Automatic Verification of Competitive Stochastic Systems

by

Aistis Šimaitis

A thesis submitted for the degree of
Doctor of Philosophy

Magdalen College, Oxford
Michaelmas Term 2013

Abstract

In this thesis we present a framework for automatic formal analysis of competitive stochastic systems, such as sensor networks, decentralised resource management schemes or distributed user-centric environments. We model such systems as stochastic multi-player games, which are turn-based models where an action in each state is chosen by one of the players or according to a probability distribution. The specifications, such as "sensors 1 and 2 can collaborate to detect the target with probability 1, no matter what other sensors in the network do" or "the controller can ensure that the energy used is less than $75mJ$, and the algorithm terminates with probability at least 0.5", are provided as temporal logic formulae. We introduce a branching-time temporal logic rPATL and its multi-objective extension to specify such probabilistic and reward-based properties of stochastic multi-player games. We also provide algorithms for these logics that can either verify such properties against the model, providing a yes/no answer, or perform strategy synthesis by constructing the strategy for the players that satisfies the specification. We conduct a detailed complexity analysis of the model checking problem for rPATL and its multi-objective extension and provide efficient algorithms for verification and strategy synthesis. We also implement the proposed techniques in the PRISM-games tool and apply them to the analysis of several case studies of competitive stochastic systems.

Acknowledgements

Firstly, I would like to thank my supervisor, Marta Kwiatkowska, for the invaluable guidance, support and critique provided throughout my studies. I would also like to thank my mentors Vojtěch Forejt, David Parker and Taolue Chen – their support has been instrumental in completing this thesis and the number of lessons learnt from them about science, life and myself is difficult to count. My thanks also go to all the colleagues and collaborators inside and outside the department who made sure that my time spent working (and not) was very enjoyable. Clemens Wiltsche was especially helpful in the discussions about stochastic games in the last year. Finally, I would like to thank my family, especially parents, Rasa and Audrius, for supporting me throughout my education, and most importantly, my wife Elena, for all the patience, encouragement and motivation that she has provided me with.

Contents

1 Introduction **1**

2 Related Work **5**

2.1 Probabilistic model checking 5

2.2 Strategy synthesis and stochastic games 6

2.3 Multi-objective verification 8

2.4 Tools and applications 9

2.5 Summary 11

3 Background Material **13**

3.1 Probabilistic models 13

3.1.1 Discrete-time Markov chains 14

3.1.2 Markov decision processes 15

3.1.3 Stochastic games 18

3.2 Properties of probabilistic models 20

3.2.1 Problem definitions 20

3.2.2 Model checking properties of DTMCs 21

3.2.3 Model checking properties of MDPs 23

3.2.4 Model checking properties of stochastic games 28

3.3 Logics 31

3.4 Strategy models 36

3.5 Example 39

3.6 Summary 43

4 The Logic rPATL **45**

4.1 Syntax and semantics 46

4.2 Model checking 51

4.2.1 Model checking algorithm 52

4.2.2 Computation of probabilities 52

4.2.3 Computation of rewards 53
4.3 Complexity 63
4.4 Strategy synthesis 65
4.5 rPATL* 69
4.6 Reward-bounded properties 71
4.7 Summary 72

5 Multi-Objective rPATL 75

5.1 Syntax and semantics 76
5.2 Model checking 81
5.2.1 Reduction to maximisation of expected reward 82
5.2.2 Reduction to conjunctive rPATL 84
5.2.3 Computation of Pareto sets 88
5.3 Complexity 92
5.3.1 Determinacy and optimal strategies 92
5.3.2 Memory requirements 94
5.3.3 Complexity bounds 97
5.4 Strategy synthesis 104
5.5 Multi-objective rPATL* 108
5.6 Summary 111

6 Tool Implementation and Applications 113

6.1 Modelling language 114
6.2 Property specification 115
6.3 Model checking 116
6.4 Strategy synthesis 124
6.5 Experimental results 125
6.6 Case studies 128
6.6.1 Microgrid demand-side management 128
6.6.2 Collective decision making for sensor networks 132
6.6.3 Reputation protocol for user-centric networks 135
6.7 Summary 141

7 Conclusions 143

7.1 Summary and evaluation 143
7.2 Future work 144
7.3 Conclusion 146

A Comparison of strategy models **149**

B Proof of Theorem 8 **155**

C Proof of Theorem 12 **161**

D Proof of Theorem 15 **165**

E PRISM-games tool **171**

F PRISM-games models **175**

F.1 Team formation protocol 175

F.2 Microgrid demand-side management protocol 180

F.3 Collective decision making protocol for sensor networks 184

F.4 User-centric network protocol 189

List of Figures

3.1 A discrete-time Markov chain. 15
3.2 A Markov decision process. 17
3.3 A stochastic multi-player game. 19
3.4 Example DTMC. 22
3.5 Example MDP. 27
3.6 Example SMG. 30
3.7 A game demonstrating exponential memory requirement. 38
3.8 Experimental configurations of the agent organisations. 41
3.9 Performance comparison of the original algorithm with the optimal one. . . 42
3.10 Comparison of the optimal coalitions of varying sizes. 43

4.1 Running example: team formation. 49

5.1 Example Pareto sets. 80
5.2 Operations of the functional for conjunctive rPATL. 89
5.3 Examples of Pareto sets for conjunction and disjunction of objectives. . . . 90
5.4 Game demonstrating nondeterminacy. 93
5.5 Game demonstrating non-existence of optimal strategies. 93
5.6 Game in which coalition requires infinite memory to win. 95
5.7 Game illustrating PSPACE-hardness. 98
5.8 Gadgets encoding counter operations. 100
5.9 Game illustrating NP-hardness. 103
5.10 Strategy example. 107

6.1 A PRISM-games SMG model of a team formation game. 114
6.2 PRISM-games screenshots. 124
6.3 Performance of numerical computation algorithms (absolute difference convergence criterion). 126
6.4 Comparison of absolute and relative termination criteria for value iteration algorithms. 127

6.5 Energy demand curve and its piecewise approximation. 129
6.6 Results for demand-side management algorithm: value per household. . . . 131
6.7 Results for collective decision making algorithm: expected running time. . 134
6.8 Results for collective decision making algorithm: probability to recover. . . 135
6.9 Maximum unpaid services the requester can achieve in obtaining k services. 138
6.10 Cost of k services for requester and a strategy example. 139
6.11 Distribution of requests among providers. 140

E.1 PRISM-games model window. 171
E.2 PRISM-games properties window. 172
E.3 Strategy information window. 172
E.4 PRISM-games strategy import/export format. 173
E.5 PRISM-games simulator window. 174

List of Tables

3.1 Task completion probabilities for the agent organisations. 41

6.1 Performance statistics for PRISM-games. 125
6.2 SMG model sizes for microgrid demand-side management protocol. 130
6.3 Optimal values for P_{start} parameter. 130
6.4 SMG model sizes for sensor network protocol. 133
6.5 SMG model sizes for user-centric network protocol. 137

List of Algorithms

1 General model checking routine for rPATL. 117

2 Probability operator model checking routine (Next). 117

3 Probability operator model checking routine (Until). 118

4 Model checking routine for the reward operator. 119

5 Model checking routine for the F^c reward operator. 120

6 Model checking routine for F^∞ reward operator. 121

7 Model checking routine for F^0 reward operator. 122

8 Routine for computing value for R_i for $B \leq i \leq B{+}r_{\max}{-}1$. 123

9 Model checking routine for exact reachability probability. 123

Chapter 1

Introduction

Competitive stochastic systems. The trust placed in computers in contemporary society is high. We trust computers to pilot planes, control spaceships, perform surgeries, etc. There is one important aspect about these systems – they are provided to us as a *service*. We know that the responsibility for the quality and the consequences largely lies with the service providers, and therefore we expect them to have invested substantially to guarantee that the software, which is running these services, is of a high quality. Automatic verification of software, the method that automatically proves that a system conforms to its specification, is one of the ways to provide such guarantees. With the use of computer systems becoming ever more pervasive, the world of computing has been experiencing a paradigm shift from a centralised to a more distributed setting. Many distributed systems have emerged and are being used more and more often, including sensor networks, smart energy management, ad-hoc information sharing, and auction trading agents. Due to exposed privacy and close interaction with the systems, trust expectations for them are also very high and the designers will have to ensure that the quality standards meet those expectations. However, one key difference and challenge in designing and verifying such systems is that they are *open* – only certain parts of them could be under the control of the designer. Other parts may be controlled by different users or be executing without any awareness of the agents present in the system. Together they form an ecosystem of agents, each of which potentially representing their owners' interests and having different goals. We refer to such systems as *competitive stochastic systems*: *competitive* because each agent could have different or opposing goals and *stochastic* because algorithms which they execute or platforms in which they operate can be probabilistic.

Aims. Probabilistic verification tools such as PRISM [77], MRMC [72] and LiQuor [45] target a variety of formal stochastic models of systems, including Markov chains, Markov decision processes, and so on. But even though some of those models, e.g., Markov decision processes, allow specifying adversarial behaviour of one controller, they do not support the modelling of competition where several components are allowed to make decisions. On the other hand, there are methods and tools that support formal modelling and analysis of competitive systems, e.g., the MCMAS [85] model checker supports games and Alternating-time Temporal Logic (ATL) model checking, but do not allow to model the stochastic nature of the systems. The aim of this thesis is to provide a comprehensive framework for the *formal analysis* of algorithms for systems that exhibit both stochastic and competitive behaviour. We do this by jointly developing the theoretical underpinnings, including *property specification language*, together with *model checking and synthesis algorithms* and *implementation* of the techniques and their application to several relevant *case studies* of competitive stochastic systems.

Approach. As an underlying formal model we use stochastic multi-player games (SMGs), which are state transition systems in which, in each state, the choice of the transition is controlled either by one of the players in the game or is chosen probabilistically. SMGs can be seen as an extension of Markov decision processes (MDPs) to multiple players, and the existence of a large core of techniques for MDPs, including logics, algorithms, tools and applications, has motivated our choice of this particular extension.

The property specification language that we develop in this thesis builds on temporal logics such as CTL, LTL and ATL, as well as their probabilistic extensions, which have been used successfully to express properties of (stochastic) systems. The model checking algorithms that we designed to verify the properties are based on value iteration, which is a popular model checking technique in many modern probabilistic model checkers to achieve scalability and to verify large models.

The PRISM-games tool developed in this thesis is an extension of the PRISM model checker, taking advantage of its modelling language, user interface and other built-in functionality, while extending it to support stochastic games and strategy synthesis.

Contributions. The contributions of this thesis can be summarised as follows.

- We have introduced Probabilistic Alternating-time Temporal Logic with Rewards (rPATL), which combines features of popular temporal logics including ATL and PCTL and extends their support to SMGs. We perform a complexity analysis of rPATL model checking and develop model checking algorithms, which can be implemented via value iteration. In addition, we provide algorithms for strategy synthesis – construction of controllers satisfying the logic specifications – that perform strategy construction from the results of the value iteration algorithms.

- We have addressed the problem of multi-objective verification for stochastic games.

We analysed the complexity of the problem and showed that, even for the simple objectives like terminal state reachability and expected total reward, the model checking problem is considerably more difficult than the corresponding problems for MDPs. We have also extended rPATL to support the specification of multi-objective properties and provided approximation algorithms for this logic, which are based on the approximation of the Pareto sets (optimal trade-offs between the objectives). Similarly to rPATL, we have provided strategy construction methods.

- We have developed PRISM-games, a model checker that supports modelling of stochastic multi-player games, specification of rPATL properties and the algorithms for their model checking and synthesis. We have also applied the tool to the analysis of several case studies of competitive stochastic systems: team formation protocol, microgrid demand-side management algorithm, collective decision making algorithm for sensor networks, and a reputation mechanism for user-centric networks.

Thesis structure

The remainder of this thesis is structured as follows. Chapter 2 provides a technical overview of related work and discusses how the results of the thesis fit in and contribute to the research area. Background material is presented in Chapter 3. This includes basic concepts from probability theory and formal probabilistic models, together with discussion of common properties and their model checking. We also discuss different representation of player strategies and compare their expressiveness. Chapters 4, 5 and 6 contain the main contributions of the thesis. In Chapters 4 and 5 we focus on the property specification language defining the logic rPATL and its extension to multiple objectives. We also study the problem of model checking this logic on stochastic multi-player games: we perform its complexity analysis and design algorithms for model checking, as well as strategy synthesis. In Chapter 6 we present PRISM-games, a model checker for stochastic multi-player games, in which we implemented the rPATL model checking and synthesis techniques developed in the thesis. We also apply the tool to three case studies of competitive stochastic systems: decentralised microgrid energy management algorithm, collective decision making algorithm for sensor networks, and a reputation and virtual currency mechanism for user-centric networks. We conclude in Chapter 7 by evaluating the work done in the thesis and highlighting the research directions that are open for future work.

