS of Example 8.1 with stochastic aspects to illustrate the notion of MSS. Let (P, I, M, O) be an MSS such that P, I and M are defined as in Example 8.1, and $O = \{takebox, takestick, takebananaswithbox, takebananaswithstick, takebananaswithboth\}$ where:

- $takebox = (\emptyset, (1 : (\{box\}, \emptyset)))$,
- $takestick = (\emptyset, (1 : (\{stick\}, \emptyset)))$,
- $takebananaswithbox = (\{box\}, (\frac{1}{4} : (\{bananas\}, \emptyset), \frac{3}{4} : (\emptyset, \emptyset)))$,
- $takebananaswithstick = (\{stick\}, (\frac{1}{5} : (\{bananas\}, \emptyset), \frac{4}{5} : (\emptyset, \emptyset)))$, and
- $takebananaswithboth = (\{box, stick\}, (\frac{1}{2} : (\{bananas\}, \emptyset), \frac{1}{2} : (\emptyset, \emptyset)))$.

In this MSS, the monkey has a strictly positive probability to fail reaching the *bananas*, whatever the items it uses. However, the probability of success increases when it has both the *box* and the *stick*. ◁

8.1.2 Reduction to monotonic MDPs

In the following, we show that MSSs naturally define monotonic MDPs on which the pseudo-antichain based symblicit algorithm of Section 7.4 can be applied.

From MSSs to monotonic MDPs. Let $\mathcal{S} = (P, I, M, O)$ be an MSS. We can derive from $\mathcal{S}$ an MDP $\mathcal{M}_\mathcal{S} = (S, \Sigma, T, \mathcal{E}, \mathcal{D})$ together with a set of goal states G and a weight function w such that:

- $S = 2^P$,
- $G = \{s \in S \mid s \supseteq M\}$,

- $\Sigma = O$, and for all $s \in S$, $\Sigma_s = \{(\gamma, \pi) \in \Sigma \mid s \supseteq \gamma\}$,
- $T = \{(\alpha, \delta) \in 2^P \times 2^P \mid \exists (\gamma, \pi) \in O, (\alpha, \delta) \in \mathsf{Supp}(\pi)\}$,
- $\mathcal{E}$, $\mathcal{D}$ and w are defined for all $s \in S$ and $\sigma = (\gamma, \pi) \in \Sigma_s$, such that:
  - for all $\tau = (\alpha, \delta) \in T$, $\mathcal{E}(s, \sigma)(\tau) = (s \cup \alpha) \setminus \delta$,
  - for all $\tau \in T$, $\mathcal{D}(s, \sigma)(\tau) = \pi(\tau)$, and
  - $w(s, \sigma) = w_O(\sigma)$.

Note that we might have that $\mathcal{M}_\mathcal{S}$ is not Σ-non-blocking, if no operator can be applied on some state of $\mathcal{S}$. In this case, we get a Σ-non-blocking MDP from $\mathcal{M}_\mathcal{S}$ by eliminating states s with $\Sigma_s = \emptyset$ as long as it is necessary.

Lemma 8.3. *The* MDP$\mathcal{M}_\mathcal{S}$ *is monotonic,* G *is closed, and functions* $\mathcal{D}$ *and* w *are independent of* S*.*

Proof. First, S is equipped with the partial order $\supseteq$ and $\langle S, \supseteq \rangle$ is a lower semilattice with greatest lower bound $\cup$.[1] Second, we have that $\supseteq$ is compatible with $\mathcal{E}$. Indeed, for all $s, s' \in S$ such that $s \supseteq s'$, for all $\sigma \in \Sigma$ and $\tau = (\alpha, \delta) \in T$, $\mathcal{E}(s, \sigma)(\tau) = (s \cup \alpha) \setminus \delta \supseteq (s' \cup \alpha) \setminus \delta = \mathcal{E}(s', \sigma)(\tau)$. Finally the set $G = {\downarrow}\{M\}$ of goal states is closed for $\supseteq$, and $\mathcal{D}$ and w are clearly independent of S. □

Symblicit algorithm. In order to apply the pseudo-antichain based symblicit algorithm of Section 7.4 on the monotonic MDPs derived from MSSs, Assumptions 7.12 and 7.14 must hold. Let us show that Assumption 7.14 is satisfied. For all $s \in S$, $\sigma = (\gamma, \pi) \in \Sigma_s$ and $\tau = (\alpha, \delta) \in T$, we clearly have an algorithm for computing $\mathcal{E}(s, \sigma)(\tau) = (s \cup \alpha) \setminus \delta$, and $\mathcal{D}(s, \sigma)(\tau) = \pi(\tau)$. Let us now consider Assumption 7.12. An algorithm for computing $\lceil \mathsf{Pre}_{\sigma,\tau}({\downarrow}\{x\}) \rceil$, for all $x \in S$, $\sigma \in \Sigma$ and $\tau \in T$, is given by the next proposition.

Proposition 8.4. *Let* $x \in S$, $\sigma = (\gamma, \pi) \in \Sigma$ *and* $\tau = (\alpha, \delta) \in T$*. If* $x \cap \delta \neq \emptyset$*, then* $\lceil \mathsf{Pre}_{\sigma,\tau}({\downarrow}\{x\}) \rceil = \emptyset$*, otherwise* $\lceil \mathsf{Pre}_{\sigma,\tau}({\downarrow}\{x\}) \rceil = \{\gamma \cup (x \setminus \alpha)\}$*.*

Proof. Suppose first that $x \cap \delta = \emptyset$.

We first prove that $s = \gamma \cup (x \setminus \alpha) \in \mathsf{Pre}_{\sigma,\tau}({\downarrow}\{x\})$. We have to show that $\sigma \in \Sigma_s$ and $\mathcal{E}(s, \sigma)(\tau) \in {\downarrow}\{x\}$. Recall that $\sigma = (\gamma, \pi)$. We have $s = \gamma \cup (x \setminus \alpha) \supseteq \gamma$,

[1] Actually, $\langle S, \supseteq \rangle$ is also an upper semilattice with least upper bound $\cap$. It is thus a complete lattice.

which shows that $\sigma \in \Sigma_s$. We have that $\mathcal{E}(s,\sigma)(\tau) = (\gamma \cup (x \setminus \alpha) \cup \alpha) \setminus \delta = (\gamma \cup x \cup \alpha) \setminus \delta \supseteq x$ since $x \cap \delta = \emptyset$. We thus have that $\mathcal{E}(s,\sigma)(\tau) \in \downarrow\{x\}$.

We then prove that for all $s \in \mathsf{Pre}_{\sigma,\tau}(\downarrow\{x\})$, $s \in \downarrow\{\gamma \cup (x \setminus \alpha)\}$, i.e. $s \supseteq \gamma \cup (x \setminus \alpha)$. Let $s \in \mathsf{Pre}_{\sigma,\tau}(\downarrow\{x\})$. We have $\sigma \in \Sigma_s$ and $\mathcal{E}(s,\sigma)(\tau) \in \downarrow\{x\}$, that is, $s \supseteq \gamma$ and $\mathcal{E}(s,\sigma)(\tau) = (s \cup \alpha) \setminus \delta \supseteq x$. By classical set properties, it follows that $(s \cup \alpha) \supseteq x$, and then $s \supseteq x \setminus \alpha$. Finally, since $s \supseteq \gamma$, we have $s \supseteq \gamma \cup (x \setminus \alpha)$, as required.

Suppose now that $x \cap \delta \neq \emptyset$, then $\mathsf{Pre}_{\sigma,\tau}(\downarrow\{x\}) = \emptyset$. Indeed for all $s \in \downarrow\{x\}$, we have $s \cap \delta \neq \emptyset$, and by definition of $\mathcal{E}$, there is no s' such that $\mathcal{E}(s',\sigma)(\tau) = s$. □

Finally, notice that for the class of monotonic MDPs derived from MSSs, the symbolic representations described in Section 7.4.2 are compact, since G is closed, and $\mathcal{D}$ and w are independent of S (see Lemma 8.3). Therefore we have all the required ingredients for an efficient pseudo-antichain based algorithm to solve the SSP problem for MSSs. The next experiments show its performance.

8.1.3 Experiments

We have implemented the pseudo-antichain based symblicit algorithm for the SSP problem in a prototype written in Python and C [STR]. The C language is used for all the low level operations while the orchestration is done with Python. The binding between C and Python is realized with the ctypes library of Python. The source code is publicly available at `http://lit2.ulb.ac.be/stripssolver/`, together with the two benchmarks presented in this section. We compare our implementation with the purely explicit strategy iteration algorithm implemented in the development release 4.1.dev.r7712 of the tool PRISM [KNP11], since to the best of our knowledge, there is no tool implementing an MTBDD based symblicit algorithm for the SSP problem.[2] Note that this explicit implementation exists primarily to prototype new techniques and is thus not fully optimized [Par13]. Note also that value iteration algorithms are also implemented in PRISM. While those algorithms are usually efficient, they only compute approximations. As a consequence, for the sake of a fair comparison, we consider here only the performance of strategy iteration algorithms.

[2] A comparison with an MTBDD based symblicit algorithm is done in the second application for the EMP problem.

Monkey. The first benchmark is obtained from Example 8.2. In this benchmark, the monkey has several items at its disposal to reach the bunch of bananas, one of them being a stick. However, the stick is available as a set of several pieces that the monkey has to assemble. Moreover, the monkey has multiple ways to build the stick as there are several sets of pieces that can be put together. However, the time required to build a stick varies from a set of pieces to another. Additionally, we add useless items in the room: there is always a set of pieces from which the probability of getting a stick is 0. The operators of getting some items are stochastic, as well as the operator of getting the bananas: the probability of success varies according to the owned items (cf. Example 8.2). The benchmark is parameterized in the pair (p, s) where p is the number of pieces required to build a stick, and $p \cdot s$ is the total number of pieces. Results are given in Table 8.1.

Moats and castles. The second benchmark is an adaptation of a benchmark of [ML98] as proposed in [BL00][3]. The goal is to build a sand castle on the beach; a moat can be dug before in a way to protect it. We consider up to 7 discrete depths of moat. The operator of building the castle is stochastic: there is a strictly positive probability for the castle to be demolished by the waves. However, the deeper the moat is, the higher the probability of success is. For example, the first depth of moat offers a probability $\frac{1}{4}$ of success, while with the second depth of moat, the castle has probability $\frac{9}{20}$ to resist to the waves. The optimal strategy for this problem is to dig up to a given depth of moat and then repeat the action of building the castle until it succeeds. The optimal depth of moat then depends on the cost of the operators and the respective probability of successfully building the castle for each depth of moat. To increase the difficulty of the problem, we consider building several castles, each one having its own moat. The benchmark is parameterized in the pair (c, d) where d is the number of depths of moat that can be dug, and c is the number of castles that have to be built. Results are given in Table 8.2.