Publications

Some of the work in this thesis has been previously published in jointly authored papers. In [36, 37] the logic rPATL was introduced. I collaborated with the other authors to perform complexity analysis and the development of all model checking algorithms for the logic. These results are presented in Chapter 4. The majority of the work regarding modelling and analysis of the two case studies in the papers has been done by me. The implementation of rPATL model checking algorithms in the PRISM-games model checker [38] has been developed jointly by myself, David Parker and Vojtěch Forejt. Strategy synthesis functionality has been implemented by me. The contents of this work appear in Chapter 6.

In [40], the problem of model checking two-player stochastic games with multiple expected total reward objectives has been analysed. I collaborated with the other authors to perform complexity analysis and the development of approximation algorithms for the problem. In Chapter 5 we build on these results to provide a model checking algorithm for the multi-objective rPATL formulae.

The analysis of stochastic games with precise expectation objectives has been published in [39]. I contributed to all the results of the paper except for Section 5 on the counter-strategy problem. The results of this paper show the nondeterminacy and non-existence of optimal strategies for multi-objective rPATL specifications, described in Section 5.3 of this thesis, as well as exponential succinctness of stochastic-update strategies in Appendix A.

In [42] the application of synthesis for multi-objective stochastic games is presented. I contributed the strategy synthesis and LTL verification sections to the paper. Clemens Wiltsche has implemented the model checking and strategy synthesis algorithms for multi-objective expected total reward objectives. A variant of the strategy synthesis algorithm is presented in Section 5.4 of this thesis to synthesise strategies for multi-objective rPATL; and also, the multi-objective rPATL* model checking algorithm presented in Section 5.5 are based on the reduction appearing in [42] for multiple LTL objectives.

The case study of a team formation protocol has been published in [41]. I have performed the majority of the modelling and analysis work. David Parker has provided the prototype implementation of the stochastic game solver. The case study is presented in Section 3.5 of this thesis. Also, a variant of the protocol is used as a running example throughout the thesis.

The analysis of a reputation-based virtual currency mechanism for user-centric networks has been published in [81]. I have performed the modelling and analysis of the protocol. The case study is presented in Section 6.6.

Chapter 2

Related Work

In this chapter we give an overview of the research that is closely related to the work of this thesis: the design of a formal analysis framework and a verification tool for competitive stochastic systems. The chapter consists of four main sections. We start by discussing the probabilistic model checking techniques in Section 2.1; then, in Section 2.2, we discuss the strategy synthesis problem, placing particular emphasis on stochastic games and their use in verification. In Section 2.3, we review multi-objective verification techniques, and in Section 2.4 we discuss verification tools and several applications to the systems similar to the ones studied in this thesis.

2.1 Probabilistic model checking

Model checking [46, 97] is an automated technique to determine whether a model of a system satisfies a property specification, which is usually provided as a temporal logic formula. The approach works by exhaustively exploring the system's state transition graph and, in the case the property is not satisfied, returns a trace witnessing a faulty execution of the system – a counter-example to the property specification. Model checking algorithms for systems with properties expressed as temporal logics have been introduced in [47] for CTL (Computational Tree Logic) and in [103] for LTL (Linear Temporal Logic), which are able to express, respectively, branching and linear-time properties that system executions should satisfy.

The approach to model checking of systems against temporal logic specifications has been successfully extended to support probabilistic models, i.e., where the transitions between states of the model happen according to a probability distribution. There are two common approaches to verification of such systems: one is *qualitative*, where we are interested in checking if a temporal logic formula is satisfied with probability 1 or 0 in the model, the second approach is *quantitative*, where we are interested in computing the probability of the specification being satisfied. For the simplest of probabilistic models that we consider, namely, discrete-time Markov chains (DTMCs), where in all states transitions are taken probabilistically, qualitative model checking has been addressed in [84, 49], and, for Markov decision processes, where in some states the transition can happen nondeterministically, qualitative model checking has been studied in [95, 108, 49]. To address the quantitative model checking problem, the logic CTL has been extended with a probabilistic operator $\mathsf{P}_{>q}[\psi]$, which is true in a state of the model if the temporal formula ψ holds with probability greater than q. This gives rise to the logics PCTL [66] and pCTL [7] (the logics are of equivalent expressiveness and we refer to them as PCTL from hereon). The model checking of the aforementioned logics for DTMCs reduces to solving a system of linear equations, and for the extension PCTL* [7], where formula ψ can be an LTL formula, the solution of the method combines linear equation system solving with the techniques for qualitative properties [49] to verify the specifications. For MDPs, the PCTL model checking problem asks whether for all nondeterministic choices the formula holds with a given probability bound. The problem was considered in [14] and [50], and has been shown to be reducible to linear programming. [14] also presented the model checking algorithm for PCTL*.

Besides temporal logics, reward-based properties have been extensively used to specify quantitative properties of systems [96, 59, 52, 6, 58, 109]. Rewards label the states of the model with rational numbers representing rewards or costs incurred in that state. The execution of a system is now also a sequence of rewards as well as states. One of the most commonly studied properties for DTMCs and MDPs is a discounted total reward [96, 59, 52], which assigns a discounted sum of the rewards to the execution trace. Another common reward-based property is the limit average reward (also known as mean-payoff) [109, 34, 26], assigning the average of the encountered rewards to an execution trace. Traditionally, model checking of such properties concerns finding the *expected value* of the reward function. Finding such a value for the above functions can be reduced to solving the system of linear equations and linear programming for DTMCs and MDPs, respectively [96, 59]. Some recent work has also considered the *satisfaction problem*, which asks to compute the probability that the reward of the path exceeds a given bound [20].

2.2 Strategy synthesis and stochastic games

In the previous section we discussed probabilistic model checking, a method of formal verification that checks whether a given model of the system satisfies the property given as a temporal logic formula or expected reward objective. Another view one could take for the models having nondeterministic choices (e.g., MDPs) is to ask, given a property specification, to provide a controller (or strategy) that resolves the nondeterminism in a way that the given property specification is satisfied. We refer to this view as *strategy synthesis*. A strategy can be seen as a function mapping an execution history ending in a control state to the choice to be taken. Strategy synthesis for MDPs can be performed using the results of the model checking algorithms described in the previous section [14, 96] and has been applied in a variety of fields, including robotics [82], sensor grids [105], etc. Strategy synthesis for Markov decision processes has also been used as a mechanism design tool for multi-agent systems to synthesise the equilibria strategies [94].

Nondeterminism in MDPs is often used to represent the choices of the controller, whereas random choices model the stochastic environment. A natural extension of this model is where the behaviour of the environment is allowed to be both nondeterministic as well as random. The probabilistic model that takes into account the two sources of nondeterminism is stochastic two-player games introduced by [102]. The central computational problem in such games is the question of whether a player has a winning strategy. The games that we study in this thesis are zero-sum, i.e., the goals of the two players, the controller and the environment, are opposing: winning for one is losing for the other, and vice versa. Such games have been studied extensively in game theory [12], but found their way into computer science via modelling of *stochastic reactive systems* [98, 59], where researchers studied the synthesis of controllers with reachability and expected total reward objectives [48, 59]. The authors have shown there that memoryless deterministic controllers are sufficient to win the games for both players, and proved the problem of deciding the existence of such strategies to be in NP$\cap$coNP [48]. The exact complexity of such games has been a long-standing open question.

More recently, the study of stochastic two-player games has been extended to broader classes of objectives, including classical ω-regular specifications [106]. In [25] the authors addressed the qualitative and quantitative verification of games with Rabin and Streett objectives, showing that memoryless strategies are sufficient for the player to win the game with the former objective, and the problems of deciding the existence of a winning strategy are NP-complete and coNP-complete, respectively. However, for the parity winning objectives, just as for reachability and expected reward objectives, memoryless strategies are sufficient for both players, and the problem is in NP$\cap$coNP [33]. We also mention that ω-regular objectives have been studied for a more general class of concurrent stochastic games, where at each state both players have to take a decision concurrently [24].

There are several logics that have been defined for stochastic games. pATL (Probabilistic Alternating-time Temporal Logic) allows one to reason about concurrent stochastic multi-player games in a similar way that PCTL allows one to specify the properties of

MDPs, but model checking of the specifications reduces to solving concurrent stochastic two-player games [43, 22]. SGL (Stochastic Game Logic) [8] is a very powerful logic, proposed to specify the properties of turn-based stochastic multi-player games, but the model checking problem for it is undecidable in general [8], as is also the case for a variant of PCTL for stochastic games [19].

In addition to serving as an independent modelling framework, stochastic games have been instrumental as a solution method for other problems, e.g., abstraction refinement for MDPs [73] or model checking for probabilistic timed automata [76].

2.3 Multi-objective verification

When considering the synthesis problem for systems modelled as games or Markov decision processes, it is useful to have a specification consisting of several properties of different type. For example, one may be interested in obtaining a controller that ensures that a given LTL formula is satisfied while making sure that total reward is less than a given bound. Similar problems (sometimes referred to as multi-dimensional optimisation) have been studied in optimisation [54], engineering [86] and operations research [104] communities.

Most of the current work in multi-objective verification has been considered for non-stochastic two-player games. In [23], the authors present model checking algorithm for games, where one objective is a Büchi fairness constraint (repeated reachability of a certain set of states) and the second objective is that the resource consumption (total reward) of the system stays below a given threshold. In [31], the approach has been extended to a combination of parity and mean-payoff objectives, where the goal is to find the controller in a two-player game, which maximises the mean-payoff objective, while making sure that the parity condition (which is a canonical way to represent ω-regular specifications) holds.

In addition to combining different types of objectives, one may consider having several objectives of the same type. For the logics like CTL and LTL, to specify that the model satisfies two formulae one can simply take their conjunction and check the resulting single-objective property. However, this approach cannot be applied for the reward-based objectives, or for probabilistic extensions of logics such as PCTL. In [27, 109], the authors studied the problem of finding a controller optimising an objective that consists of several mean-payoff functions, and in [35] this work has been extended to also incorporate parity goals. In general, infinite memory strategies are required to win the games with the aforementioned objectives, but, for finite strategies, if such exist, exponential upper and lower bounds have been shown [35]. In [16], the authors take a different view, and instead of looking for a strategy in a game representing the system, they consider the synthesis of programs directly from LTL specifications under mean-payoff constraints, and two-player games are used as a tool to perform such synthesis.

For probabilistic models without nondeterminism (e.g., DTMCs), the multi-objective verification problem reduces to simply checking each objective individually. In [26], the authors studied the existence of a controller for Markov decision processes, which makes sure that a parity condition, and a mean-payoff objective are satisfied with probability 1, and showed that the problem can be solved in polynomial time. The work of [57] studied a more general problem of providing algorithms for verifying whether a boolean combination of ω-regular and expected total reward objectives can be achieved from a state of the MDP. For terminal state reachability and expected total reward objectives, the proposed algorithm works in polynomial time. The method developed in [57] has been applied to construct the assume-guarantee reasoning framework for MDPs [79].

The approach of [57] works by constructing a linear program to verify whether a specific bound for objectives can be obtained, i.e., whether one can answer questions like "does there exist a controller strategy which reaches state s with probability 0.5 and state t with probability 0.7?" Sometimes one may be interested in computing the set of all trade-offs between objectives that one can achieve (i.e., the Pareto set). This provides an alternative method to model check the multi-objective properties: one can compute the Pareto set and then check whether the required values are in this set, e.g., for the question above one would compute the Pareto set for the objectives "reach s" and "reach t" and then check whether the value pair $(0.5, 0.7)$ is in this set. An approximation algorithm for computing Pareto sets for MDPs has been presented in [61].

The authors of [21] study the problem of optimising both the mean-payoff objective and the stability of its value. Several notions of stability are proposed. One notion is the variance of the value of the mean-payoff objective; another one is the local variance, which considers how the rewards along the path differ from the mean-payoff of the path.

To the best of our knowledge, little work has been done to address the multi-objective verification for stochastic games. SGL [8] is able to express multi-objective ω-regular properties for SMGs, but model checking of the logic is undecidable. We also mention the work of [19] which considers the satisfiability of PCTL over MDPs, where the problem has been shown to be undecidable in general.

2.4 Tools and applications

We conclude the chapter by discussing several tools that support formal analysis and verification of systems, as well as applications of formal analysis to competitive stochastic systems.

MOCHA [5] was the first model checker to provide the model checking of ATL over concurrent multi-player games. As a modelling language, it uses a variant of the Reactive Modules formalism [3]. An extended version of the MOCHA modelling language is adopted by MCMAS [85], a model checker for multi-agent systems. MCMAS also supports the verification of ATL specifications, as well as various epistemic operators. Both MOCHA and MCMAS take advantage of symbolic representations, based on binary decision diagrams (BDDs), to achieve better scalability. The SPIN model checker [71], which supports verification of LTL properties for systems modelled in Promela [70], has been used to model check properties of multi-agent systems [18]. The technique performs a translation of the AgentSpeak [17] programming language into a Promela specification, which is then verified by SPIN. UPPAAL [13] is a tool focused on the analysis of real-time systems specified as timed-automata; the statistical model checking of hybrid models of systems is also supported.