[3] In [BL00], the authors study a different problem which is to maximize the probability of reaching the goal within a given number of steps.

Table 8.1: Stochastic shortest path problem on the Monkey benchmark. The column (s, p) gives the parameters of the problem, $\mathbb{E}_\lambda^{\mathsf{TS}_G}$ the expected truncated sum of the computed strategy λ, and $|\mathcal{M}_S|$ the number of states of the MDP. For the pseudo-antichain based implementation (PA), #it is the number of iterations of the strategy iteration algorithm, $|S_{\sim_L}|$ the maximum size of computed bisimulation quotients, *lump* the total time spent for lumping, *syst* the total time spent for solving the linear systems, and *impr* the total time spent for improving the strategies. For the explicit implementation (Explicit), *constr* is the time spent for model construction and *strat* the time spent for the strategy iteration algorithm. For both implementations, *total* is the total execution time and *mem* the total memory consumption. All times are given in seconds and all memory consumptions are given in megabytes.

			PA							Explicit			
(s,p)	$\mathbb{E}_\lambda^{\mathsf{TS}_G}$	$\lvert\mathcal{M}_S\rvert$	#it	$\lvert S_{\sim_L}\rvert$	lump	syst	impr	total	mem	constr	strat	total	mem
$(1,2)$	35.75	256	4	15	0.01	0.00	0.02	0.03	15.6	0.4	0.03	0.43	178.2
$(1,3)$	35.75	1024	5	19	0.04	0.00	0.03	0.07	15.8	3.42	0.09	3.51	336.7
$(1,4)$	35.75	4096	6	31	0.17	0.00	0.12	0.29	16.3	55.29	0.16	55.45	1735.1
$(1,5)$	36.00	16384	7	39	0.75	0.00	0.62	1.37	17.1				**MO**
$(2,2)$	34.75	1024	5	19	0.05	0.00	0.03	0.09	15.8	3.2	0.12	3.32	379.9
$(2,3)$	34.75	8192	5	37	0.32	0.00	0.13	0.45	16.4	240.66	0.30	240.96	3463.2
$(2,4)$	34.75	65536	6	45	2.39	0.01	1.04	3.44	18.0				**MO**
$(2,5)$	35.75	524288	7	65	27.56	0.02	10.13	37.71	23.4				**MO**
$(3,2)$	35.75	4096	4	23	0.09	0.00	0.07	0.16	16.0	60.43	0.16	60.59	1625.8
$(3,3)$	35.75	65536	5	43	1.14	0.00	0.43	1.57	17.3				**MO**
$(3,4)$	35.75	1048576	6	57	12.89	0.01	4.92	17.83	21.7				**MO**
$(3,5)$	36.00	16777216	7	88	208.33	0.05	63.73	272.13	37.5				**MO**
$(4,2)$	35.75	16384	4	29	0.22	0.02	0.14	0.38	16.3	1114.19	0.70	1114.89	1704.3
$(4,3)$	35.75	524288	5	50	2.72	0.00	1.26	4.00	18.3				**MO**
$(4,4)$	35.75	16777216	6	87	45.68	0.04	22.41	68.14	25.0				**MO**
$(4,5)$	36.00	536870912	7	114	724.77	0.11	532.46	1257.41	60.9				**MO**
$(5,2)$	35.75	65536	4	31	0.36	0.00	0.18	0.54	16.6	20312.67	3.50	20316.17	2342.6
$(5,3)$	35.75	4194304	5	56	5.71	0.02	2.47	8.20	19.5				**MO**
$(5,4)$	35.75	268435456	6	97	95.49	0.04	101.27	196.83	31.3				**MO**
$(5,5)$	36.00	17179869184	7	152	1813.78	0.08	5284.31	7098.40	81.3				**MO**

Table 8.2: Stochastic shortest path problem on the Moats and castles benchmark. The column (c, d) gives the parameters of the problem and all other columns have the same meaning as in Table 8.1.

			PA							Explicit			
(c, d)	$\mathbb{E}_\lambda^{\mathsf{TS}_G}$	$\lvert M_S \rvert$	#it	$\lvert S_{\sim_L} \rvert$	lump	syst	impr	total	mem	constr	strat	total	mem
$(2, 3)$	39.3333	256	3	17	0.05	0.01	0.04	0.10	15.8	0.46	0.03	0.49	206.9
$(2, 4)$	34.6667	1024	3	34	0.41	0.00	0.17	0.58	16.5	6.30	0.08	6.38	483.8
$(2, 5)$	32.2222	4096	3	49	1.36	0.00	0.45	1.82	17.3	133.46	0.20	133.66	1202.5
$(2, 6)$	32.2222	16384	3	66	9.71	0.01	1.95	11.68	19.3	2966.01	0.79	2966.80	1706.2
$(3, 2)$	72.6667	512	3	45	0.52	0.00	0.24	0.77	16.6	1.77	0.06	1.83	282.9
$(3, 3)$	59.0000	4096	3	84	12.58	0.03	2.73	15.35	20.2	149.44	0.20	149.64	1205.5
$(3, 4)$	52.0000	32768	3	219	129.17	0.05	21.56	150.83	30.7	14658.22	2.47	14660.69	1610.9
$(3, 5)$	48.3333	262144	3	357	658.86	0.13	81.08	740.17	49.1				**MO**
$(3, 6)$	48.3333	2097152	3	595	10730.09	0.42	865.48	11596.71	145.8				**MO**
$(4, 2)$	96.8889	4096	3	132	31.61	0.03	12.06	43.72	26.5	173.40	0.22	173.62	1211.2
$(4, 3)$	78.6667	65536	3	464	1376.94	0.21	217.06	1594.48	82.2				**MO**

Observations. On those two benchmarks, we observe that the explicit implementation quickly runs out of memory when the state space of the MDP grows. Indeed, with this method, we were not able to solve MDPs with more than 65536 (resp. 32768) states in Table 8.1 (resp. Table 8.2). On the other hand, the symblicit algorithm behaves well on large models: the memory consumption never exceeds 150Mo and this even for MDPs with hundreds of millions of states. For instance, the example $(5, 5)$ of the Monkey benchmark is an MDP of more than 17 billions of states that is solved in less than 2 hours with only 82Mo of memory.

Finally, note that the value iteration algorithm of PRISM performs better than the strategy iteration one with respect to the run time and memory consumption on our benchmarks. However, it still consumes more memory than the pseudo-antichain based algorithm, and runs out of memory on several examples.

8.2 Expected mean-payoff with $\mathsf{LTL}_{\mathsf{MP}}$ synthesis

We consider another application of the pseudo-antichain based symblicit algorithm, but now for the EMP problem. This application is related to the problems of $\mathsf{LTL}_{\mathsf{MP}}$ realizability and synthesis introduced in Chapter 6. We here show how we can go beyond $\mathsf{LTL}_{\mathsf{MP}}$ synthesis.

Given an LTL formula ϕ on $P = I \uplus O$ with derived alphabets Σ_P, Σ_I and Σ_O, a weight function[4] $w' : \mathsf{Lit}(O) \to \mathbb{Z}$ and a threshold $\nu \in \mathbb{Q}$, recall from Chapter 6 that we know how to construct a family of SGs $\langle \mathcal{G}_{\phi,K,C}, \alpha \rangle$ with $\mathcal{G}_{\phi,K,C} = (S_1 \uplus S_2, s_0, E)$ from energy deterministic automata $\det(\mathcal{A}^{w'}, K) = (\langle \Sigma_P, w' \rangle, \mathcal{F}, F_0, \beta, \Delta)$ with $0 \leq K \leq \mathbb{K}$, $0 \leq C \leq \mathbb{C}$, such that ϕ is MP-realizable under finite memory if and only if Player 1 has a winning strategy in the SG $\langle \mathcal{G}_{\phi,K,C}, \alpha \rangle$ for some K and C. Moreover, we showed how to compute an arbitrary winning strategy from the set $\lceil \mathsf{Win}_1(\alpha) \rceil$, where $\mathsf{Win}_1(\alpha)$ is the fixpoint computed by the backward algorithm, when $s_0 \in \mathsf{Win}_1(\alpha)$. Recall that any winning strategy extracted from $\lceil \mathsf{Win}_1(\alpha) \rceil$ ensures a mean-payoff value greater than or equal to the given threshold ν in the worst-case, that is, against any strategy of Player 2. In this application, we are interested in constructing, from $\mathsf{Win}_1(\alpha)$ the winning strategy that behaves *the best against a stochastic environment*, that

[4] In Chapter 6, our definition of the weight function is more general since it associates values to $\mathsf{Lit}(P)$. However, for this application, we restrict it to $\mathsf{Lit}(O)$.

is, a strategy that is both winning in the worst-case for threshold ν, and optimal for the expected mean-payoff against an environment playing according to a probability distribution on Σ_I.

8.2.1 Reduction to monotonic MDPs

We first give a reduction from $\langle \mathcal{G}_{\phi,K,C}, \alpha \rangle$ to a monotonic MDP on which the pseudo-antichain based symblicit algorithm of Section 7.4 can be applied. For the sake of simplicity, we assume that $\langle \mathcal{G}_{\phi,K,C}, \alpha \rangle$ is simplified as suggested in Remark 6.19, that is, $S_1 \in \{(F, c) \mid F \in \mathcal{F}, c \in \mathcal{C}\}$, where $\mathcal{C} = \{\bot, 0, 1, \dots, C\}$.

From SGs to monotonic MDPs. For each state $s \in \mathsf{Win}_1(\alpha) \cap S_1$, we denote by $\Sigma_{O,s} \subseteq \Sigma_O$ the set of actions that force Player 2 to stay in $\mathsf{Win}_1(\alpha) \cap S_1$, that is $\Sigma_{O,s} = \{o \in \Sigma_O \mid \forall i \in \Sigma_I, \Delta(s, o \cup i) \in \mathsf{Win}_1(\alpha) \cap S_1\}$. For all $s \in \mathsf{Win}_1(\alpha) \cap S_1$, we know that $\Sigma_{O,s} \neq \emptyset$ by construction of $\mathsf{Win}_1(\alpha)$. Let $\pi_I : \Sigma_I \to]0, 1]$ be a probability distribution on the actions of the environment. Note that we require $\mathsf{Supp}(\pi_I) = \Sigma_I$ so that it makes sense with the worst-case. By replacing Player 2 by π_I in the SG $\langle \mathcal{G}_{\phi,K,C}, \alpha \rangle$ restricted to $\mathsf{Win}_1(\alpha)$, we derive an MDP $\mathcal{M}_{\mathcal{G}} = (S, \Sigma_O, \Sigma_I, \mathcal{E}, \mathcal{D})$ where:

- $S = \mathsf{Win}_1(\alpha) \cap S_1$,
- $\Sigma = \Sigma_O$, and for all $s \in S$, $\Sigma_s = \Sigma_{O,s}$,
- $T = \Sigma_I$,
- $\mathcal{E}$, $\mathcal{D}$ and w are defined for all $s = (F, c) \in S$ and $\sigma \in \Sigma_s$, such that:
  - for all $\tau \in T$, $\mathcal{E}(s, \sigma)(\tau) = (\Delta(F, \sigma \cup \tau), c \oplus w'(\sigma))$,
  - for all $\tau \in T$, $\mathcal{D}(s, \sigma)(\tau) = \pi_I(\tau)$, and
  - $w(s, \sigma) = w'(\sigma)$.

Note that since $\Sigma_{O,s} \neq \emptyset$ for all $s \in S$, we have that $\mathcal{M}_{\mathcal{G}}$ is Σ-non-blocking.

Computing the best strategy against a stochastic environment among the set of winning strategies represented in $\mathsf{Win}_1(\alpha)$ reduces to solving the EMP problem for the MDP $\mathcal{M}_{\mathcal{G}}$. We have the following lemma.

Lemma 8.5. *The MDP $\mathcal{M}_{\mathcal{G}}$ is monotonic, and functions $\mathcal{D}$ and w are independent of S.*

Proof. As shown in Section 6.4, the SG $\langle \mathcal{G}_{\phi,K,C}, \alpha \rangle$ has properties of monotony. The set $S_1 \uplus S_2$ is equipped with a partial order $\preceq$ such that $\langle S_1 \uplus S_2, \preceq \rangle$ is a lower semilattice, and the sets S_1, S_2 and $\mathsf{Win}_1(\alpha)$ are closed for $\preceq$. For the MDP $\mathcal{M}_{\mathcal{G}}$ derived from $\langle \mathcal{G}_{\phi,K,C}, \alpha \rangle$, we thus have that $\langle S, \preceq \rangle$ is a lower semilattice, and S is closed for $\preceq$. Moreover, by construction of $\langle \mathcal{G}_{\phi,K,C}, \alpha \rangle$ (see Section 6.4.1) and $\mathsf{Win}_1(\alpha)$, by Lemmas 5.14 and 6.22, and by construction of $\mathcal{M}_{\mathcal{G}}$, we have that $\preceq$ is compatible with $\mathcal{E}$. By construction, $\mathcal{D}$ and w are independent of S. □

Symblicit algorithm. In order to apply the pseudo-antichain based symblicit algorithm of Section 7.4, Assumptions 7.12 and 7.14 must hold for $\mathcal{M}_{\mathcal{G}}$. This is the case for Assumption 7.14 since $\mathcal{E}(s, \sigma)(\tau)$ can be computed for all $s \in S, \sigma \in \Sigma_s$ and $\tau \in T$ with function Δ and operator $\oplus$, and $\mathcal{D}$ is given by π_I. Moreover, from Proposition 6.24 and $\mathsf{Supp}(\pi_I) = \Sigma_I$, we have an algorithm for computing $\lceil \mathsf{Pre}_{\sigma,\tau}(\downarrow\{x\}) \rceil$, for all $x \in S$. So, Assumption 7.12 holds too. Notice also that for the MDP $\mathcal{M}_{\mathcal{G}}$ derived from the SG $\langle \mathcal{G}_{\phi,K,C}, \alpha \rangle$, the symbolic representations described in Section 7.4.2 are compact, since $\mathcal{D}$ and w are independent of S (see Lemma 8.5).

Therefore, for this second class of MDPs, we have again an efficient pseudo-antichain based algorithm to solve the EMP problem, as indicated by the next experiments.

8.2.2 Experiments

We have implemented the pseudo-antichain based symblicit algorithm for the EMP problem and integrated it into the tool Acacia+ [Aca] presented in Section 5.6. There is thus a fourth option in Acacia+ for strategy synthesis (in addition to those described in Section 5.6.3), which is the synthesis of an *optimal* winning strategy from $\mathsf{Win}_1(\alpha)$ against a stochastic environment given by a probability distribution on its set of actions. We compared our implementation (Acacia+ v2.2) with an MTBDD based symblicit algorithm implemented in a version of the tool PRISM [VE13]. In the sequel, for the sake of simplicity, we refer to the MTBDD based implementation as PRISM and to the pseudo-antichain based one as Acacia+. Notice that for Acacia+, the given execution times and memory consumptions only correspond to the part of the execution concerning the symblicit algorithm (and not the computation of the fixpoint $\mathsf{Win}_1(\alpha)$ in

the SG $\langle \mathcal{G}_{\phi,K,C}, \alpha \rangle$). All the examples considered in this section are available together with Acacia+ at http://lit2.ulb.ac.be/acaciaplus/.

Stochastic shared resource arbiter. We compare the two implementations on the benchmark of Table 6.1 (page 108) obtained from the client-server $\mathsf{LTL}_{\mathsf{MP}}$ specification of Example 6.2 (page 88) extended with stochastic aspects. For the stochastic environment, we set a probability distribution such that requests of client 1 are more likely to happen than requests of client 2: at each turn, client 1 has probability $\frac{3}{5}$ to make a request, while client 2 has probability $\frac{1}{5}$. The probability distribution $\pi_I : \Sigma_I \to]0, 1]$ is then defined as $\pi_I(\{\neg r_1, \neg r_2\}) = \frac{8}{25}$, $\pi_I(\{r_1, \neg r_2\}) = \frac{12}{25}$, $\pi_I(\{\neg r_1, r_2\}) = \frac{2}{25}$ and $\pi_I(\{r_1, r_2\}) = \frac{3}{25}$. We use the backward algorithm of Acacia+ for solving the related SGs. The benchmark is parameterized in the threshold value ν.

Results are given in Table 8.3. Note that the number of states in the MDPs depends on the implementation. Indeed, for PRISM, it is the number of reachable states in the MDP from the initial state s_0 of the related SG $\langle \mathcal{G}_{\phi,K,C}, \alpha \rangle$, that is, the states that are actually taken into account by the algorithm. This set of reachable states is denoted $|\mathcal{M}^R_{\mathcal{G}}|$. Unlike PRISM, our implementation does not prune unreachable states and thus, $|\mathcal{M}_{\mathcal{G}}|$ denotes the total number of states in $\mathcal{M}_{\mathcal{G}}$. For this application scenario, we observe that the ratio (number of reachable states)/(total number of states) is in general quite small.[5]

Observations. On this benchmark, PRISM is faster that Acacia+ on large models, but Acacia+ is more efficient regarding the memory consumption and this in spite of considering the whole state space. For instance, the last MDP of Table 8.3 contains more than 450 millions of states and is solved by Acacia+ in around 6.5 hours with less than 100Mo of memory, while for this example, PRISM runs out of memory. Note that the surprisingly large amount of memory consumption of both implementations on small instances is due to Python libraries loaded in memory for Acacia+, and to the JVM and the CUDD package for PRISM [JKO+07].

To fairly compare the two implementations, let us consider Figure 8.1 (resp. Figure 8.2) that gives a graphical representation of the execution times (resp.

[5] For all the MDPs considered in Tables 8.1 and 8.2, this ratio is 1.

Table 8.3: Expected mean-payoff problem on the Stochastic shared resource arbiter benchmark with 2 clients and decreasing threshold values. The column ν gives the threshold, $|\mathcal{M}^R_\mathcal{G}|$ the number of reachable states in the MDP, and all other columns have the same meaning as in Table 8.1. The expected mean-payoff $\mathbb{E}^{\mathsf{MP}}_\lambda$ of the optimal strategy λ for all the examples is -0.130435.

	Acacia+								PRISM				
ν	$\lvert\mathcal{M}_\mathcal{G}\rvert$	#it	$\lvert S_{\sim_L}\rvert$	lump	LS	impr	total	mem	$\lvert\mathcal{M}^R_\mathcal{G}\rvert$	constr	strat	total	mem
-1.4	366	2	8	0.01	0.0	0.01	0.02	16.7	186	0.2	0.04	0.24	149.0
-1.2	663	2	12	0.03	0.01	0.01	0.05	17.1	497	0.24	0.04	0.28	151.9
-1.1	5259	2	22	0.12	0.01	0.02	0.15	17.4	691	0.43	0.07	0.50	168.1
-1.05	72159	2	42	0.86	0.01	0.09	0.97	17.7	6440	1.58	0.18	1.76	249.9
-1.04	35750	2	52	1.63	0.02	0.13	1.79	18.1	3325	1.78	0.28	2.06	264.1
-1.03	501211	2	70	4.41	0.04	0.26	4.71	18.8	15829	4.83	0.46	5.29	277.0
-1.02	530299	2	102	16.62	0.11	0.64	17.39	20.2	11641	6.74	0.59	7.33	343.4
-1.01	4120599	2	202	237.78	0.50	3.94	242.30	26.2	43891	29.91	1.61	31.52	642.5
-1.005	64801599	2	402	3078.88	3.07	28.19	3110.50	48.0	563585	179.23	4.72	183.95	1629.2
-1.004	63251499	2	502	7357.72	5.68	52.81	7416.77	60.5	264391	270.30	7.71	278.01	2544.0
-1.003	450012211	2	670	23455.44	12.72	120.25	23589.49	93.6					**MO**

the memory consumption) of Acacia+ and PRISM as a function of the number of states taken into account, that is, the total number of states for Acacia+ and the number of reachable states for PRISM. For that experiment, we consider the benchmark of examples of Table 8.3 with four different probability distributions on Σ_I. Moreover, for each instance, we consider the two MDPs obtained with the backward and the forward algorithms of Acacia+ for solving safety games. The forward algorithm always leads to smaller MDPs. On the whole benchmark, Acacia+ times out on three instances, while PRISM runs out of memory on four of them. Note that all scales in Figures 8.1 and 8.2 are logarithmic.