Probabilistic model checker PRISM [77] supports the modelling and verification of various probabilistic models, including Markov chains and MDPs. The modelling language is a probabilistic variant of the Reactive Modules (see [93] for details) and the supported property specification language for these models is a variant of PCTL, extended with the reward operators [60], PCTL* and LTL. PRISM has several engines, including a symbolic one, which is based on multi-terminal BDDs. Modelling and verification of Markov chains and MDPs is also supported by several other model checkers. For example, in addition to verification of PCTL, the MRMC [72] model checker supports model checking of long-run average reward properties for several Markov models, but does not provide a high-level modelling language (models have to be constructed using PRISM or input as PEPA [69] specifications). The tool LiQuor [45] supports the verification of MDPs, modelled as Probmela programs [9], against LTL specifications.

There are several tools which support modelling of stochastic games. GIST [30] is a tool for the verification of qualitative ω-regular objectives for turn-based stochastic games containing two players, the system and the environment. If there exists no controller for the system achieving the objective, the tool can perform synthesis of the assumptions for the environment (i.e., restrictions for the strategies that the player can propose) that are sufficient for the system to achieve the objective. We also mention GAVS+ [44], an algorithmic game solver that supports, amongst other models, solution of MDPs and stochastic two-player games for reachability objectives.

We conclude the section by reviewing some applications of probabilistic model checking to the analysis of competitive stochastic systems. In [1], the authors use model checking of PCTL and reward-based properties over DMTCs and MDPs to analyse the trust-based reputation and virtual currency scheme for user-centric networks. The in-depth formal analysis of the trade-off between cooperation incentives and performance has been provided, and the possible improvements to the algorithm have been proposed. [11] presents an analysis of the alternating-offers negotiation protocol [91] for multi-agent systems. In the paper, the protocol is modelled as a DTMC by fixing the strategy profiles of the players (and hence making the behaviour of the system purely probabilistic). Using this approach, the authors were able to explore formally the effects that particular variables of the strategies have on the outcome of negotiation. In [88], authors present an analysis of the futures market investor modelled as a two-player stochastic game between the market and investor. Using MDP model checking, the authors explore various strategies available for investor against the worst-case scenarios (modelled as nondeterministic choices of the market). The results for the optimal strategies for the investor are obtained by solving a stochastic game.

2.5 Summary

In this chapter we have reviewed the work that is related to the research that we present in this thesis. In Section 2.1, we provided a general overview of the research area – probabilistic model checking. Then, in Section 2.2, we discussed stochastic games, which is the underlying formalism used in this thesis, together with some applications to strategy synthesis. This work is mostly related to Chapter 4. Section 2.3 presented an overview of the multi-objective verification, which is the focus of Chapter 5 addressing multi-objective verification for stochastic games. Finally, in Section 2.4, we discussed the tools available for formal modelling and analysis of systems and their applications to the analysis of competitive stochastic systems. The model checker PRISM-games, which we present in Chapter 6, contributes to this body of work by providing a modelling and verification tool for SMGs, which we then apply to analyse several case studies.

Chapter 3

Background Material

In this chapter we present the background material and known results that we use in the thesis. The chapter structure is as follows. In Section 3.1 we review the relevant probabilistic models (discrete-time Markov chains, Markov decision processes and stochastic games) as well as introduce the associated concepts. Then, in Section 3.2, we present the definitions, results and examples for the common properties that are analysed for those models: reachability, parity and expected total reward objectives. In Section 3.3 we discuss several temporal logics, which are used to express properties of probabilistic models and present their model checking algorithms. Then, in Section 3.4, we discuss the strategy representation models that we use for strategy synthesis. Finally, before summarising the material in Section 3.6, we present a case study to illustrate the application of the introduced probabilistic models and properties to the competitive stochastic systems in Section 3.5.

3.1 Probabilistic models

In this section we review probabilistic labelled state transitions systems commonly used for modelling and analysis of systems. We first introduce the notation that we use throughout. For an alphabet Σ we denote by Σ^* the set of all finite words formed from Σ, and similarly by Σ^ω we denote the set of all infinite words formed from the alphabet. Given two words, $\lambda \in \Sigma^*$ and $\lambda' \in \Sigma^* \cup \Sigma^\omega$, we denote by $\lambda \cdot \lambda'$ the concatenation of them. A *discrete probability distribution* (or just *distribution*) over a (countable) set X is a function $\mu : X \to [0,1]$ such that $\sum_{x \in X} \mu(x) = 1$. We write $\mathcal{D}(X)$ for the set of all probability distributions over X. Let $\mathsf{supp}(\mu) = \{x \in X \mid \mu(x) > 0\}$ be the *support set* of $\mu \in \mathcal{D}(X)$. We say that a distribution $\mu \in \mathcal{D}(X)$ is a *Dirac distribution* if $\mu(x) = 1$ for some $x \in X$. We represent a distribution $\mu \in \mathcal{D}(X)$ on a set $X = \{x_1, \ldots, x_n\}$ as a map $[x_1 \mapsto \mu(x_1), \ldots, x_n \mapsto \mu(x_n)]$

and omit the elements of X outside $\mathsf{supp}(\mu)$ to simplify the presentation. If the context is clear, we sometimes identify a Dirac distribution μ with the unique element in $\mathsf{supp}(\mu)$.

3.1.1 Discrete-time Markov chains

One of the simplest probabilistic models are *discrete-time Markov chains* (DTMCs), in which the transitions between states happen in a purely probabilistic way. They can be used to model systems, which evolve in a fully defined deterministic or probabilistic fashion; for example, DTMCs have been used to analyse Herman's self-stabilisation algorithm [78] for ring networks, the Bluetooth device discovery protocol [55], and many others.

Definition 1 (DTMC) *A* discrete-time Markov chain *(DTMC) is represented by a tuple $\mathcal{D} = \langle S, \Delta, AP, \chi \rangle$, where:*

- *S is a countable, non-empty set of* states*;*
- *$\Delta : S \times S \to [0, 1]$ is a* transition function*;*
- *AP is a finite set of* atomic propositions*; and*
- *$\chi : S \to 2^{AP}$ is a* labelling function*.*

An element $\Delta(s, t)$ of the transition function defines the probability of making a transition from state s to state t. Hence, we require that for all $s \in S$ we have $\sum_{t \in S} \Delta(s, t) = 1$. We assume that the transition function is defined for all states. We denote by $\Delta(s)$ the set of all successors that have positive probability to be reached in one step from s, i.e., $\Delta(s) \overset{\text{def}}{=} \{t \in S \mid \Delta(s, t) > 0\}$. States which have only one outgoing transition to themselves are called *terminal*; we denote the set of all terminal states by $\mathsf{Term} \overset{\text{def}}{=} \{s \in S \mid \Delta(s, s) = 1\}$. The labelling function assigns atomic propositions to states to label them with properties of interest.

An execution of the system is a sequence of states obtained by stepping through a DTMC according to a transition function Δ, which starts in some specified initial state. We refer to such a (possibly infinite) sequence as a *path* $\lambda = s_0 s_1 \ldots$, where $s_i \in S$ for all i and $s_j \in \Delta(s_{j-1})$. We use λ_j to denote the jth state in the path. Also, by $\mathrm{Suffix}(\lambda, j) = s_j s_{j+1} \ldots$ we denote the suffix of the path λ starting in state s_j, e.g., $\mathrm{Suffix}(\lambda, 1) = s_1 s_2 \cdots$; and by $\mathrm{Prefix}(\lambda, j) = s_0 s_1 \cdots s_{j-1}$ we denote the prefix of length j, e.g., $\mathrm{Prefix}(\lambda, 2) = s_0 s_1$. By $|\lambda|$ we denote the number of states in the path λ. The set of all infinite paths of $\mathcal{D}$ is denoted by $\Omega_{\mathcal{D}}$ and the set of infinite paths starting in state s is denoted by $\Omega_{\mathcal{D},s}$. Similarly, we use $\Omega^+_{\mathcal{D}}$ and $\Omega^+_{\mathcal{D},s}$ to denote the sets of finite paths. We often refer to a finite path of a DTMC as a *history*.

In order to reason about a system's behaviour, we need to be able to determine the probability of the paths in a DTMC. This is done in a standard way [74] (see also, e.g., [75, 60], for a more detailed discussion). For a state $s \in S$, the basic open sets of $\Omega_{\mathcal{D},s}$ are the *cylinder sets* $\mathrm{Cyl}(\lambda) \overset{\text{def}}{=} \lambda \cdot S^\omega$ for every finite path $\lambda = s_0 s_1 \ldots s_k$ of $\mathcal{D}$ and the probability assigned to $\mathrm{Cyl}(\lambda)$ is equal to $\prod_{i=1}^{k} \Delta(\lambda_{i-1}, \lambda_i)$. This definition induces a probability measure on the algebra of cylinder sets, which, by Carathéodory's extension theorem, can be extended to a unique probability measure on the σ-algebra generated by these sets. We denote the resulting probability measure by $\mathrm{Pr}_{\mathcal{D},s}$. When the DTMC is clear from the context, we simply refer to it by Pr_s.

A *bottom strongly connected component* (BSCC) of a DTMC $\mathcal{D}$ is a subset of states $S' \subseteq S$ such that, for all $s \in S'$ we have that if $\Delta(s,t) > 0$, then $t \in S'$, and for any pair of states $s, t \in S'$, there is a path from s to t and from t to s.

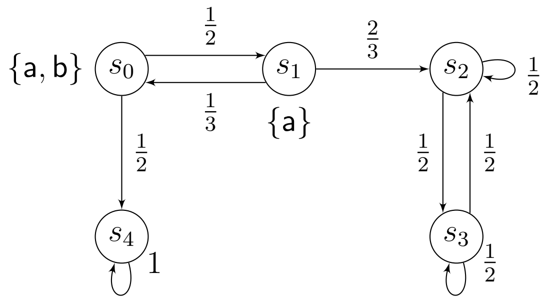


Figure 3.1: Example DTMC.

Example 1 *Consider the DTMC shown in Figure 3.1. The set of states of the model is $S = \{s_0, s_1, s_2, s_3, s_4\}$, the transition function for the state s_0 is $\Delta(s_0) = [s_1 \mapsto \frac{1}{2}, s_4 \mapsto \frac{1}{2}]$ and similarly for others. State s_0 is labelled with atomic propositions* a *and* b *($\chi(s_0) = \{\mathsf{a}, \mathsf{b}\}$), state s_1 is labelled with* a *and all others are labelled with none. An example of an infinite path through this DTMC is $s_0 s_1 s_0 (s_4)^\omega$. The probability of this path is $\frac{1}{2} \cdot \frac{1}{3} \cdot \frac{1}{2} = \frac{1}{12}$. This DTMC has two BSCCs, $\{s_4\}$ and $\{s_2, s_3\}$. The state s_4 is the only terminal state of the DTMC, i.e., $\mathsf{Term} = \{s_4\}$.*

3.1.2 Markov decision processes

The next type of model that we consider are *Markov decision processes* (MDPs), which generalise DTMCs by adding nondeterministic behaviour. In a state of an MDP a successor can now be chosen either according to a probability distribution (as in DTMCs) or nondeterministically. There are several ways that nondeterminism can be used to model the system. One example is where the nondeterministic states represent the control states of the model, i.e., states where the controller of the system is able to change the path of execution, e.g., an investor choosing whether to buy or sell the stock. Another common use is to model potential adversarial behaviour, where the states represent the environment of the system, the available actions of which are known, but the way that they are chosen is not, e.g., nondeterministic states could represent the possible actions of an adversary in a security protocol. MDPs have been applied to the analysis of such systems, for example, to analyse the strategies for futures market investor [88] and a probabilistic contract signing protocol [90].

Definition 2 (MDP) *A* Markov decision process *(MDP) is represented by a tuple* $\mathcal{M} = \langle S, (S_{\bigcirc}, S_{\Box}), \Delta, AP, \chi \rangle$*, where:*

- S *is a finite, non-empty set of* states*;*
- $S_{\bigcirc} \subseteq S$ *is the set of* stochastic states *and* $S_{\Box} \subseteq S$ *is the set of* control states*, such that* $(S_{\bigcirc}, S_{\Box})$ *is a partition of* S*;*
- $\Delta : S \times S \to [0, 1]$ *is a* transition function*;*
- AP *is a finite set of* atomic propositions*; and*
- $\chi : S \to 2^{AP}$ *is a* labelling function*.*

The key difference between DTMCs and MDPs lies in the transition function. If s is a stochastic state (i.e., $s \in S_{\bigcirc}$), then transition function behaves in the same way as for DTMCs, i.e., for each successor state it assigns the probability to move to it in the next step and $\sum_{t \in S} \Delta(s, t) = 1$. But if s is a control state (i.e., $s \in S_{\Box}$), then transition function simply indicates which successors are available to choose from in that state, i.e., for all $t \in S$ we have $\Delta(s, t) \in \{0, 1\}$, where 1 indicates that an action is available. We often refer to the choice of successor as an *action*. As for DTMCs, we assume that $\Delta(s)$ is not empty for all $s \in S$, and define terminal states as $\mathsf{Term} \overset{\text{def}}{=} \{s \in S \,|\, \Delta(s, s) = 1 \text{ and } \Delta(s, t) = 0 \text{ for } t \neq s\}$.