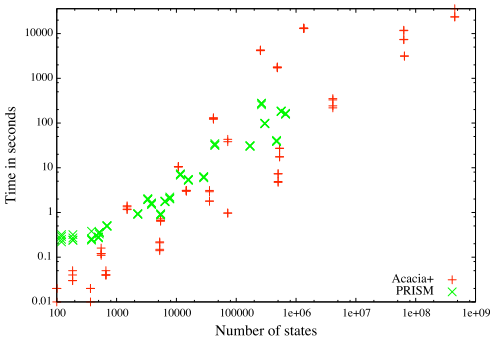

Figure 8.1: Execution times

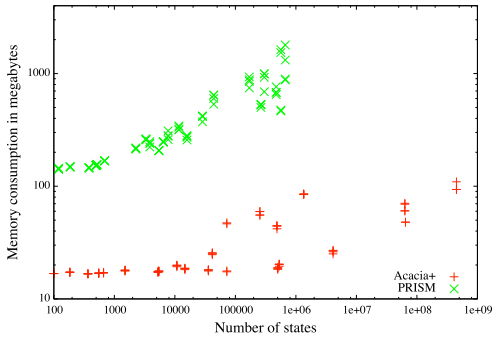

Figure 8.2: Memory consumptions

On Figure 8.1, we observe that for most of the executions, Acacia+ works faster that PRISM. We also observe that Acacia+ does not behave well for a few particular executions. These executions all correspond to MDPs obtained from the forward algorithm of Acacia+.

Figure 8.2 shows that regarding the memory consumption, Acacia+ is more efficient than PRISM and it can thus solve larger MDPs (the largest MDP solved by PRISM contains half a million states while Acacia+ solves MDPs of more than 450 million states). This points out that monotonic MDPs are better handled by pseudo-antichains, which exploit the partial order on the state space, than by MTBDDs.

Finally, in the majority of experiments we performed for both the EMP and the SSP problems, we observe that most of the execution time of the pseudo-antichain based symblicit algorithms is spent for lumping. It is also the case for the MTBDD based symblicit algorithm [WBB+10].

CHAPTER 9

Conclusion

In this final chapter, we summarize our contributions and we list possible extensions of this work.

9.1 Summary

With this thesis, we have contributed to the research in formal verification by providing algorithmic solutions to existing or new problems, and by validating those solutions experimentally with implementations in open-source tools.

Theoretical contributions. From the theoretical point of view, we introduced in Chapter 6 new problems in quantitative synthesis that are the problems of synthesis from LTL specifications with mean-payoff or energy objectives. We also considered the extension of those problems to multi-dimensional mean-payoff and multi-dimensional energy objectives. We showed that those problems are 2ExpTime-complete in the one-dimensional case, and in coN-2ExpTime in the multi-dimensional case. We also proposed antichain based algorithms for those problems based on a reduction to safety games.

In Chapter 7, we considered symblicit algorithms in MDPs, that is, algorithms that efficiently combine explicit and symbolic data structures. The symblicit approach has been initially proposed in [WBB+10] for the expected mean-payoff

problem, with (MT)BDDs as symbolic data structure. We showed that alternative data representations can be used instead of BDDs and MTBDDs. In particular, we considered the natural class of monotonic MDPs on which antichains like data structures have potential. To this end, we needed to generalize the structure of antichains in order to be closed under negation. We thus introduced the new data structure of pseudo-antichains, that we presented in Chapter 4. Moreover, we showed that the symblicit approach can also be applied to the stochastic shortest path problem, as shown in Chapter 7.

Practical contributions. From the practical point of view, we presented in Chapter 5 a new tool called Acacia+ implementing recent antichain based algorithms for solving the LTL realizability and synthesis problems. This tool is publicly available and it has been developed with emphasis on modularity, efficiency and usability. We reported experimental results for several application scenarios underlying the main advantage of Acacia+ which is the synthesis of compact strategies. Acacia+ is also extended to the synthesis from quantitative specifications. Indeed, it includes the antichain based algorithms presented in Chapter 6 for the synthesis from LTL specifications with (multi-dimensional) mean-payoff objectives. Finally, as indicated in Chapter 8, when performing the synthesis from quantitative specifications, Acacia+ allows to synthesize, among a set of worst-case winning strategies, the optimal strategy against a stochastic environment. Indeed, we have implemented the pseudo-antichain based symblicit algorithm for the expected mean-payoff problem and plugged it in Acacia+. To the best of our knowledge, this is the first tool for the automated synthesis of worst-case winning strategies from quantitative specifications which also ensure a good expected performance.

In Chapter 8, we also provided an implementation of the symblicit algorithm for the stochastic shortest path problem for monotonic MDPs arising in the field of probabilistic planning. This has been implemented in a publicly available prototype for which we reported promising experimental results.

9.2 Future work

To conclude this thesis, we discuss several possible extensions of our work.

Theoretical extensions. We first consider theoretical extensions of this thesis, starting with an extension of Chapter 5. In [Fae09], Faella considers two-player games played on finite graphs with a focus on *losing states*, that is, states from which Player 1 cannot enforce a winning outcome. He introduces the notion of *admissible strategy* which dictates the preferable behavior of Player 1 from those states. Intuitively, if Player 1 plays according to an admissible strategy, the outcome of the game might be winning for Player 1 from losing states if Player 2 does not play optimally. In the context of LTL synthesis, it might be interesting to consider admissible strategies for unrealizable specifications. This makes sense for instance when considering non-critical systems which may exhibit failures in some unusual circumstances.

We now consider future work related to Chapter 6. Theorem 6.26 establishes an upper bound on the complexity of the multi-dimensional $\mathsf{LTL}_{\mathsf{MP}}$ under finite memory and the multi-dimensional $\mathsf{LTL}_{\mathsf{E}}$ realizability problems. However, the exact complexity of those problems is still an open question.

Thanks to the determinacy of ω-regular games [Mar75], we have an efficient algorithm for checking the unrealizability of LTL formulas (see Theorem 5.13). Unfortunately, this is not the case for $\mathsf{LTL}_{\mathsf{MP}}$ unrealizability (because $\mathsf{LTL}_{\mathsf{MP}}$ realizability requires the conjunction of two objectives: realizing the given specification and ensuring a mean-payoff greater than or equal to the given threshold). An important question thus left open is how to determine in reasonable time whether an $\mathsf{LTL}_{\mathsf{MP}}$ specification is MP-unrealizable for a given threshold.

In Section 6.6.3, we reported experimental results on the problem of approaching the Pareto curve of optimal thresholds for a given multi-dimensional $\mathsf{LTL}_{\mathsf{MP}}$ specification. This raised an interesting question, directly linked to the problem of checking the MP-unrealizability of $\mathsf{LTL}_{\mathsf{MP}}$ specifications, that is to automatically compute points of this Pareto curve efficiently. Note that in a recent work [AMSS13], Abdulla et al. have devised an algorithm for computing the Pareto curve of minimum initial credits needed to win in a multi-dimensional energy parity game, solving an open question from the literature. However, our problem is different since we do not have a single game but rather a family of games depending on parameters K and C.

Finally, one could consider other quantitative objectives than mean-payoff or energy to combine with LTL. For instance, a generalization of the mean-payoff

objective, called *ratio* objective, has been considered in [vEJ12] and implemented in PRISM. The ratio objective is interesting for the synthesis of systems ensuring a good trade-off between conflicting objectives like the speed of a car against the fuel consumption. Another interesting alternative objective is the *window mean-payoff* objective, a conservative approximation of mean-payoff introduced in [CDRR13]. Unlike the classical mean-payoff objective, which asks to ensure a given threshold in the long-run, the window mean-payoff objective requires the mean of weights to satisfy the given threshold within a bounded window, sliding along a play. The window objective thus provides better time guarantee than the classical mean-payoff objective.

In Chapter 7, we proposed pseudo-antichain based symblicit algorithms for the expected mean-payoff (EMP) and the stochastic shortest path (SSP) problems in monotonic MDPs. The algorithmic basis of those symblicit algorithms is the strategy iteration algorithm of Howard and Veinott [How60, Vei66]. While strategy iteration algorithms usually ensure the convergence towards an optimal strategy, they are in general outperformed by value iteration algorithms [Bel57] for which only approximations are computed. MTBDD based value iteration algorithms (e.g. for the SSP problem) are implemented in PRISM. It could be interesting to investigate whether pseudo-antichains can be used instead of MTBDDs in those algorithms.

Practical extensions. We now consider practical extensions of our work. We start with extensions concerning Acacia+. In [GKG+10], the authors introduce *lattice-valued binary decision diagrams (LVBDDs)*, an extension of BDDs for the compact representation and manipulation of functions of the form $\{0,1\}^n \to \mathcal{L}$, where $\mathcal{L}$ is a finite complete lattice. They apply their new data structure to the satisfiability problem of LTL formula over finite words, that is, the problem of deciding, given an LTL formula ϕ over P, whether there exists a finite word $u \in (2^P)^*$ such that $u \models \phi$. They show experimentally that algorithms combining antichains and LVBDDs can outperform algorithms combining antichains and BDDs, and purely BDD based algorithms. The structure of LVBDDs could be used in Acacia+ to encode the set of edges of the manipulated safety games. This might speed up the computation of the set of predecessors of a given set of states in those games.

One of the main advantages of Acacia+, in comparison with other synthesis tools, is probably the synthesis of *compact* strategies, represented by infinite word automata or Moore machines. However, those representations could sometimes be even more succinct. For instance, in Section 6.6.1, we mentioned that all strategies of size $|\mathcal{M}|$ of Table 6.1 could be represented by a two-state automaton equipped with a counter that counts up to $|\mathcal{M}|$. Novel formats of winning strategies like *programs* have been recently investigated to obtain a more succinct representation of the synthesized systems (e.g. [Mad11, Gel12, Brü13]). It could be interesting to consider those representations of winning strategies in Acacia+.

Finally, as an alternative to $\mathsf{LTL}_{\mathsf{MP}}$ synthesis, it could also be interesting to allow the user to explicitly express the specification using a game graph, and to solve it with the antichain based algorithm of [CRR12] for computing winning strategies in multi-dimensional energy games.

In Chapter 8, we compared our symblicit algorithm for the SSP problem with an explicit implementation of the strategy iteration algorithm. It would have been interesting to compare it also with a symblicit implementation using MTBDDs. However, to the best of our knowledge, no such implementation exists for the SSP problem. It would thus be nice to implement it. Moreover, in the EMP context, we saw that the ratio (number of reachable states)/(total number of states) is in general quite small. One could thus investigate a way to prune unreachable states within the pseudo-antichain based symblicit algorithm, as done in the MTBDD based one.

Finally, we introduced the new data structure of pseudo-antichains, and we used it for solving the EMP and SSP problems in monotonic MDPs for which we reported promising experimental results. We are convinced that this new data structure can be used in the design of efficient algorithms in other contexts like for instance model-checking or synthesis with non-stochastic models, as soon as a natural partial order can be exploited.

Bibliography

[AaP] AaPAL website. http://lit2.ulb.ac.be/aapal/.
Cited in pages 7 and 73

[Aca] Acacia+ website. http://lit2.ulb.ac.be/acaciaplus/.
Cited in pages 7, 55, 108 and 152

[ACH+10] Parosh Aziz Abdulla, Yu-Fang Chen, Lukás Holík, Richard Mayr, and Tomás Vojnar. When simulation meets antichains. In Esparza and Majumdar [EM10], pages 158–174. *Cited in page 35*

[ADCR89] Giorgio Ausiello, Mariangiola Dezani-Ciancaglini, and Simona Ronchi Della Rocca, editors. *Automata, Languages and Programming, 16th International Colloquium, ICALP89, Stresa, Italy, July 11-15, 1989, Proceedings*, volume 372 of *Lecture Notes in Computer Science*. Springer, 1989. *Cited in pages 164 and 179*

[AGK+10] Samson Abramsky, Cyril Gavoille, Claude Kirchner, Friedhelm Meyer auf der Heide, and Paul G. Spirakis, editors. *Automata, Languages and Programming, 37th International Colloquium, ICALP 2010, Bordeaux, France, July 6-10, 2010, Proceedings, Part II*, volume 6199 of *Lecture Notes in Computer Science*. Springer, 2010.
Cited in pages 167 and 169

[AHKV98] Rajeev Alur, Thomas A. Henzinger, Orna Kupferman, and Moshe Y. Vardi. Alternating refinement relations. In Davide Sangiorgi and Robert de Simone, editors, *CONCUR*, volume 1466 of

Lecture Notes in Computer Science, pages 163–178. Springer, 1998. *Cited in pages 67 and 102*

[AL04] Rajeev Alur and Salvatore La Torre. Deterministic generators and games for LTL fragments. *ACM Trans. Comput. Log.*, 5(1):1–25, 2004. *Cited in page 40*

[AL11] Parosh Aziz Abdulla and K. Rustan M. Leino, editors. *Tools and Algorithms for the Construction and Analysis of Systems - 17th International Conference, TACAS 2011, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2011, Saarbrücken, Germany, March 26-April 3, 2011. Proceedings*, volume 6605 of *Lecture Notes in Computer Science*. Springer, 2011. *Cited in pages 170 and 172*

[ALW89] Martín Abadi, Leslie Lamport, and Pierre Wolper. Realizable and unrealizable specifications of reactive systems. In Ausiello et al. [ADCR89], pages 1–17. *Cited in pages 3 and 36*

[AMSS13] Parosh Aziz Abdulla, Richard Mayr, Arnaud Sangnier, and Jeremy Sproston. Solving parity games on integer vectors. In Pedro R. D'Argenio and Hernán C. Melgratti, editors, *CONCUR*, volume 8052 of *Lecture Notes in Computer Science*, pages 106–120. Springer, 2013. *Cited in page 159*

[ATW06] Christoph Schulte Althoff, Wolfgang Thomas, and Nico Wallmeier. Observations on determinization of Büchi automata. *Theor. Comput. Sci.*, 363(2):224–233, 2006. *Cited in pages 36 and 61*

[BBF+12] Aaron Bohy, Véronique Bruyère, Emmanuel Filiot, Naiyong Jin, and Jean-François Raskin. Acacia+, a tool for LTL synthesis. In Madhusudan and Seshia [MS12], pages 652–657. *Cited in pages 7, 34, 37, 55, 59 and 73*

[BBFR12] Aaron Bohy, Véronique Bruyère, Emmanuel Filiot, and Jean-François Raskin. Synthesis from LTL specifications with mean-payoff objectives. *CoRR*, abs/1210.3539, 2012. *Cited in pages 38 and 85*

[BBFR13] Aaron Bohy, Véronique Bruyère, Emmanuel Filiot, and Jean-François Raskin. Synthesis from LTL specifications with mean-payoff objectives. In Nir Piterman and Scott A. Smolka, editors, *TACAS*, volume 7795 of *Lecture Notes in Computer Science*, pages 169–184. Springer, 2013. *Cited in pages 7, 34, 38, 42, 55 and 85*

[BBKS13] Tomás Babiak, Frantisek Blahoudek, Mojmír Kretínský, and Jan Strejcek. Effective translation of LTL to deterministic Rabin automata: Beyond the (F, G)-fragment. In Hung and Ogawa [HO13], pages 24–39. *Cited in page 40*

[BBR14a] Aaron Bohy, Véronique Bruyère, and Jean-François Raskin. Symblicit algorithms for optimal strategy synthesis in monotonic Markov decision processes. *CoRR*, abs/1402.1076, 2014. *Cited in pages 34, 42, 49, 55, 115 and 141*

[BBR14b] Aaron Bohy, Véronique Bruyère, and Jean-François Raskin. Symblicit algorithms for optimal strategy synthesis in monotonic Markov decision processes. In *SYNT*, *EPTCS*, 2014. 17 pages. *Cited in pages 7, 34, 42, 47, 49, 115 and 141*

[BCD+11] Lubos Brim, Jakub Chaloupka, Laurent Doyen, Raffaella Gentilini, and Jean-François Raskin. Faster algorithms for mean-payoff games. *Formal Methods in System Design*, 38(2):97–118, 2011. *Cited in pages 40 and 96*

[BCG+10] Roderick Bloem, Alessandro Cimatti, Karin Greimel, Georg Hofferek, Robert Könighofer, Marco Roveri, Viktor Schuppan, and Richard Seeber. Ratsy - a new requirements analysis tool with synthesis. In Tayssir Touili, Byron Cook, and Paul Jackson, editors, *CAV*, volume 6174 of *Lecture Notes in Computer Science*, pages 425–429. Springer, 2010. *Cited in pages 35 and 40*

[BCHJ09] Roderick Bloem, Krishnendu Chatterjee, Thomas A. Henzinger, and Barbara Jobstmann. Better quality in synthesis through quantitative objectives. In Bouajjani and Maler [BM09], pages 140–156. *Cited in page 40*

[BCHK11] Udi Boker, Krishnendu Chatterjee, Thomas A. Henzinger, and Orna Kupferman. Temporal specifications with accumulative values. In *LICS*, pages 43–52. IEEE Computer Society, 2011. *Cited in page 40*

[BCL91] Jerry R. Burch, Edmund M. Clarke, and David E. Long. Symbolic model checking with partitioned transition relations. In Arne Halaas and Peter B. Denyer, editors, *VLSI*, volume A-1 of *IFIP Transactions*, pages 49–58. North-Holland, 1991. *Cited in page 3*

[BCM+92] Jerry R. Burch, Edmund M. Clarke, Kenneth L. McMillan, David L. Dill, and L. J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Inf. Comput.*, 98(2):142–170, 1992. *Cited in pages 3 and 35*

[BDBK+94] Shai Ben-David, Allan Borodin, Richard M. Karp, Gábor Tardos, and Avi Wigderson. On the power of randomization in on-line algorithms. *Algorithmica*, 11(1):2–14, 1994. *Cited in page 4*

[BDLL13] Peter Bulychev, Alexandre David, Kim G. Larsen, and Guangyuan Li. Efficient controller synthesis for a fragment of $\mathrm{MTL}_{0,\infty}$. *Acta Informatica*, pages 1–28, 2013. *Cited in page 40*

[Bel57] Richard E. Bellman. *Dynamic Programming*. Princeton University Press, 1957. *Cited in pages 41 and 160*

[BFL+08] Patricia Bouyer, Ulrich Fahrenberg, Kim G. Larsen, Nicolas Markey, and Jirí Srba. Infinite runs in weighted timed automata with energy constraints. In Franck Cassez and Claude Jard, editors, *FORMATS*, volume 5215 of *Lecture Notes in Computer Science*, pages 33–47. Springer, 2008. *Cited in pages 5, 22 and 40*

[BFRR14] Véronique Bruyère, Emmanuel Filiot, Mickael Randour, and Jean-François Raskin. Meet your expectations with guarantees: Beyond worst-case synthesis in quantitative games. In Ernst W. Mayr and Natacha Portier, editors, *STACS*, volume 25 of *LIPIcs*, pages 199–213. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2014. *Cited in page 43*

[BGJ+07] Roderick Bloem, Stefan J. Galler, Barbara Jobstmann, Nir Piterman, Amir Pnueli, and Martin Weiglhofer. Specify, compile, run: Hardware from PSL. *Electr. Notes Theor. Comput. Sci.*, 190(4):3–16, 2007. *Cited in page 40*

[BHH+08] Ahmed Bouajjani, Peter Habermehl, Lukás Holík, Tayssir Touili, and Tomás Vojnar. Antichain-based universality and inclusion testing over nondeterministic finite tree automata. In Oscar H. Ibarra and Bala Ravikumar, editors, *CIAA*, volume 5148 of *Lecture Notes in Computer Science*, pages 57–67. Springer, 2008. *Cited in pages 6 and 35*

[BJK10] Tomás Brázdil, Petr Jancar, and Antonín Kucera. Reachability games on extended vector addition systems with states. In Abramsky et al. [AGK+10], pages 478–489. *Cited in page 24*

[BK08] Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT Press, 2008. *Cited in pages 2, 12, 29 and 133*

[BK09] Udi Boker and Orna Kupferman. Co-ing Büchi made tight and useful. In *LICS*, pages 245–254. IEEE Computer Society, 2009. *Cited in page 39*

[BKHW05] Christel Baier, Joost-Pieter Katoen, Holger Hermanns, and Verena Wolf. Comparative branching-time semantics for Markov chains. *Inf. Comput.*, 200(2):149–214, 2005. *Cited in page 124*

[BKRS12] Tomás Babiak, Mojmír Kretínský, Vojtech Rehák, and Jan Strejcek. LTL to Büchi automata translation: Fast and more deterministic. In Cormac Flanagan and Barbara König, editors, *TACAS*, volume 7214 of *Lecture Notes in Computer Science*, pages 95–109. Springer, 2012. *Cited in pages 75 and 77*

[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. *Cited in page 3*

[BL00] Avrim L. Blum and John C. Langford. Probabilistic planning in the graphplan framework. In *Recent Advances in AI Planning*, pages 319–332. Springer, 2000. *Cited in pages 42, 144 and 147*

[BM09] Ahmed Bouajjani and Oded Maler, editors. *Computer Aided Verification, 21st International Conference, CAV 2009, Grenoble, France, June 26 - July 2, 2009. Proceedings*, volume 5643 of *Lecture Notes in Computer Science*. Springer, 2009. *Cited in pages 165 and 175*