The definitions of paths through the MDPs are the same as for DTMCs, and we denote the set of all infinite paths in the MDP $\mathcal{M}$ by $\Omega_{\mathcal{M}}$, the set of infinite paths starting in a state s are denoted by $\Omega_{\mathcal{M},s}$, and we use $\Omega^{+}_{\mathcal{M}}$ and $\Omega^{+}_{\mathcal{M},s}$ for sets of finite paths. An execution of the system (that results in a path) in an MDP is a sequence of probabilistic (for $s \in S_{\bigcirc}$) and nondeterministic (for $s \in S_{\Box}$) choices. In order to define the probability measure for paths, we need to specify a way to resolve the nondeterministic choices. This is done by means of a *strategy.* A controller *strategy* σ is a function mapping from histories ending in a control state to a distribution on the available successors, i.e., $\sigma : S^{*}S_{\Box} \to \mathcal{D}(S)$. We denote the set of all controller strategies by Σ. An initial state s of an MDP $\mathcal{M}$ and a controller strategy σ together induce a (possibly infinite) DTMC, on which we can define a probability measure via cylinder sets as before, except the probability of the cylinder $\mathrm{Cyl}(\lambda)$ induced by a finite path λ of length $k+1$ is now equal to $\prod_{i=1}^{k} \Delta(\lambda_{i-1}, \lambda_i) \cdot p(i)$, where $p(i) = \sigma(\mathrm{Prefix}(\lambda, i))(\lambda_i)$ if $\lambda_{i-1} \in S_\Box$ and $p(i) = 1$ otherwise. We denote this probability measure by $\mathrm{Pr}^{\sigma}_{\mathcal{M},s}$, and, if σ and $\mathcal{M}$ are clear from the context, for clarity of presentation we use Pr_s.

Strategies are often classified based on their use of memory and randomisation. A strategy $\sigma \in \Sigma$ is called *memoryless* if $\sigma(\lambda{\cdot}s) = \sigma(\lambda'{\cdot}s)$ for all paths $\lambda{\cdot}s, \lambda'{\cdot}s \in \Omega^{+}_{\mathcal{M}}$, and *deterministic* if $\sigma(\lambda{\cdot}s)$ is a Dirac distribution for all $\lambda{\cdot}s \in \Omega^{+}_{\mathcal{M}}$.

An *end-component* of an MDP $\mathcal{M}$ is a subset of states $S' \subseteq S$ such that, if for all $s, t \in S'$ there exists strategy such that the probability to reach t from s is one, and the probability to reach s from t is one, i.e., $\exists \sigma \in \Sigma \,.\, \mathrm{Pr}_s(\{\lambda \in \Omega_{\mathcal{M}} \mid \lambda_i = t \text{ for some } i\}) = 1$ and $\exists \sigma \in \Sigma \,.\, \mathrm{Pr}_t(\{\lambda \in \Omega_{\mathcal{M}} \mid \lambda_i = s \text{ for some } i\}) = 1$. A *maximal end-component* is an end-component to which adding any other state would make it not an end-component. A *bottom end-component* is a maximal end-component that has no transitions leaving it.

Example 2 *Consider the MDP from Figure 3.2. It has four control states, s_0, s_2, s_3 and s_4, but only in two of them (s_0 and s_2) there are choices available for the controller. For the ease of presentation, we sometimes label transitions with action names, i.e., in s_0 there are two actions available: action "a" and action "b". Consider the memoryless*

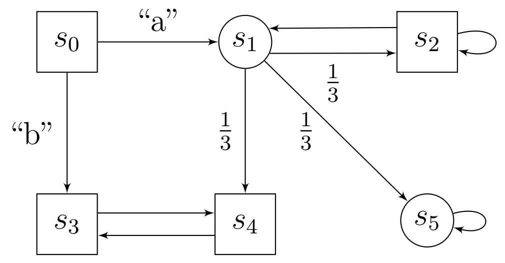


Figure 3.2: Example MDP.

deterministic controller strategy σ defined by $\sigma(\lambda \cdot s_0) = \sigma(\lambda \cdot s_2) = s_1$ for all $\lambda \in \Omega^{+}_{\mathcal{M}}$. An example of an infinite path through the MDP is $s_0 s_1 s_2 s_1 (s_4 s_3)^{\omega}$. The probability of this path under σ is $\frac{1}{9} = \sigma(s_0)(s_1) \cdot \Delta(s_1, s_2) \cdot \sigma(s_2)(s_1) \cdot \Delta(s_1, s_4) = 1 \cdot \frac{1}{3} \cdot 1 \cdot \frac{1}{3}$. This MDP has three end-components: $\{s_2\}$, $\{s_3, s_4\}$ and $\{s_5\}$ (all maximal). The last two are also bottom end-components. The only terminal state in the MDP is s_5.

3.1.3 Stochastic games

The final model that we introduce is *stochastic multi-player games* (SMGs), which generalise MDPs by differentiating between the sources of nondeterminism. Similarly to MDPs, in each state of an SMG the successor is either chosen randomly or nondeterministically, but this time the choice in different control states may belong to different controllers, i.e., players of the game. A special case of SMGs, stochastic two-player games, have been used for synthesis of systems (see, e.g., [30]), where control states of one player represent the actions available to the system and the control states of the other represent the environment, combining the two potential sources of nondeterminism that we discussed for MDPs into one game. The synthesis problem then asks to construct a strategy for the controller, such that, if implemented, ensures that the system satisfies a certain property, no matter what the environment does.

Definition 3 (SMG) *A* stochastic multi-player game *(SMG) is defined by a tuple* $\mathcal{G} = \langle \Pi, S, (S_{\bigcirc}, (S_i)_{i\in\Pi}), \Delta, AP, \chi\rangle$*, where:*

- Π *is a finite set of* players*;*
- S *is a finite, non-empty set of* states*;*
- $S_{\bigcirc}$ *is a set of* stochastic states *and every* S_i *in* $(S_i)_{i\in\Pi}$ *is the set of states* controlled *by the player* i *(also,* $(S_{\bigcirc}, (S_i)_{i\in\Pi})$ *is a partition of* S*);*
- $\Delta : S \times S \to [0,1]$ *is a* transition function*;*
- AP *is a finite set of* atomic propositions*; and*
- $\chi : S \to 2^{AP}$ *is a* labelling function*.*

The transition function behaves similarly as for MDPs: for stochastic states, it assigns the probability to move to the next state and we have $\sum_{t\in S} \Delta(s,t) = 1$ if $s \in S_{\bigcirc}$, and for control states we have $\Delta(s,t) \in \{0,1\}$ for $s \in \bigcup_{i\in\Pi} S_i$. As for the previous models, we assume that $\Delta(s)$ is defined for all $s \in S$, and define terminal states Term in the same way as for MDPs.

The executions of the system are modelled as paths of the game in the same way as for DTMCs and MDPs. The set of all infinite paths in $\mathcal{G}$ is denoted by $\Omega_{\mathcal{G}}$ and the set of infinite paths starting in state s by $\Omega_{\mathcal{G},s}$. Similarly, we use $\Omega^+_{\mathcal{G}}$ and $\Omega^+_{\mathcal{G},s}$ to denote the sets of finite paths. In order to define the probability measure on the induced DTMC, we need to have one strategy per player. The strategy σ_i for player i is a function $\sigma_i : S^* S_i \to \mathcal{D}(S)$. The probability of the cylinder set $\mathrm{Cyl}(\lambda)$ induced by a finite path λ of length k is defined as $\prod_{i=1}^{k} \Delta(\lambda_{i-1}, \lambda_i) \cdot p(i)$, where $p(i) = 1$ if $\lambda_i \in S_{\bigcirc}$ and $p(i) = \sigma_i(\text{Prefix}(\lambda, i))(\lambda_i)$ if $\lambda_{i-1} \in S_i$. We denote this probability measure by $\text{Pr}_{\mathcal{G},s}^{\sigma_0,\ldots,\sigma_n}$, where $\sigma_0, \ldots, \sigma_n$ are strategies of all players in the game (referred to as a *strategy profile*). If player strategies and the game are clear from the context, for clarity of presentation we use Pr_s.

A *subgame* is a game composed from a subset of states $S' \subseteq S$ such that there are no successors of stochastic states in S' leaving it, i.e., if $s \in S' \cap S_{\bigcirc}$ and $t \in \Delta(s)$ then $t \in S'$; the set of players, transition function, atomic propositions and labelling function are the same as for the original game.

A *stopping game* is a game $\mathcal{G}$ in which a terminal state is reached with probability 1 for any strategy profile from any state $s \in S$, i.e., for a game with $\Pi = \{0, \ldots, n\}$ we have $\forall s \in S \,.\, \forall \sigma_0, \ldots, \sigma_n \,.\, \text{Pr}_s(\{\lambda \in \Omega_s \,|\, \exists i \in \mathbb{N} \,.\, \lambda_i \in \mathsf{Term}\}) = 1$. Note that the game is stopping if and only if every subgame contains a terminal state of the original game. The class of stopping games is quite broad and subsumes important classes such as discounted games, games on trees and other acyclic graphs, etc.

Example 3 *Consider the SMG shown in Figure 3.3, with* $\Pi=\{\Box, \Diamond\}$. *The state controlled by a player has a corresponding shape, i.e.,* $S_\Box = \{s_0, s_2\}$ *and* $S_\Diamond = \{s_1\}$. *We have one stochastic state* $S_{\bigcirc} = \{s_3\}$. *State* s_2 *is labelled with atomic proposition* $\mathtt{t}$, *and all others are labelled with no atomic propositions. An example of an infinite path is* $\lambda =$

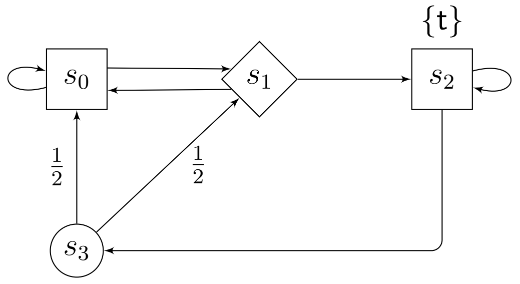


Figure 3.3: Example SMG.

$s_0 s_1 s_2 s_3 s_1 (s_2)^\omega$; *if player* $\Box$ *strategy* $\sigma_\Box$ *is such that* $\sigma_\Box(s_0)(s_1) = 0.3$, $\sigma_\Box(s_0 s_1 s_2)(s_3) = 1$ *and* $\sigma_\Box(s_0 s_1 s_2 s_3 s_1 (s_2)^+)(s_2) = 1$, *player* $\Diamond$ *strategy* $\sigma_\Diamond$ *is defined by* $\sigma_\Diamond(\lambda \cdot s_1) = s_2$ *for all* $\lambda \in \Omega_{\mathcal{G}}^+$, *we have that the probability of this path is* 0.15, *i.e.,* $\text{Pr}_{s_0}(\{s_0 s_1 s_2 s_3 s_1 (s_2)^\omega\}) = \sigma_\Box(s_0)(s_1) \cdot \sigma_\Diamond(s_0 s_1)(s_2) \cdot \sigma_\Box(s_0 s_1 s_2)(s_3) \cdot \Delta(s_3, s_1) \cdot \sigma_\Diamond(s_0 s_1 s_2 s_3 s_1)(s_2) \cdot \sigma_\Box(s_0 s_1 s_2 s_3 s_1 s_2)(s_2) \cdot \sigma_\Box(s_0 s_1 s_2 s_3 s_1 s_2 s_2)(s_2) \cdot \dots = 0.3 \cdot 1 \cdot 1 \cdot 0.5 \cdot 1 \cdot 1 \cdot 1 \cdot \dots = 0.15$. *Player* $\Box$ *strategy* $\sigma_\Box$ *uses both memory and randomisation, and player* $\Diamond$ *strategy* $\sigma_\Diamond$ *is memoryless deterministic. This game is not stopping and has no terminal states. Examples of subgames of* $\mathcal{G}$ *are* $\{s_0\}$ *and* $\{s_0, s_1, s_3\}$, *but* $\{s_0, s_2, s_3\}$ *is not a subgame.*

3.2 Properties of probabilistic models

In this section we provide an overview of behavioural properties that are commonly studied for the probabilistic models discussed in Section 3.1, and which are also a fundamental part of the algorithms for model checking specifications of competitive stochastic systems that we develop in this thesis. We consider three property types: reachability, parity and expected total reward objectives. We present the problem formulations, model checking methods and complexity results for DTMCs, MDPs and stochastic two-player games.

3.2.1 Problem definitions

We start by providing the problem definitions that are common for all model types presented in this section. We define the properties on the sample space of infinite paths Ω (for different model types this sample space is obtained in a different way as discussed in the previous section).

Reachability. This is the simplest class of properties for probabilistic models. The reachability problem asks to compute, for a given state, the probability to reach some state in a specified set of states in the model, i.e., given an initial state $s \in S$ and a set of target states $T \subseteq S$, the reachability probability is a measure of paths starting in s that eventually reach a state in T, i.e., $\mathrm{Pr}_s(\{\lambda \in \Omega \,|\, \exists i \in \mathbb{N} \,.\, \lambda_i \in T\})$. Reachability is one of the most fundamental properties in probabilistic model checking, and many model checking problems can be solved by using (or reducing to) reachability. For example, model checking of PCTL for DTMCs and MDPs uses reachability as a fundamental tool (see, e.g., [60] for an in-depth discussion).

Parity. The second property type that we consider for probabilistic models are parity objectives. Given a model and a parity function $P : S \to \{0, \ldots, d-1\}$ with d priorities assigning one to each state, a *parity objective* is the probability that a path through the model satisfies the parity condition, namely, that the lowest priority of the states occurring infinitely often in the path is even. Given a path $\lambda \in \Omega$, let $inf(\lambda)$ be the set of states that occur infinitely often in λ. Then, the probability that the parity objective is satisfied in a state s of the model is given by $\mathrm{Pr}_s(\{\lambda \in \Omega \,|\, \min_{s \in inf(\lambda)} P(s) \text{ is even}\})$.

Parity objectives can express any ω-regular property of the path of the model by encoding it in a deterministic parity automaton, which has states of the model as its alphabet. Then, the product of the model and the automaton can be built to obtain a model of the same class (i.e., the product of a DTMC and a deterministic parity automaton is a DTMC; the same also holds for MDPs and SMGs). In order to check whether a model satisfies an ω-regular property, it suffices to compute the probability of the parity objective in the product model. Computing the probabilities of ω-regular properties of paths is instrumental in model checking the logic PCTL* and verifying LTL properties. Also, some important path properties can be expressed directly using parity objectives, e.g., a Büchi objective (i.e., repeated reachability) can be expressed by constructing a priority function P which assigns priority 0 to the set of states that we want to visit infinitely often, and assigns priority 1 to all other states. Similarly, one could express a coBüchi (persistence) property by assigning 1 to states, which we do not want to see infinitely often, and 2 to all others. Similarly to reachability, the computation of Büchi and coBüchi objectives is a fundamental component of many model checking algorithms for more general property types.

Expected total reward. The third type of property that we consider here is the expected total reward. Here we augment the models with the *reward function*, assigning a non-negative reward to each state $r : S \to \mathbb{Q}_{\geq 0}$. The reward of the path is the sum of the rewards of the states on that path. The *expected total reward objective* asks to compute the expected reward accumulated along the path of the model, i.e., for a given state s, the expected total reward is $\mathbb{E}_s[rew(r)] = \int_{\Omega_s} rew(r) d\mathrm{Pr}_s$, where $rew(r)(\lambda) = \sum_{i=0}^{\infty} r(\lambda_i)$ for all $\lambda \in \Omega$.

Expected total reward objectives have been used extensively to model various properties of systems, with the most common properties being resource consumption and running time. For example, in [101] the authors analyse dynamic power management for priority queue systems using expected total rewards on DTMCs; also, in [80] the expected rewards were used to model running time and transmission costs in the IPv4 Zeroconf protocol modelled as an MDP.

3.2.2 Model checking properties of DTMCs

In this section we discuss model checking of the properties introduced in the previous section on DTMCs.

Reachability. For DTMCs, the reachability problem can be solved by finding a solution to a system of linear equations [84, 49]. Consider a DTMC $\mathcal{D} = \langle S, \Delta, AP, \chi \rangle$, where $S = \{s_0, \ldots, s_n\}$ and a set of target states is $T \subseteq S$, and let x_i represent the probability to reach a state in T from state $s_i \in S$. To compute the reachability probability we first need to identify the states $S_{=0}$ for which the probability x_i of reaching T is 0, which can be done using standard graph reachability. Then, the following system of equations provides the reachability probability for each state [84, 49]. For all states $s_i \in T$ we have $x_i = 1$, for states $s_i \in S_{=0}$ we have $x_i = 0$ and for all other states we have $x_i = \sum_j \Delta(s_i, s_j) \cdot x_j$. The values x_i can be computed in polynomial time by solving the system of linear equations (e.g., by Gaussian elimination). Iterative solution techniques, such as Jacobi or Gauss-Seidel iterations, can also be used, which generally achieve good scalability (this can be further improved by using symbolic methods to store the matrices, see [93] for further details).

Parity. For DTMCs, model checking of the parity objectives is performed in the following way [10]. First, using graph algorithms we can identify BSCCs in which the lowest priority is even. In such components, all sets of paths that have odd priority as the smallest priority have probability 0, and hence the parity objective holds with probability 1. Having identified all states belonging to such components, we can reduce the problem to that of finding the probability to reach a state in such a component (for which we can apply the algorithm presented in the previous section). Identification of BSCCs can be performed in polynomial time [52] and hence the model checking problem for parity objectives for DTMCs can be performed in polynomial time.

Expected total reward. Similarly to reachability objectives, the expected total reward for DTMCs can be computed by solving a system of linear equations. However, additional pre-processing is needed before this system of linear equations can be constructed. First, we need to identify BSCCs in which all states have reward 0, i.e., compute the set $S_{=0}$. In these states the expected total reward is 0. Next, we identify the BSCCs, which contain at least one state with positive reward. These states have infinite expected total reward, because the probability of the set of paths that do not visit the state having positive reward infinitely often is 0. We add the states belonging to such a component to the set $S_{=\infty}$. Also, we add to this set the states, which have a positive probability to reach a state in $S_{=\infty}$, as the expected reward in such states is infinity too. For all other states, the expected total reward value is finite and can be obtained by solving a system of linear equations constructed in the following way. Let x_i be a variable representing the expected total reward in a state s_i. If $s_i \in S_{=0}$, then $x_i = 0$. Otherwise, for $s_i \in S \setminus (S_{=0} \cup S_{=\infty})$ we have $x_i = r(s) + \sum_j \Delta(s_i, s_j) \cdot x_j$.

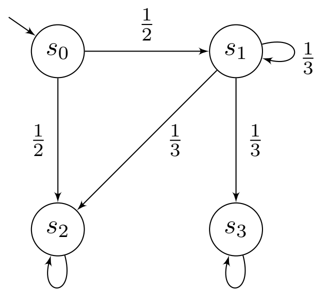


Figure 3.4: Example DTMC.

Example 4 *To illustrate the discussed properties, consider the DTMC from Figure 3.4. We show how to compute the probability to reach a state s_2. First, we identify the set $S_{=0} = \{s_3\}$, and then construct the system of linear equations, where x_i represents the probability to reach s_2 from s_i, as follows.*

$$\begin{aligned} x_0 &= \frac{1}{2} \cdot x_2 + \frac{1}{2} \cdot x_1 \\ x_1 &= \frac{1}{3} \cdot x_1 + \frac{1}{3} \cdot x_2 + \frac{1}{3} \cdot x_3 \\ x_2 &= 1 \\ x_3 &= 0. \end{aligned}$$

Values $x_0 = \frac{3}{4}$, $x_1 = \frac{1}{2}$, $x_2 = 1$ and $x_3 = 0$ are a unique solution and represent the reachability probabilities to s_2, e.g., $\mathrm{Pr}_{s_0}(\{\lambda \in \Omega_{s_0} \mid \exists i \in \mathbb{N} \,.\, \lambda_i = s_2\}) = x_0 = \frac{3}{4}$. Consider the reward function defined by $r(s_0) = r(s_1) = 1$ and $r(s_2) = r(s_3) = 0$. We have $S_{=0} = \{s_2, s_3\}$ and $S_{=\infty} = \emptyset$, and so the expected total reward in the states is given by the following system of linear equations.

$$\begin{aligned} x_0 &= 1 + \frac{1}{2} \cdot x_2 + \frac{1}{2} \cdot x_1 \\ x_1 &= 1 + \frac{1}{3} \cdot x_1 + \frac{1}{3} \cdot x_2 + \frac{1}{3} \cdot x_3 \\ x_2 &= 0 \\ x_3 &= 0. \end{aligned}$$

The unique solution $x_0 = 1.75$, $x_1 = 1.5$, $x_2 = 0$ and $x_3 = 0$ provides the expected total reward in each state, e.g., $\mathbb{E}_{s_1}[rew(r)] = x_1 = 1.5$. Now let us consider a parity objective. Define the priority function $P(s_i) = i$. There are two BSCCs in the DTMC, namely $\{s_2\}$ and $\{s_3\}$, and the probability that the parity objective holds is equal to reaching a BSCC where the even priority is lowest. Hence, it is equal to the probability to reach a state in $\{s_2\}$, which can be computed using the system of linear equations for reachability discussed previously, e.g., $\mathrm{Pr}_{s_1}(\{\lambda \in \Omega \mid \min_{s \in inf(\lambda)} P(s)$ is even$\}) = x_1 = \frac{1}{2}$).

3.2.3 Model checking properties of MDPs

For MDPs, the objectives are dependent on the choice of the strategy. As described in Section 3.1.2, fixing a strategy for an MDP, the resulting DTMC, and hence the above algorithm for DTMCs can be applied to compute the reachability probability. However, in MDPs we are often interested in finding the *optimal* (*minimum* or *maximum*) probability or expectation for a given objective. Memoryless deterministic strategies are sufficient for the controller to achieve optimal values for reachability, parity and expected total reward objectives; also, these values can be computed in polynomial time, as summarised by the following theorem.

Theorem 1 ([96, 59, 10, 63]) *In Markov decision processes, memoryless deterministic strategies are sufficient for the controller to achieve minimum and maximum values for reachability, parity and expected total reward objectives; these optimal values can be computed in polynomial time.*

We now discuss the algorithms for the computation of optimal values.

Reachability. The optimal reachability probabilities for MDPs can be computed in polynomial time using linear programming. The linear program (LP) for finding maximum probabilities can be constructed as follows (the LP for the minimum value can be constructed analogously). Consider an MDP $\mathcal{M} = \langle S, (S_{\bigcirc}, S_{\Box}), \Delta, AP, \chi \rangle$, where $S = \{s_0, \ldots, s_n\}$, and a set of target states is $T \subseteq S$. Let x_i denote the maximum probability of reaching T from s_i. We first use graph algorithms to identify states from which the target set is not reachable for any strategy, $S_{=0}$ and set $x_i = 0$. For states $s_i \in T$ we set $x_i = 1$. For the remaining states we add the following constraints. If $s_i \in S_{\bigcirc}$ then we add the constraint $x_i \geq \sum_j \Delta(s_i, s_j) \cdot x_j$. If $s_i \in S_{\Box}$ then we add the constraint $x_i \geq x_j$ for each successor $s_j \in \Delta(s_i)$. Finally, the objective function is $\min \sum_{s_i \in S} x_i$. The linear program can be solved in polynomial time (e.g., using the Ellipsoid or interior point methods [89]), in practice other algorithms like the Simplex method [89] perform better, even though they do not provide performance guarantees.

Parity. Similarly as for DTMCs, model checking of parity objectives in MDPs can be reduced to the reachability problem via computation of end-components, in which there exists a strategy for the parity objective to hold with probability 1 (resp. 0, if minimising). This can be done by computing the end-components containing the states having lowest even (resp. odd) priority [63] and then computing the maximum probability to reach a state in that set.

Expected total reward. For MDPs the expected total reward can be computed by using a combination of methods for solving reachability and parity objectives [96, 52]. We consider the maximisation problem (solution for minimisation works analogously). For every state that has a positive reward, we check if there exists a end-component that contains it. All states belonging to such a component are assigned the value of infinity and are added to the set $S_{=\infty}$. We also add to this set the states from which there exists a strategy to reach a state in $S_{=\infty}$ with positive probability (these can be identified by finding the maximum reachability probability using the methods described earlier).