[BMOU11] Patricia Bouyer, Nicolas Markey, Jörg Olschewski, and Michael Ummels. Measuring permissiveness in parity games: Mean-payoff parity games revisited. In Tevfik Bultan and Pao-Ann Hsiung, editors, *ATVA*, volume 6996 of *Lecture Notes in Computer Science*, pages 135–149. Springer, 2011. *Cited in pages 23 and 40*

[Bor21] Emile Borel. La théorie du jeu et les équations intégrales à noyau symétrique. *Comptes Rendus de l'Académie des Sciences*, 173:1304–1308, 1921. *Cited in page 4*

[Brü13] Benedikt Brütsch. Synthesizing structured reactive programs via deterministic tree automata. In Fabio Mogavero, Aniello Murano, and Moshe Y. Vardi, editors, *SR*, volume 112 of *EPTCS*, pages 107–113, 2013. *Cited in page 161*

[Bry86] Randal E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Computers*, 35(8):677–691, 1986. *Cited in pages 3, 6 and 34*

[BT91] Dimitri P. Bertsekas and John N. Tsitsiklis. An analysis of stochastic shortest path problems. *Mathematics of Operations Research*, 16(3):580–595, 1991. *Cited in pages 6 and 30*

[BT96] Dimitri P. Bertsekas and John N. Tsitsiklis. *Neuro-dynamic programming*. Anthropological Field Studies. Athena Scientific, 1996. *Cited in pages 30, 42, 119 and 121*

[Büc62] J. Richard Büchi. On a decision method in restricted second-order arithmetic. In *International Congress on Logic, Methodology, and Philosophy of Science*, pages 1–11. Stanford University Press, 1962. *Cited in page 15*

[Buc94] Peter Buchholz. Exact and ordinary lumpability in finite Markov chains. *Journal of applied probability*, pages 59–75, 1994. *Cited in pages 42 and 123*

[CB06] Frank Ciesinski and Christel Baier. LiQuor: A tool for qualitative and quantitative linear time analysis of reactive systems. In *QEST*, pages 131–132. IEEE Computer Society, 2006. *Cited in page 43*

[CCGR00] Alessandro Cimatti, Edmund M. Clarke, Fausto Giunchiglia, and Marco Roveri. NuSMV: A new symbolic model checker. *STTT*, 2(4):410–425, 2000. *Cited in pages 3 and 35*

[CD10] Krishnendu Chatterjee and Laurent Doyen. Energy parity games. In Abramsky et al. [AGK+10], pages 599–610. *Cited in pages 22, 26, 38 and 40*

[CdAHS03] Arindam Chakrabarti, Luca de Alfaro, Thomas A. Henzinger, and Mariëlle Stoelinga. Resource interfaces. In Rajeev Alur and Insup Lee, editors, *EMSOFT*, volume 2855 of *Lecture Notes in Computer Science*, pages 117–133. Springer, 2003. *Cited in pages 5 and 22*

[CDF+05] Franck Cassez, Alexandre David, Emmanuel Fleury, Kim G. Larsen, and Didier Lime. Efficient on-the-fly algorithms for the analysis of timed games. In Martín Abadi and Luca de Alfaro, editors, *CONCUR*, volume 3653 of *Lecture Notes in Computer Science*, pages 66–80. Springer, 2005. *Cited in page 69*

[CDHR10] Krishnendu Chatterjee, Laurent Doyen, Thomas A. Henzinger, and Jean-François Raskin. Generalized mean-payoff and energy games. In Lodaya and Mahajan [LM10], pages 505–516. *Cited in pages 24, 26 and 40*

[CDRR13] Krishnendu Chatterjee, Laurent Doyen, Mickael Randour, and Jean-François Raskin. Looking at mean-payoff and total-payoff through windows. In Hung and Ogawa [HO13], pages 118–132. *Cited in page 160*

[CE81] Edmund M. Clarke and E. Allen Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In Dexter Kozen, editor, *Logic of Programs*, volume 131 of *Lecture Notes in Computer Science*, pages 52–71. Springer, 1981. *Cited in pages 2 and 43*

[CGJ+00] Edmund M. Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-guided abstraction refinement. In Emerson and Sistla [ES00], pages 154–169. *Cited in page 3*

[CGL94] Edmund M. Clarke, Orna Grumberg, and David E. Long. Model checking and abstraction. *ACM Trans. Program. Lang. Syst.*, 16(5):1512–1542, 1994. *Cited in page 3*

[CGP01] Edmund M. Clarke, Orna Grumberg, and Doron Peled. *Model checking*. MIT Press, 2001. *Cited in page 2*

[CHJ05] Krishnendu Chatterjee, Thomas A. Henzinger, and Marcin Jurdziński. Mean-payoff parity games. In *LICS*, pages 178–187. IEEE Computer Society, 2005. *Cited in pages 23, 38 and 40*

[CHJS11] Krishnendu Chatterjee, Thomas A. Henzinger, Barbara Jobstmann, and Rohit Singh. Quasy: Quantitative synthesis tool. In Abdulla and Leino [AL11], pages 267–271. *Cited in pages 40 and 43*

[Chu62] Alonzo Church. Logic, arithmetic and automata. In *Proceedings of the international congress of mathematicians*, pages 23–35, 1962. *Cited in page 3*

[CMZ+93] Edmund M. Clarke, Kenneth L. McMillan, Xudong Zhao, Masahiro Fujita, and J. Yang. Spectral transforms for large Boolean functions with applications to technology mapping. In *DAC*, pages 54–60, 1993. *Cited in pages 6 and 34*

[Cou38] Antoine-Augustin Cournot. *Recherches sur les principes mathématiques de la théorie des richesses.* Hachette, 1838. *Cited in page 4*

[CRR12] Krishnendu Chatterjee, Mickael Randour, and Jean-François Raskin. Strategy synthesis for multi-dimensional quantitative objectives. In Maciej Koutny and Irek Ulidowski, editors, *CONCUR*, volume 7454 of *Lecture Notes in Computer Science*, pages 115–131. Springer, 2012. *Cited in pages 23, 24, 25, 40 and 161*

[CY88] Costas Courcoubetis and Mihalis Yannakakis. Verifying temporal properties of finite-state probabilistic programs. In *FOCS*, pages 338–345. IEEE Computer Society, 1988. *Cited in page 6*

[CY95] Costas Courcoubetis and Mihalis Yannakakis. The complexity of probabilistic verification. *J. ACM*, 42(4):857–907, 1995. *Cited in page 5*

[dA99] Luca de Alfaro. Computing minimum and maximum reachability times in probabilistic systems. In Jos C. M. Baeten and Sjouke Mauw, editors, *CONCUR*, volume 1664 of *Lecture Notes in Computer Science*, pages 66–81. Springer, 1999. *Cited in pages 6, 42, 119, 120, 121 and 132*

[dAKN+00] Luca de Alfaro, Marta Z. Kwiatkowska, Gethin Norman, David Parker, and Roberto Segala. Symbolic model checking of probabilistic processes using MTBDDs and the Kronecker representation. In Susanne Graf and Michael I. Schwartzbach, editors, *TACAS*, volume 1785 of *Lecture Notes in Computer Science*, pages 395–410. Springer, 2000. *Cited in page 35*

[DDHR06] Martin De Wulf, Laurent Doyen, Thomas A. Henzinger, and Jean-François Raskin. Antichains: A new algorithm for checking universality of finite automata. In Thomas Ball and Robert B. Jones, editors, *CAV*, volume 4144 of *Lecture Notes in Computer Science*, pages 17–30. Springer, 2006. *Cited in pages 6, 34, 35 and 73*

[DHS03] Salem Derisavi, Holger Hermanns, and William H. Sanders. Optimal state-space lumping in Markov chains. *Inf. Process. Lett.*, 87(6):309–315, 2003. *Cited in pages 124 and 125*

[DLP04] Alexandre Duret-Lutz and Denis Poitrenaud. SPOT: An extensible model checking library using transition-based generalized Büchi automata. In Doug DeGroot, Peter G. Harrison, Harry A. G. Wijshoff, and Zary Segall, editors, *MASCOTS*, pages 76–83. IEEE Computer Society, 2004. *Cited in page 75*

[DR07] Laurent Doyen and Jean-François Raskin. Improved algorithms for the automata-based approach to model-checking. In Orna Grumberg and Michael Huth, editors, *TACAS*, volume 4424 of *Lecture Notes in Computer Science*, pages 451–465. Springer, 2007. *Cited in pages 6, 35 and 73*

[DR10] Laurent Doyen and Jean-François Raskin. Antichain algorithms for finite automata. In Esparza and Majumdar [EM10], pages 2–22. *Cited in pages 6 and 35*

[Ehl10] Rüdiger Ehlers. Minimising deterministic Büchi automata precisely using SAT solving. In Ofer Strichman and Stefan Szeider, editors, *SAT*, volume 6175 of *Lecture Notes in Computer Science*, pages 326–332. Springer, 2010. *Cited in page 81*

[Ehl11] Rüdiger Ehlers. Unbeast: Symbolic bounded synthesis. In Abdulla and Leino [AL11], pages 272–275. *Cited in pages 39, 77 and 80*

[Ehl12] Rüdiger Ehlers. Symbolic bounded synthesis. *Formal Methods in System Design*, 40(2):232–262, 2012. *Cited in pages 35 and 39*

[EJ88] E. Allen Emerson and Charanjit S. Jutla. The complexity of tree automata and logics of programs (extended abstract). In *FOCS*, pages 328–337. IEEE Computer Society, 1988. *Cited in page 39*

[EJ91] E. Allen Emerson and Charanjit S. Jutla. Tree automata, mu-calculus and determinacy (extended abstract). In *FOCS*, pages 368–377. IEEE Computer Society, 1991. *Cited in page 4*

[EM79] Andrzej Ehrenfeucht and Jan Mycielski. Positional strategies for mean payoff games. *International Journal of Game Theory*, 8(2):109–113, 1979. *Cited in page 5*

[EM10] Javier Esparza and Rupak Majumdar, editors. *Tools and Algorithms for the Construction and Analysis of Systems, 16th International Conference, TACAS 2010, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2010, Paphos, Cyprus, March 20-28, 2010. Proceedings*, volume 6015 of *Lecture Notes in Computer Science*. Springer, 2010. *Cited in pages 163 and 172*

[ES00] E. Allen Emerson and A. Prasad Sistla, editors. *Computer Aided Verification, 12th International Conference, CAV 2000, Chicago, IL, USA, July 15-19, 2000, Proceedings*, volume 1855 of *Lecture Notes in Computer Science*. Springer, 2000. *Cited in pages 170 and 181*

[Fae09] Marco Faella. Admissible strategies in infinite games over graphs. In Rastislav Královic and Damian Niwinski, editors, *MFCS*, volume 5734 of *Lecture Notes in Computer Science*, pages 307–318. Springer, 2009. *Cited in page 159*

[Fil11] Emmanuel Filiot. Personal communication, 07-Jan-2011. *Cited in page 70*

[FJR11] Emmanuel Filiot, Naiyong Jin, and Jean-François Raskin. Antichains and compositional algorithms for LTL synthesis. *Formal Methods in System Design*, 39(3):261–296, 2011. *Cited in pages 7, 37, 39, 48, 59, 61, 62, 63, 64, 69, 72, 99 and 105*

[FJR13] Emmanuel Filiot, Naiyong Jin, and Jean-François Raskin. Exploiting structure in LTL synthesis. *STTT*, 15(5-6):541–561, 2013. *Cited in pages 72 and 77*

[FMY97] Masahiro Fujita, Patrick C. McGeer, and Jerry Chih-Yuan Yang. Multi-terminal binary decision diagrams: An efficient data struc-

ture for matrix representation. *Formal Methods in System Design*, 10(2/3):149–169, 1997. *Cited in pages 6 and 34*

[FN72] Richard E. Fikes and Nils J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial intelligence*, 2(3):189–208, 1972. *Cited in page 142*

[FS01] Alain Finkel and Philippe Schnoebelen. Well-structured transition systems everywhere! *Theor. Comput. Sci.*, 256(1-2):63–92, 2001. *Cited in pages 119 and 129*

[FV97] Jerzy Filar and Koos Vrieze. *Competitive Markov decision processes*. Springer, 1997. *Cited in pages 6 and 29*

[Gel12] Marcus Gelderie. Strategy machines and their complexity. In Branislav Rovan, Vladimiro Sassone, and Peter Widmayer, editors, *MFCS*, volume 7464 of *Lecture Notes in Computer Science*, pages 431–442. Springer, 2012. *Cited in page 161*

[GH82] Yuri Gurevich and Leo Harrington. Trees, automata, and games. In Harry R. Lewis, Barbara B. Simons, Walter A. Burkhard, and Lawrence H. Landweber, editors, *STOC*, pages 60–65. ACM, 1982. *Cited in page 4*

[GKG+10] Gilles Geeraerts, Gabriel Kalyon, Tristan Le Gall, Nicolas Maquet, and Jean-François Raskin. Lattice-valued binary decision diagrams. In Ahmed Bouajjani and Wei-Ngan Chin, editors, *ATVA*, volume 6252 of *Lecture Notes in Computer Science*, pages 158–172. Springer, 2010. *Cited in page 160*

[GO01] Paul Gastin and Denis Oddoux. Fast LTL to Büchi automata translation. In Gérard Berry, Hubert Comon, and Alain Finkel, editors, *CAV*, volume 2102 of *Lecture Notes in Computer Science*, pages 53–65. Springer, 2001. *Cited in page 75*

[God90] Patrice Godefroid. Using partial orders to improve automatic verification methods. In Edmund M. Clarke and Robert P. Kurshan, editors, *CAV*, volume 531 of *Lecture Notes in Computer Science*, pages 176–185. Springer, 1990. *Cited in page 3*

[GS09] Thomas Gawlitza and Helmut Seidl. Games through nested fixpoints. In Bouajjani and Maler [BM09], pages 291–305. *Cited in page 5*

[GTW02] Erich Grädel, Wolfgang Thomas, and Thomas Wilke. *Automata, Logics, and Infinite Games: A Guide to Current Research*, volume 2500 of *Lecture Notes in Computer Science*. Springer-Verlag, 2002. *Cited in pages 21 and 180*

[GZ04] Hugo Gimbert and Wieslaw Zielonka. When can you play positionally? In Jirí Fiala, Václav Koubek, and Jan Kratochvíl, editors, *MFCS*, volume 3153 of *Lecture Notes in Computer Science*, pages 686–697. Springer, 2004. *Cited in page 5*

[Har12] Arnd Hartmanns. Modest - a unified language for quantitative models. In *FDL*, pages 44–51. IEEE, 2012. *Cited in page 43*

[Hen12] Thomas A. Henzinger. Quantitative reactive models. In Robert B. France, Jürgen Kazmeier, Ruth Breu, and Colin Atkinson, editors, *MoDELS*, volume 7590 of *Lecture Notes in Computer Science*, pages 1–2. Springer, 2012. *Cited in page 37*

[HJ94] Hans Hansson and Bengt Jonsson. A logic for reasoning about time and reliability. *Formal Asp. Comput.*, 6(5):512–535, 1994. *Cited in page 43*

[HO13] Dang Van Hung and Mizuhito Ogawa, editors. *Automated Technology for Verification and Analysis - 11th International Symposium, ATVA 2013, Hanoi, Vietnam, October 15-18, 2013. Proceedings*, volume 8172 of *Lecture Notes in Computer Science*. Springer, 2013. *Cited in pages 165, 170 and 177*

[Hol04] Gerard J. Holzmann. *The SPIN model checker - primer and reference manual*. Addison-Wesley, 2004. *Cited in page 3*

[How60] Ronald A. Howard. *Dynamic programming and Markov processes*. John Wiley and Sons, 1960. *Cited in pages 41, 119 and 160*

[HRS05] Aidan Harding, Mark Ryan, and Pierre-Yves Schobbens. A new algorithm for strategy synthesis in LTL games. In Nicolas Halbwachs and Lenore D. Zuck, editors, *TACAS*, volume 3440 of *Lecture Notes in Computer Science*, pages 477–492. Springer, 2005. *Cited in page 40*

[HS84] Sergiu Hart and Micha Sharir. Probabilistic temporal logics for finite and bounded models. In Richard A. DeMillo, editor, *STOC*, pages 1–13. ACM, 1984. *Cited in page 6*

[HSP83] Sergiu Hart, Micha Sharir, and Amir Pnueli. Termination of probabilistic concurrent program. *ACM Trans. Program. Lang. Syst.*, 5(3):356–380, 1983. *Cited in page 6*

[IBM] IBM. Rulebase tutorial. `https://www.research.ibm.com/haifa/projects/verification/rb_homepage/`. [Online; accessed 3-Mar-2014]. *Cited in page 77*

[JB06] Barbara Jobstmann and Roderick Bloem. Optimizations for LTL synthesis. In *FMCAD*, pages 117–124. IEEE Computer Society, 2006. *Cited in pages 39, 75 and 77*

[JGWB07] Barbara Jobstmann, Stefan J. Galler, Martin Weiglhofer, and Roderick Bloem. Anzu: A tool for property synthesis. In Werner Damm and Holger Hermanns, editors, *CAV*, volume 4590 of *Lecture Notes in Computer Science*, pages 258–262. Springer, 2007. *Cited in page 40*

[JKO+07] David N. Jansen, Joost-Pieter Katoen, Marcel Oldenkamp, Mariëlle Stoelinga, and Ivan S. Zapreev. How fast and fat is your probabilistic model checker? An experimental performance comparison. In Karen Yorav, editor, *Haifa Verification Conference*, volume 4899 of *Lecture Notes in Computer Science*, pages 69–85. Springer, 2007. *Cited in page 153*

[KB05] Joachim Klein and Christel Baier. Experiments with deterministic omega-automata for formulas of linear temporal logic. In Jacques Farré, Igor Litovsky, and Sylvain Schmitz, editors, *CIAA*,

volume 3845 of *Lecture Notes in Computer Science*, pages 199–212. Springer, 2005. *Cited in page 40*

[KE12] Jan Kretínský and Javier Esparza. Deterministic automata for the (F, G)-fragment of LTL. In Madhusudan and Seshia [MS12], pages 7–22. *Cited in pages 40 and 83*

[KHB10] Robert Könighofer, Georg Hofferek, and Roderick Bloem. Debugging unrealizable specifications with model-based diagnosis. In Sharon Barner, Ian G. Harris, Daniel Kroening, and Orna Raz, editors, *Haifa Verification Conference*, volume 6504 of *Lecture Notes in Computer Science*, pages 29–45. Springer, 2010. *Cited in page 80*

[KLG13] Jan Kretínský and Ruslán Ledesma-Garza. Rabinizer 2: Small deterministic automata for LTL\GU. In Hung and Ogawa [HO13], pages 446–450. *Cited in page 40*

[KNP11] Marta Z. Kwiatkowska, Gethin Norman, and David Parker. PRISM 4.0: Verification of probabilistic real-time systems. In Ganesh Gopalakrishnan and Shaz Qadeer, editors, *CAV*, volume 6806 of *Lecture Notes in Computer Science*, pages 585–591. Springer, 2011. *Cited in pages 35, 43 and 146*

[Koz83] Dexter Kozen. Results on the propositional mu-calculus. *Theor. Comput. Sci.*, 27:333–354, 1983. *Cited in page 119*

[KS60] John G. Kemeny and J. L. Snell. *Finite Markov chains*. Van Nostrand Company, Inc, 1960. *Cited in pages 42 and 123*

[KV05] Orna Kupferman and Moshe Y. Vardi. Safraless decision procedures. In *FOCS*, pages 531–542. IEEE Computer Society, 2005. *Cited in pages 17, 37, 39, 81 and 82*

[KZH+11] Joost-Pieter Katoen, Ivan S. Zapreev, Ernst Moritz Hahn, Holger Hermanns, and David N. Jansen. The ins and outs of the probabilistic model checker MRMC. *Perform. Eval.*, 68(2):90–104, 2011. *Cited in page 43*

[Lan69] Lawrence H. Landweber. Decision problems for omega-automata. *Mathematical Systems Theory*, 3(4):376–384, 1969. *Cited in page 17*

[LM10] Kamal Lodaya and Meena Mahajan, editors. *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2010, December 15-18, 2010, Chennai, India*, volume 8 of *LIPIcs*. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2010. *Cited in pages 169 and 181*

[LN] Jørn Lind-Nielsen. BuDDy: a BDD package. `http://sourceforge.net/projects/buddy/`. [Online; accessed 20-Mar-2014]. *Cited in page 35*

[LPY97] Kim G. Larsen, Paul Pettersson, and Wang Yi. UPPAAL in a nutshell. *STTT*, 1(1-2):134–152, 1997. *Cited in page 3*

[LS91] Kim G. Larsen and Arne Skou. Bisimulation through probabilistic testing. *Inf. Comput.*, 94(1):1–28, 1991. *Cited in page 123*