For the remaining states, the expected total reward is finite, which means that, for any controller strategy, a state is reached that has expected total reward 0 for all strategies. We can identify such states efficiently by computing maximal end-components for each state having 0 reward; if all states in such a component have zero reward, then we add those states to $S_{=0}$. Note that, from any state, which is not in $S_{=\infty}$, the path ends up in a state in $S_{=0}$ with probability 1, and the expected total reward can be computed by solving the following linear program. As before, let x_i denote the expected total reward in state s_i. For each $s_i \in S_{=0}$ we have $x_i = 0$. For the remaining states we add the following constraints. If $s_i \in S_{\bigcirc}$ then we add the constraint $x_i \geq r(s_1) + \sum_j \Delta(s_i, s_j) \cdot x_j$. If $s_i \in S_{\Box}$ then we add the constraint $x_i \geq r(s_1) + x_j$ for each successor $s_j \in \Delta(s_i)$. The objective function is $\min \sum_{s_i \in S} x_i$. This linear program can be efficiently solved as discussed earlier.

Value iteration. There are other methods for computing the optimal reachability values in MDPs that are commonly used in practice. For example, for computing optimal reachability probabilities for MDPs, in addition to other methods, PRISM [77] implements the value iteration algorithm, which utilises the Bellman equations that can be derived from the above LP, e.g., for maximisation of the reachability probability we have that the Bellman operator for reachability is given by:

$$F(X)(s) = \begin{cases} 1 & \text{if } s \in T, \\ \sum_{t \in \Delta(s)} \Delta(s,t) \cdot X_t & \text{if } s \in S_{\bigcirc}, \\ \max_{t \in \Delta(s)} X_t & \text{otherwise.} \end{cases}$$

The iteration is performed starting from the initial value $X = 0^{|S|}$.

Value iteration can also be used to compute the optimal values for the expected total reward objectives for states with bounded values (i.e., states which are not in the set $S_{=\infty}$). The Bellman functional for the maximisation of the expected total reward is:

$$F(X)(s) = \begin{cases} r(s) + \sum_{t \in \Delta(s)} \Delta(s,t) \cdot X_t & \text{if } s \in S_{\bigcirc}, \\ r(s) + \max_{t \in \Delta(s)} X_t & \text{otherwise.} \end{cases}$$

As for reachability, the iteration is started with the initial value $X = 0^{|S|}$.

The best known bounds on convergence of the value iteration algorithms defined above are exponential in the number of states of the MDP, and, in practice, the iteration is terminated earlier with some convergence check (e.g., terminate if $\max_{s \in S} |\frac{X_s^k - X_s^{k-1}}{X_s^k}| < \epsilon$). The ease of implementation and scalability outweighs the drawbacks of value iteration and in many practical case studies value iteration has produced good results and was able to solve models with state spaces that are hard to handle for LP solvers. We also note that DTMC is a special case of MDP with no control states, an the above value iteration algorithms can be used to model check properties for DTMCs as well. For an in-depth discussion about the value iteration algorithms we refer the reader to [29].

Multiple objectives. All of the model checking problems that we considered so far deal with single-objective verification, but often the specification of the system contains several objectives (i.e., multi-objective verification). For DTMCs, problems reduce to checking each of the properties individually. However, this approach cannot be applied to MDPs, because, dependent on the strategy, the values for the objectives can be different. So, for MDPs we are interested in finding out if the controller can achieve a certain trade-off between objectives, e.g., in a given MDP, does there exist a strategy for the controller, that makes sure that the probability to reach a set T is greater than x, and at the same time expected total reward is less than or equal to y. Such problem can be solved using linear programming [57], and, for terminal state reachability and (finite) expected total reward objectives, the program is of polynomial size. Also, memoryless randomised strategies are sufficient to achieve such objectives.

Theorem 2 ([57]) *Memoryless randomised strategies are sufficient for the controller to achieve a boolean combination of terminal state reachability and finite expected total reward objectives in Markov decision processes; and the problem of deciding whether there exists such a strategy can be solved in polynomial time.*

Value iteration techniques have also been developed for the computation of the trade-offs achievable by the controller in MDPs [61]. These are based on the computation of the Pareto sets (i.e., the sets of optimal trade-offs between objectives for which there exists a controller strategy). The question of whether a combination of objectives can be achieved then reduces to the problem of checking whether the objective belongs to the Pareto set. In [61] the authors provide techniques to efficiently compute approximations of the Pareto sets utilising the value iteration algorithms for single-objective probability and expected total reward objectives. The algorithm takes advantage of the result of [57] showing that a conjunction of (maximisation) objectives is achievable if and only if there exists a strategy, which achieves a certain single-objective expected reward, which is obtained as a weighted sum of the original objectives.

Example 5 *To illustrate the properties discussed, we consider the example MDP from Figure 3.5. We show how to compute the minimum and maximum probabilities to reach a state s_2. First, we identify the set $S_{=0} = \{s_3\}$, and then construct the following linear*

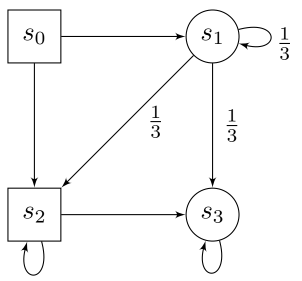


Figure 3.5: Example MDP.

program for finding the maximum reachability probability to s_2.

$$\begin{array}{rcl} \min & & \sum_i x_i \\ x_0 & \geq & x_1 \\ x_0 & \geq & x_2 \\ x_1 & = & \frac{1}{3} \cdot x_1 + \frac{1}{3} \cdot x_2 + \frac{1}{3} \cdot x_3 \\ x_2 & = & 1 \\ x_3 & = & 0\,. \end{array}$$

The solution to the linear program is $x_0 = x_2 = 1$, $x_1 = \frac{1}{2}$ and $x_3 = 0$. We can also extract the optimal strategy from the solution, which picks successor s_2 in s_0 achieving $\Pr_{s_0}(\{\lambda \in \Omega_{s_0} \mid \exists i \in \mathbb{N} \,.\, \lambda_i = s_2\}) = x_0 = 1$. To compute the minimum probability to reach s_2 we have to solve the following linear program.

$$\begin{array}{rcl} \max & & \sum_i x_i \\ x_0 & \leq & x_1 \\ x_0 & \leq & x_2 \\ x_1 & = & \frac{1}{3} \cdot x_1 + \frac{1}{3} \cdot x_2 + \frac{1}{3} \cdot x_3 \\ x_2 & = & 1 \\ x_3 & = & 0\,. \end{array}$$

The solution is $x_0 = x_1 = \frac{1}{2}$, $x_2 = 1$ and $x_3 = 0$. The minimising strategy in s_0 is to pick s_1 as successor achieving $\Pr_{s_0}(\{\lambda \in \Omega_{s_0} \mid \exists i \in \mathbb{N} \,.\, \lambda_i = s_2\}) = x_0 = \frac{1}{2}$.

Let us now consider a parity objective with priority function $P(s_i) = i$. To compute the maximum probability to satisfy this parity objective, we first identify the set of states from which there is a strategy to win with probability 1, $S_{=1} = \{s_0, s_2\}$, and for the remaining states we compute the maximum probability to reach a state in $S_{=1}$. Because probability *to reach s_0 is 0 from all other states, we can use the solution to the first linear program to give us the answer, e.g.,* $\Pr_{s_1}(\{\lambda \in \Omega \mid \min_{s \in inf(\lambda)} P(s) \text{ is even}\} = x_1 = \frac{1}{2})$.

Now let us consider the reward function defined by $r(s_0) = 1$, $r(s_1) = 1$, $r(s_2) = 2$ and $r(s_3) = 0$. To compute the maximum expected total reward we first solve the parity game where the states having positive reward are given priority 0, and others priority 1, i.e., $P(s_0) = P(s_1) = P(s_2) = 0$ and $P(s_3) = 1$. We now compute the states, in which the maximum winning probability is positive. Note that the probabilities to satisfy this parity objective coincide with the one discussed in the previous objective, and hence we have that in states s_0, s_1 and s_2 we can achieve infinite reward (i.e., $S_{=\infty} = \{s_0, s_1, s_2\}$) this is because in s_2 the strategy can always pick itself as a successor to accumulate infinite reward. For the case of minimisation we identify the states in which minimum expected reward is 0, $S_{=0} = \{s_3\}$. Now we need to solve the following linear program.

$$\begin{array}{rcl}
\max & & \sum_i x_i \\
x_0 & \leq & 1 + x_1 \\
x_0 & \leq & 1 + x_2 \\
x_1 & = & 1 + \dfrac{1}{3} \cdot x_1 + \dfrac{1}{3} \cdot x_2 + \dfrac{1}{3} \cdot x_3 \\
x_2 & \leq & 2 + x_2 \\
x_2 & \leq & 2 + x_3 \\
x_3 & = & 0\,.
\end{array}$$

The solution $x_0 = 3$, $x_1 = 2.5$ $x_2 = 2$, $x_3 = 0$ provides the minimum expected rewards that can be achieved in the respective states, e.g., $\mathbb{E}_{s_0}[rew(r)] = x_0 = 3$. The optimal strategy is to pick s_2 as successor in s_0 and s_3 as successor in s_2.

3.2.4 Model checking properties of stochastic games

Finally, we discuss model checking of properties for stochastic two-player games. The value of the objective in stochastic games depends on the strategies of both players. Here we consider that the game is zero-sum, i.e., the objectives of the players are opposing, that is, one player ($\Box$) is trying to maximise the probability to satisfy the objective and another player ($\Diamond$) is minimising it. Similarly to MDPs, we are interested in finding the *optimal* values for the objective, but in this case those values correspond to both players playing their optimal strategies – maximising and minimising, respectively. An important result for stochastic two-player games is that they are determined for the objectives considered in this section.

Theorem 3 ([87]) *Stochastic two-player games with reachability, parity and expected total reward objectives are determined.*

Determinacy means that, for all states s of the game we have

$$\sup_{\sigma_\Box \in \Sigma_\Box} \inf_{\sigma_\Diamond \in \Sigma_\Diamond} \mathrm{Pr}_s(\Theta) = \inf_{\sigma_\Diamond \in \Sigma_\Diamond} \sup_{\sigma_\Box \in \Sigma_\Box} \mathrm{Pr}_s(\Theta),$$

where Θ denotes the set of paths satisfying the (reachability or parity) objective. Similarly, for expected total reward we have

$$\sup_{\sigma_\Box \in \Sigma_\Box} \inf_{\sigma_\Diamond \in \Sigma_\Diamond} \mathbb{E}_s[rew(r)] = \inf_{\sigma_\Diamond \in \Sigma_\Diamond} \sup_{\sigma_\Box \in \Sigma_\Box} \mathbb{E}_s[rew(r)].$$

An important consequence is that the maximum value for the objective for one player is the minimal value for another, which allows us to solve minimisation problems with algorithms that compute the maximum value by swapping players in the game (and vice versa).

The next important result for games with the types of objectives discussed here is that memoryless deterministic strategies are sufficient for both players to achieve optimal values.

Theorem 4 ([48, 59, 33]) *Memoryless deterministic strategies are sufficient for achieving optimal values for both players in zero-sum stochastic two-player games with reachability, parity and expected total reward objectives.*

The existence of a polynomial-time algorithm for the computation of the optimal values for such games has been an open problem even for reachability objectives. In order to decide whether the optimal value exceeds a certain given bound, one can use Theorem 4 with the determinacy result to guess a memoryless deterministic strategy for one of the players (depending on whether the answer is yes or no), and then verify the answer in polynomial time by computing the optimal value in the resulting MDP in polynomial time. This procedure gives an NP$\cap$coNP upper bound for the complexity of the problem [48]. It also provides an exponential algorithm to compute optimal values by enumerating memoryless deterministic strategies.

Value iteration. Similarly to MDPs, we define Bellman functionals that are used to obtain value iteration algorithms for computing successive approximations of the optimal values. The optimal reachability probabilities for target set T can be computed using the following Bellman operator:

$$F(X)(s) = \begin{cases} 1 & \text{if } s \in T \\ \sum_{t\in\Delta(s)} \Delta(s,t)\cdot X_t & \text{if } s \in S_{\bigcirc}, \\ \max_{t\in\Delta(s)} X_t & \text{if } s \in S_{\Box}, \\ \min_{t\in\Delta(s)} X_t & \text{if } s \in S_{\Diamond}. \end{cases}$$

Starting from the initial value $X = 0^{|S|}$, the iteration of this function is guaranteed to converge in the number of steps that is exponential in the size of the game [29].

For the expected total reward, we define the operator:

$$F(X)(s) = \begin{cases} r(s) + \sum_{t\in\Delta(s)} \Delta(s,t)\cdot X_t & \text{if } s \in S_{\bigcirc}, \\ r(s) + \max_{t\in\Delta(s)} X_t & \text{if } s \in S_{\Box}, \\ r(s) + \min_{t\in\Delta(s)} X_t & \text{if } s \in S_{\Diamond}. \end{cases} \tag{3.1}$$

The iteration, when starting from the initial value $X = 0^{|S|}$, converges in a similar way as for reachability [29], but only in the case where the expected total rewards are bounded for all states. This is true for the class of *stopping games* where $r(t) = 0$ for all $t \in \mathsf{Term}$. For non-stopping games, the states that receive infinite reward ($S_{=\infty}$) can be identified by solving a parity game as follows. Assign parity 0 to all states that have positive reward and parity 1 to states that have reward 0. If in a state s player $\Box$ has a strategy to make sure that this parity objective holds with positive probability, then $s \in S_{=\infty}$, and otherwise the expected total reward is bounded, because positive rewards appear on the path only finitely many times. After removing theses states, value iteration algorithm can be applied for the remaining states.

Example 6 *To illustrate the properties discussed above, we consider the game in Figure 3.6. To achieve the maximum reachability probability to s_3, player $\Box$ has to choose*

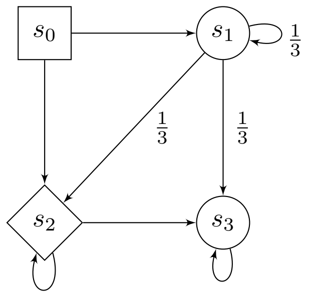


Figure 3.6: Example SMG.

s_1 as successor in s_0, and player $\Diamond$ chooses to stay in s_2 ensuring that the probability to reach s_3 from s_2 is 0. This results in $\sup_{\sigma_\Box \in \Sigma_\Box} \inf_{\sigma_\Diamond \in \Sigma_\Diamond} \Pr_{s_0}(\{\lambda \in \Omega_{s_0} \mid \exists i \in \mathbb{N} \,.\, \lambda_i = s_3\}) = \frac{1}{2}$. The minimum probability to reach s_3 that player $\Box$ can ensure is 1: we have that $\inf_{\sigma_\Box \in \Sigma_\Box} \sup_{\sigma_\Diamond \in \Sigma_\Diamond} \Pr_s(\{\lambda \in \Omega_s \mid \exists i \in \mathbb{N} \,.\, \lambda_i = s_3\}) = 1$ for all s, because the optimal strategy for player $\Diamond$ is to take the action leading to s_3 in s_2 and for any player $\Box$ strategy the game ends either in s_2 or s_3.

For an example of the parity objective consider priority function $P(s_i) = i$. We have that $\sup_{\sigma_\Box \in \Sigma_\Box} \inf_{\sigma_\Diamond \in \Sigma_\Diamond} \Pr_{s_0}(\{\lambda \in \Omega_{s_0} \mid \min_{s \in inf(\lambda)} P(s) \text{ is even}\}) = 0$, because player $\Diamond$ can ensure that the game ends in state s_3 with probability 1 by picking s_3 as successor in s_2; and $\inf_{\sigma_\Box \in \Sigma_\Box} \sup_{\sigma_\Diamond \in \Sigma_\Diamond} \Pr_{s_0}(\{\lambda \in \Omega_{s_0} \mid \min_{s \in inf(\lambda)} P(s) \text{ is even}\}) = \frac{1}{2}$, because player $\Box$ can choose s_1 as successor in s_0, and, even though player $\Diamond$ makes the game stay in s_2 if this state is visited, with probability $\frac{1}{2}$ the game ends in s_3, from where the parity objective is not satisfied.

Finally, we illustrate how one can use value iteration to compute the successive approximations of the expected total reward that can be achieved in the game. Consider the reward function defined by $r(s_0) = 1$, $r(s_1) = 2$, $r(s_2) = 3$, and $r(s_3) = 0$. It is easy to see that the minimum reward that player $\Box$ can guarantee is infinity in all states except for s_3 where the expected reward is always zero (it is because the probability to reach s_2 is positive from all states other than s_3, and player $\Diamond$ can choose to always loop in s_2 to accumulate infinite reward). Computing the maximum expected reward that player $\Box$ can guarantee using the value iteration defined above leads to the following sequence of values. We start from the initial value $X^0 = (X^0_{s_0}, X^0_{s_1}, X^0_{s_2}, X^0_{s_3}) = (0, 0, 0, 0)$, then apply one step of equations (3.1) to obtain $F(X^0) = (1, 2, 3, 0)$, $F^2(X^0) = (4, \frac{11}{3}, 3, 0)$, $F^3(X^0) = (\frac{14}{3}, \frac{31}{9}, 3, 0)$, etc. This sequence converges to $F^\infty(X^0) = (\frac{11}{2}, \frac{9}{2}, 3, 0)$, which is the least fixpoint of the functional F, representing the maximum expected total reward that player $\Box$ can guarantee.

3.3 Logics

In this section we present the logics PCTL and pATL that are probabilistic extensions of the logics CTL and ATL, respectively.

PCTL. We provide the semantics for the logic for MDPs [14] extended with the reward operator of [75]. Throughout this section we fix an MDP $\mathcal{M} = \langle S, (S_\bigcirc, S_\Box), \Delta, AP, \chi \rangle$.

The syntax of the logic PCTL is defined by the following grammar:

$$\begin{aligned} \phi &::= \top \mid a \mid \neg\phi \mid \phi \wedge \phi \mid \mathsf{P}_{\bowtie p}[\psi] \mid \mathsf{R}^r_{\bowtie q}[\mathsf{F}\,\phi] \\ \psi &::= \mathsf{X}\,\phi \mid \phi\,\mathsf{U}\,\phi, \end{aligned}$$

where $a \in AP$, $\bowtie \in \{<, \leq, \geq, >\}$, $p, q \in \mathbb{Q} \cap [0,1]$, $r : S \to \mathbb{Q}_{\geq 0}$.

PCTL is a branching-time temporal logic, where the formulae are split into state (ϕ) and path (ψ) formulae. We can also derive other standard temporal operators like $\mathsf{F}\,\phi \equiv \top\,\mathsf{U}\,\phi$, etc. For a state $s \in S$, we write $s \models \phi$ to denote that a state satisfies the state formula ϕ, similarly we denote that a path $\lambda \in \Omega_{\mathcal{M}}$ satisfies path formula ψ by $\lambda \models \psi$. The satisfaction relation $\models$ for PCTL is defined inductively for all states of $\mathcal{M}$ as follows.

$$\begin{array}{lcl} s \models \top & & \text{always} \\ s \models a & \Leftrightarrow & a \in \chi(s) \\ s \models \neg\phi & \Leftrightarrow & s \not\models \phi \\ s \models \phi_1 \wedge \phi_2 & \Leftrightarrow & s \models \phi_1 \text{ and } s \models \phi_2 \\ s \models \mathsf{P}_{\bowtie p}[\psi] & \Leftrightarrow & \forall \sigma \in \Sigma \,.\, \mathrm{Pr}_s(\psi) \bowtie p \\ s \models \mathsf{R}^r_{\bowtie q}[\mathsf{F}\,\phi] & \Leftrightarrow & \forall \sigma \in \Sigma \,.\, \mathbb{E}_s[f(r, \mathit{Sat}(\phi))] \bowtie q \end{array}$$

and for any path λ in $\Omega_{\mathcal{M}}$:

$$\begin{array}{lcl} \lambda \models \mathsf{X}\,\phi & \Leftrightarrow & \lambda_1 \models \phi \\ \lambda \models \phi_1\,\mathsf{U}\,\phi_2 & \Leftrightarrow & \lambda_i \models \phi_2 \text{ for some } i \in \mathbb{N} \text{ and } \lambda_j \models \phi_1 \text{ for } 0 \leq j < i. \end{array}$$

In the above definition, by $\mathrm{Pr}_s(\psi)$ we denote the probability of the paths satisfying ψ, i.e., $\mathrm{Pr}_s(\psi) \overset{\text{def}}{=} \mathrm{Pr}_s(\{\lambda \in \Omega_{\mathcal{M}} \mid \lambda \models \psi\})$. Also, by $\mathit{Sat}(\phi) \overset{\text{def}}{=} \{s \in S \mid s \models \phi\}$ we denote the set of states that satisfy formula ϕ. The reward function $f : \Omega_{\mathcal{M}} \times 2^S \to \mathbb{Q}_{\geq 0} \cup \{\infty\}$ is defined as follows:

$$f(\lambda, T) = \begin{cases} \infty & \text{if } \forall i \in \mathbb{N} \,.\, \lambda_i \notin T, \\ \sum_{i=0}^{k-1} r(\lambda_i) & \text{otherwise, where } k = \min\{j \mid \lambda_j \in T\}. \end{cases}$$

The main motivation for the above reward function is to analyse properties like the expected running time until algorithm's completion. For example, when considering the reward structure $r(s) = 1$ for all $s \in S$ representing time-steps, the PCTL formula $\mathsf{R}^r_{\leq 100}[\mathsf{F}\,\mathsf{finished}]$ is true in all states, from which the algorithm always reaches a state labelled with $\mathsf{finished}$, and, in addition, for all controller strategies the expected number of steps taken before reaching such a state is less or equal to 100.

The algorithm for PCTL model checking follows the standard structure for branching-time logics like CTL, by recursively computing the truth-value for states, starting from the most deeply nested sub-formulae and following the recursive satisfaction relation described above [75]. Computation of atomic propositions and boolean combinations is straightforward. To evaluate the operator $\mathsf{P}_{\bowtie p}[\psi]$ we take advantage of the properties

$$\mathsf{P}_{\bowtie p}[\psi] \Leftrightarrow \inf_{\sigma\in\Sigma} \mathrm{Pr}_s(\psi) \bowtie p \quad \text{if } \bowtie \in \{\geq, >\}$$

and

$$\mathsf{P}_{\bowtie p}[\psi] \Leftrightarrow \sup_{\sigma\in\Sigma} \mathrm{Pr}_s(\psi) \bowtie p \quad \text{if } \bowtie \in \{\leq, <\},$$

which imply that it suffices to compute the optimal values in the MDP and compare them to the bound. Their computation when $\psi = \mathsf{X}\,\phi$ is performed as follows. Assuming that the set of states satisfying a formula ϕ, denoted $\mathit{Sat}(\phi)$, has already been computed, we have

$$\inf_{\sigma\in\Sigma} \mathrm{Pr}_s(\mathsf{X}\,\phi) = \begin{cases} 0 & \text{if } s \in S_\Box \text{ and } \Delta(s) \setminus \mathit{Sat}(\phi) \neq \emptyset, \\ \sum_{t\in\Delta(s)\cap \mathit{Sat}(\phi)} \Delta(s,t) & \text{if } s \in S_\bigcirc, \\ 1 & \text{otherwise,} \end{cases}$$

and similarly

$$\sup_{\sigma\in\Sigma} \mathrm{Pr}_s(\mathsf{X}\,\phi) = \begin{cases} 1 & \text{if } s \in S_\Box \text{ and } \Delta(s) \cap \mathit{Sat}(\phi) \neq \emptyset, \\ \sum_{t\in\Delta(s)\cap \mathit{Sat}(\phi)} \Delta(s,t) & \text{if } s \in S_\bigcirc, \\ 0 & \text{otherwise.} \end{cases}$$

The computation of optimal values for $\mathrm{Pr}_s(\phi_1 \,\mathsf{U}\, \phi_2)$ reduces to the computation of optimal reachability probabilities discussed in Section 3.2.3, and can be solved using linear programming or computed using value iteration. The reduction works as follows. For all states $s \notin \mathit{Sat}(\phi_1) \cup \mathit{Sat}(\phi_2)$ we remove all transitions leaving the state and add a self-loop, and for the remaining states we compute the optimal (minimum or maximum depending on $\bowtie$) probability to reach a state in $\mathit{Sat}(\phi_2)$. Finally, the computation of $\mathsf{R}^r_{\bowtie q}[\mathsf{F}\,\phi]$ reduces to the computation of the optimal values for the expected total reward as follows. First note that we have

$$\mathsf{R}^r_{\bowtie q}[\mathsf{F}\,\phi] \Leftrightarrow \inf_{\sigma\in\Sigma} \mathbb{E}_s[f(r, \mathit{Sat}(\phi))] \bowtie q \quad \text{if } \bowtie \in \{\geq, >\}$$

and

$$\mathsf{R}^r_{\bowtie q}[\mathsf{F}\,\phi] \Leftrightarrow \sup_{\sigma\in\Sigma} \mathbb{E}_s[f(r, \mathit{Sat}(\phi))] \bowtie q \quad \text{if } \bowtie \in \{\leq, <\}.$$

To compute $\inf_{\sigma\in\Sigma} \mathbb{E}_s[f(r, \mathit{Sat}(\phi))]$, we first identify for which states this value is infinite. By definition of f, the this value is infinite if and only if for all strategies, the probability to reach a state in $\mathit{Sat}(\phi)$ is less than 1. These states can be identified by computing the set of states satisfying the PCTL formula $\mathsf{P}_{<1}[\mathsf{F}\,\phi]$, using the previously described methods.