[Mad11] Parthasarathy Madhusudan. Synthesizing reactive programs. In Marc Bezem, editor, *CSL*, volume 12 of *LIPIcs*, pages 428–442. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2011. *Cited in page 161*

[Mar75] Donald A. Martin. Borel determinacy. *Annals of Mathematics*, 102:363–371, 1975. *Cited in pages 4, 23, 66 and 159*

[McM93] Kenneth L. McMillan. *Symbolic model checking: An approach to the state explosion problem*. Kluwer Academic, 1993. *Cited in page 3*

[ML98] Stephen M. Majercik and Michael L. Littman. Maxplan: A new approach to probabilistic planning. In Reid G. Simmons, Manuela M. Veloso, and Stephen F. Smith, editors, *AIPS*, pages 86–93. AAAI, 1998. *Cited in page 147*

[Mos84] Andrzej W. Mostowski. Regular expressions for infinite trees and a standard form of automata. In Andrzej Skowron, editor, *Symposium*

on Computation Theory, volume 208 of *Lecture Notes in Computer Science*, pages 157–168. Springer, 1984. *Cited in page 15*

[MS85] David E. Muller and Paul E. Schupp. The theory of ends, pushdown automata, and second-order logic. *Theor. Comput. Sci.*, 37:51–75, 1985. *Cited in page 4*

[MS12] P. Madhusudan and Sanjit A. Seshia, editors. *Computer Aided Verification - 24th International Conference, CAV 2012, Berkeley, CA, USA, July 7-13, 2012 Proceedings*, volume 7358 of *Lecture Notes in Computer Science*. Springer, 2012. *Cited in pages 164 and 177*

[Mul63] David E. Muller. Infinite sequences and finite machines. In *SWCT (FOCS)*, pages 3–16. IEEE Computer Society, 1963. *Cited in page 15*

[Pap03] Christos H. Papadimitriou. *Computational complexity*. John Wiley and Sons Ltd., 2003. *Cited in page 17*

[Par13] David Parker. Personal communication, 05-Dec-2013. *Cited in page 146*

[Pel96] Doron Peled. Combining partial order reductions with on-the-fly model-checking. *Formal Methods in System Design*, 8(1):39–64, 1996. *Cited in page 3*

[PPS06] Nir Piterman, Amir Pnueli, and Yaniv Sa'ar. Synthesis of reactive(1) designs. In E. Allen Emerson and Kedar S. Namjoshi, editors, *VMCAI*, volume 3855 of *Lecture Notes in Computer Science*, pages 364–380. Springer, 2006. *Cited in page 40*

[PR89a] Amir Pnueli and Roni Rosner. On the synthesis of a reactive module. In *POPL*, pages 179–190. ACM Press, 1989. *Cited in pages 3, 35 and 36*

[PR89b] Amir Pnueli and Roni Rosner. On the synthesis of an asynchronous reactive module. In Ausiello et al. [ADCR89], pages 652–671. *Cited in pages 3, 36, 60 and 61*

[PT87] Robert Paige and Robert E. Tarjan. Three partition refinement algorithms. *SIAM J. Comput.*, 16(6):973–989, 1987.
Cited in page 124

[Put94] Martin L. Puterman. *Markov decision processes: Discrete stochastic dynamic programming*. John Wiley and Sons, 1994.
Cited in pages 6, 29, 41 and 122

[PZ86] Amir Pnueli and Lenore D. Zuck. Verification of multiprocess probabilistic protocols. *Distributed Computing*, 1(1):53–72, 1986.
Cited in page 6

[QS82] Jean-Pierre Queille and Joseph Sifakis. Specification and verification of concurrent systems in CESAR. In Mariangiola Dezani-Ciancaglini and Ugo Montanari, editors, *Symposium on Programming*, volume 137 of *Lecture Notes in Computer Science*, pages 337–351. Springer, 1982. *Cited in page 2*

[Rab69] Michael O. Rabin. Decidability of second-order theories and automata on infinite trees. *Transactions of the American Mathematical Society*, 141:1–35, 1969. *Cited in pages 4 and 15*

[Rab72] Michael O. Rabin. *Automata on infinite objects and Church's problem*. Number 13 in Conference Board of the mathematical science. American Mathematical Soc., 1972. *Cited in pages 3 and 39*

[RN95] Stuart J. Russell and Peter Norvig. *Artificial intelligence: a modern approach*. Prentice hall Englewood Cliffs, 1995. *Cited in page 116*

[Rog01] Markus Roggenbach. Determinization of Büchi-automata. In *Automata, Logics, and Infinite Games* [GTW02], pages 43–60.
Cited in page 17

[Ros92] Roni Rosner. *Modular synthesis of reactive systems*. PhD thesis, Weizmann Institute of Science, 1992.
Cited in pages 3, 36, 60 and 61

[Saf88] Shmuel Safra. On the complexity of omega-automata. In *FOCS*, pages 319–327. IEEE Computer Society, 1988.
Cited in pages 17 and 36

[SB00] Fabio Somenzi and Roderick Bloem. Efficient Büchi automata from LTL formulae. In Emerson and Sistla [ES00], pages 248–263.
Cited in page 75

[Sch09] Sven Schewe. Tighter bounds for the determinisation of Büchi automata. In Luca de Alfaro, editor, *FOSSACS*, volume 5504 of *Lecture Notes in Computer Science*, pages 167–181. Springer, 2009.
Cited in pages 36 and 61

[Sch10] Sven Schewe. Beyond hyper-minimisation—Minimising DBAs and DPAs is NP-complete. In Lodaya and Mahajan [LM10], pages 400–411.
Cited in page 81

[SF07] Sven Schewe and Bernd Finkbeiner. Bounded synthesis. In Kedar S. Namjoshi, Tomohiro Yoneda, Teruo Higashino, and Yoshio Okamura, editors, *ATVA*, volume 4762 of *Lecture Notes in Computer Science*, pages 474–488. Springer, 2007.
Cited in pages 39, 61 and 62

[Sha53] Lloyd S. Shapley. Stochastic games. *Proceedings of the National Academy of Sciences of the United States of America*, 39(10):1095, 1953.
Cited in page 6

[Sip06] Michael Sipser. *Introduction to the Theory of Computation*. Cengage Learning, 2006.
Cited in page 17

[Som12] Fabio Somenzi. CUDD: CU decision diagram package release 2.5.0. *University of Colorado at Boulder*, 2012. `http://vlsi.colorado.edu/~fabio/CUDD/`. [Online; accessed 20-Mar-2014].
Cited in page 35

[SS09] Saqib Sohail and Fabio Somenzi. Safety first: A two-stage algorithm for LTL games. In *FMCAD*, pages 77–84. IEEE, 2009.
Cited in page 39

[STR] STRIPSSolver website. http://lit2.ulb.ac.be/stripssolver/. *Cited in pages 55 and 146*

[Str82] Robert S. Streett. Propositional dynamic logic of looping and converse is elementarily decidable. *Information and control*, 54(1):121–141, 1982. *Cited in page 15*

[TFVT10] Ming-Hsien Tsai, Seth Fogarty, Moshe Y. Vardi, and Yih-Kuen Tsay. State of Büchi complementation. In Michael Domaratzki and Kai Salomaa, editors, *CIAA*, volume 6482 of *Lecture Notes in Computer Science*, pages 261–271. Springer, 2010. *Cited in pages 36 and 61*

[Tho97] Wolfgang Thomas. Languages, automata, and logic. In Grzegorz Rozenberg and Arto Salomaa, editors, *Handbook of Formal Languages*, volume 3, Beyond words, pages 389–455. Springer, 1997. *Cited in pages 4 and 17*

[Val92] Antti Valmari. A stubborn attack on state explosion. *Formal Methods in System Design*, 1(4):297–322, 1992. *Cited in page 3*

[Var85] Moshe Y. Vardi. Automatic verification of probabilistic concurrent finite-state programs. In *FOCS*, pages 327–338. IEEE Computer Society, 1985. *Cited in page 5*

[VCD+12] Yaron Velner, Krishnendu Chatterjee, Laurent Doyen, Thomas A. Henzinger, Alexander Rabinovich, and Jean-François Raskin. The complexity of multi-mean-payoff and multi-energy games. *CoRR*, abs/1209.3234, 2012. *Cited in page 107*

[VE13] Christian Von Essen. Personal communication, 20-Nov-2013. *Cited in page 152*

[Vei66] Arthur F. Veinott. On finding optimal policies in discrete dynamic programming with no discounting. *The Annals of Mathematical Statistics*, 37(5):1284–1294, 1966. *Cited in pages 41, 122 and 160*

[vEJ12] Christian von Essen and Barbara Jobstmann. Synthesizing efficient controllers. In Viktor Kuncak and Andrey Rybalchenko, editors,

VMCAI, volume 7148 of *Lecture Notes in Computer Science*, pages 428–444. Springer, 2012. *Cited in pages 43 and 160*

[vNM44] John von Neumann and Oskar Morgenstern. *Theory of games and economic behavior*. Princeton University Press, 1944. *Cited in page 4*

[WBB+10] Ralf Wimmer, Bettina Braitling, Bernd Becker, Ernst Moritz Hahn, Pepijn Crouzen, Holger Hermanns, Abhishek Dhama, and Oliver E. Theel. Symblicit calculation of long-run averages for concurrent probabilistic systems. In *QEST*, pages 27–36. IEEE Computer Society, 2010. *Cited in pages 7, 35, 41, 42, 115, 119, 123, 127, 155 and 157*

[WHT03] Nico Wallmeier, Patrick Hütten, and Wolfgang Thomas. Symbolic synthesis of finite-state controllers for request-response specifications. In Oscar H. Ibarra and Zhe Dang, editors, *CIAA*, volume 2759 of *Lecture Notes in Computer Science*, pages 11–22. Springer, 2003. *Cited in pages 35 and 40*

[WVS83] Pierre Wolper, Moshe Y. Vardi, and A. Prasad Sistla. Reasoning about infinite computation paths (extended abstract). In *FOCS*, pages 185–194. IEEE Computer Society, 1983. *Cited in page 17*

[Zer13] Ernst Zermelo. Über eine anwendung der mengenlehre auf die theorie des schachspiels. In *Proceedings of the fifth international congress of mathematicians*, volume 2, pages 501–504. Cambridge University Press, 1913. *Cited in page 4*

[ZP96] Uri Zwick and Mike Paterson. The complexity of mean payoff games on graphs. *Theor. Comput. Sci.*, 158(1&2):343–359, 1996. *Cited in pages 5 and 40*

Index