For the remaining states, the value is finite and the computation of $\inf_{\sigma\in\Sigma}\mathbb{E}_s[f(r, Sat(\phi))]$ can be reduced to the computation of the minimum expected total reward in the following way. Firstly, we make all states in $Sat(\phi)$ terminal and construct a reward structure r' defined by $r'(s) = 0$ if $s \in Sat(\phi)$ and $r'(s) = r(s)$ otherwise. Secondly, we remove the zero-reward end-components from MDP, for which we can use methods from [52]. Now, in the new MDP $\mathcal{M}'$, we have that $\inf_{\sigma\in\Sigma}\mathbb{E}_{\mathcal{M}',s}[rew(r')] = \inf_{\sigma\in\Sigma}\mathbb{E}_{\mathcal{M},s}[f(r, Sat(\phi))]$ as we can apply algorithms for expected total reward from Section 3.2.3 to obtain the value.

To compute $\sup_{\sigma\in\Sigma}\mathbb{E}_s[f(r, Sat(\phi))]$, we first identify the states which do not satisfy PCTL formula $\mathsf{P}_{\geq 1}[\mathsf{F}\,\phi]$. Those states get a value of infinity, because there exists a strategy which does not reach a state in $Sat(\phi)$ with probability 1. For the remaining states we reduce the computation to computing the optimal expected total reward in the following way. We make all states in $Sat(\phi)$ terminal and construct a reward structure r' defined by $r'(s) = 0$ if $s \in Sat(\phi)$ and $r'(s) = r(s)$ otherwise. Then, we have that $\sup_{\sigma\in\Sigma}\mathbb{E}_{\mathcal{M}',s}[rew(r')] = \sup_{\sigma\in\Sigma}\mathbb{E}_{\mathcal{M},s}[f(r, Sat(\phi))]$, where $\mathcal{M}'$ is the modified MDP and we can again invoke algorithms for expected total reward from Section 3.2.3 to compute this value.

pATL. We now present Probabilistic Alternating-time Temporal Logic (pATL), which has been introduced as an extension of ATL in [43, 22], where the semantics have been provided for concurrent SMGs, which are multi-player games where at each state several players can take actions simultaneously. Turn-based SMGs that we study in this thesis are a special case of concurrent SMGs, where at each state only one player has more than one action available. Here we present the semantics and model checking for pATL for the turn-based SMGs. Throughout the section we assume a fixed SMG $\mathcal{G} = \langle \Pi, S, (S_{\bigcirc}, (S_i)_{i\in\Pi}), \Delta, AP, \chi\rangle$.
The syntax of the *Probabilistic Alternating-time Temporal Logic (pATL)* is given by the following grammar:

$$\begin{aligned}
\phi &::= \top \mid a \mid \neg\phi \mid \phi \wedge \phi \mid \langle\langle C\rangle\rangle \mathsf{P}_{\bowtie p}[\psi] \\
\psi &::= \mathsf{X}\,\phi \mid \phi\,\mathsf{U}\,\phi,
\end{aligned}$$

where $a \in AP$, $C \subseteq \Pi$, $\bowtie \in \{<, \leq, \geq, >\}$, $p \in \mathbb{Q} \cap [0, 1]$.

Similarly to PCTL, pATL is a branching-time temporal logic, where we distinguish between state formulae (ϕ) and path formulae (ψ). In addition, it adopts the coalition operator $\langle\langle C\rangle\rangle$ of ATL [4], combining it with the probabilistic operator $\mathsf{P}_{\bowtie p}$ and path formulae from PCTL [66, 14]. A typical usage of the coalition operator is $\langle\langle\{1, 3\}\rangle\rangle\mathsf{P}_{\geq 0.9}[\mathsf{X}\,\mathsf{good}]$, which means "players 1 and 3 have a strategy to ensure that the probability of reaching a good state in the next step is at least 0.9, regardless of the strategies of other players". The *satisfaction relation* $\models$ for pATL is defined inductively for each state s of $\mathcal{G}$, as follows:

$$\begin{array}{lcl}
s \models \top & & \text{always} \\
s \models a & \Leftrightarrow & a \in \chi(s) \\
s \models \neg\phi & \Leftrightarrow & s \not\models \phi \\
s \models \phi_1 \wedge \phi_2 & \Leftrightarrow & s \models \phi_1 \text{ and } s \models \phi_2 \\
s \models \langle\langle C \rangle\rangle \mathsf{P}_{\bowtie p}[\psi] & \Leftrightarrow & \exists (\sigma_i)_{i\in C} \text{ such that } \forall (\sigma_j)_{j\in \Pi\setminus C} \text{ we have } \mathrm{Pr}_s(\psi) \bowtie p,
\end{array}$$

and for any path λ in $\Omega_{\mathcal{G}}$:

$$\begin{array}{lcl}
\lambda \models \mathsf{X}\,\phi & \Leftrightarrow & \lambda_1 \models \phi \\
\lambda \models \phi_1 \,\mathsf{U}\, \phi_2 & \Leftrightarrow & \lambda_i \models \phi_2 \text{ for some } i \in \mathbb{N} \text{ and } \lambda_j \models \phi_1 \text{ for } 0 \leq j < i.
\end{array}$$

The model checking algorithm of pATL for all formulae, other than $\langle\langle C \rangle\rangle \mathsf{P}_{\bowtie p}[\psi]$, is the same as for PCTL. Model checking of the coalition operator reduces to the computation of optimal probabilities in a two-player stochastic game (referred to as *coalition game*), where the players in the coalition represent one player ($\Box$) and the remaining players represent the other ($\Diamond$), and similarly as for PCTL on MDPs we have

$$\langle\langle C \rangle\rangle \mathsf{P}_{\bowtie p}[\psi] \Leftrightarrow \sup_{\sigma_\Box \in \Sigma_\Box} \inf_{\sigma_\Diamond \in \Sigma_\Diamond} \mathrm{Pr}_s(\psi) \bowtie p \quad \text{if} \bowtie \in \{\geq, >\}$$

and

$$\langle\langle C \rangle\rangle \mathsf{P}_{\bowtie p}[\psi] \Leftrightarrow \inf_{\sigma_\Box \in \Sigma_\Box} \sup_{\sigma_\Diamond \in \Sigma_\Diamond} \mathrm{Pr}_s(\psi) \bowtie p \quad \text{if} \bowtie \in \{\leq, <\}.$$

Values for the operator $\mathsf{X}\,\phi$ can be computed as follows:

$$\sup_{\sigma_\Box \in \Sigma_\Box} \inf_{\sigma_\Diamond \in \Sigma_\Diamond} \mathrm{Pr}_s(\mathsf{X}\,\phi) = \begin{cases} 1 & \text{if } s \in S_\Box \text{ and } \Delta(s) \cap \mathit{Sat}(\phi) \neq \emptyset, \\ 1 & \text{if } s \in S_\Diamond \text{ and } \Delta(s) \cap \mathit{Sat}(\phi) = \Delta(s), \\ \sum_{t \in \Delta(s) \cap \mathit{Sat}(\phi)} \Delta(s,t) & \text{if } s \in S_\bigcirc, \\ 0 & \text{otherwise,} \end{cases}$$

and similarly

$$\inf_{\sigma_\Box \in \Sigma_\Box} \sup_{\sigma_\Diamond \in \Sigma_\Diamond} \mathrm{Pr}_s(\mathsf{X}\,\phi) = \begin{cases} 0 & \text{if } s \in S_\Box \text{ and } \Delta(s) \setminus \mathit{Sat}(\phi) \neq \emptyset, \\ 0 & \text{if } s \in S_\Diamond \text{ and } \Delta(s) \cap \mathit{Sat}(\phi) = \emptyset, \\ \sum_{t \in \Delta(s) \cap \mathit{Sat}(\phi)} \Delta(s,t) & \text{if } s \in S_\bigcirc, \\ 1 & \text{otherwise.} \end{cases}$$

And for the case of $\phi_1 \mathsf{U} \phi_2$, we apply the same reduction as for PCTL, by making states $s \notin Sat(\phi_1) \cup Sat(\phi_2)$ absorbing and then computing the optimal probability in a coalition game to reach a state in $Sat(\phi_2)$ as described in Section 3.2.4.

3.4 Strategy models

When discussing MDPs and SMGs in Section 3.1, we defined the strategies as functions mapping from finite histories of the states of the model to a next action to be taken by the controller (for MDPs) or player (for SMGs). Then, in the definitions of properties in Section 3.2 and logic specifications in Section 3.3, we used quantification over strategies, e.g., for stochastic two-player games we addressed the problem of whether there exists a strategy for player $\Box$ such that, for all strategies of player $\Diamond$, the expected total reward is at least x. Often we are not only interested in deciding the existence of the strategy, but also want to obtain the actual strategy to, for example, implement it as a program for an agent. In order to do this, we need to define a *strategy model* that specifies how the function is implemented.

In this section we discuss the representation of the strategies and compare two strategy models, *deterministic-update* strategy, which is the most commonly seen in the literature, and the *stochastic-update* strategy introduced in [20], which is the representation that we adopt in this thesis and which we have chosen for the implementation of the strategy construction engine in the PRISM-games model checker (presented in Chapter 6). We prove that, for two reachability objectives (which are a special case of the multi-objective properties that we discuss in Chapter 5), the stochastic-update strategy can be exponentially more succinct than the classical deterministic-update representation.

Following [56, 20], we define a strategy by a set of memory elements, memory update function, next move function, and the initial distribution on the memory elements. We provide the definition for SMGs, but it also applies to Markov decision processes, as MDPs can be viewed as SMGs with one non-stochastic player.

Definition 4 *A* strategy σ_i *of player* i *in a game* $\mathcal{G} = \langle \Pi, S, (S_\bigcirc, (S_i)_{i\in\Pi}), \Delta, AP, \chi \rangle$ *is a tuple* $\sigma_i = \langle \mathcal{M}, \sigma_i^u, \sigma_i^n, \alpha \rangle$*, where:*

- $\mathcal{M}$ *is a countable set of* memory elements*;*
- $\sigma_i^u \colon \mathcal{M} \times S \to \mathcal{D}(\mathcal{M})$ *is a* memory update function*;*
- $\sigma_i^n \colon S_i \times \mathcal{M} \to \mathcal{D}(S)$ *is a* next move function*; and*

- $\alpha\colon S \to \mathcal{D}(\mathcal{M})$ *defines an* initial distribution *on the memory elements for a given initial state.*

A strategy is *memoryless* if $|\mathcal{M}| = 1$, requires finite memory if $|\mathcal{M}| < \infty$ and infinite memory if $|\mathcal{M}| = \infty$. We also classify the strategies based on the use of randomisation. A strategy $\sigma_i = \langle \mathcal{M}, \sigma_i^u, \sigma_i^n, \alpha \rangle$ is *memoryless deterministic* if σ_i^u, σ_i^n, and α map to Dirac distributions; *deterministic-update* if σ_i^u and α map to Dirac distributions, while σ_i^n maps to an arbitrary distribution; and *stochastic-update* where σ_i^u, σ_i^n, and α can map to arbitrary distributions. Note that, from an implementation point of view, the controller using a *stochastic-update* or a *deterministic-update* strategy where σ_i^n uses randomisation has to be equipped with a random number generator to provide a correct realisation of the strategy.

As mentioned before, the most popular strategy model in the literature is deterministic-update, i.e., when the strategy memory is updated deterministically and randomisation is allowed to happen only in the next move function (that is, the player can randomise only if the game is in the state that it controls). We show that allowing randomisation on the memory update can have significant influence on the size of the memory required when considering strategies for stochastic games having multiple reachability objectives (i.e., ensuring that two or more reachability goals are satisfied with the same strategy). In Theorem 5, we prove that stochastic-update strategies can be exponentially more succinct than deterministic-update ones already for the case of two terminal state reachability objectives. The proof of the theorem below has been adapted from the proof provided in [39], where sufficient and necessary conditions to achieve a precise expectation for any linearly bounded function in stopping games are presented.

Theorem 5 *Stochastic-update strategies are at least exponentially more succinct than deterministic-update in stochastic games.*

We only explain the intuition here and the detailed proof can be found in Appendix A. To prove this result we use the game from Figure 3.7 starting in state s_1 and consider the objective, where we want player $\Box$ to reach the target set of states labelled with $\mathtt{t}$ with probability 0.5 and the target set of remaining terminal states with probability 0.5 as well, i.e., the winning strategy for player $\Box$ has to distribute the probability equally between the terminal states labelled with $\mathtt{t}$ and not by picking the appropriate probability distribution in its only state s_d, no matter what choices player $\Diamond$ makes in states s_1 through to s_n.

We can show that by picking values $x_1, \ldots, x_n$ appropriately we can ensure that for every of the 2^n set of choices of player $\Diamond$, player $\Box$ needs to play a different distribution in s_d, and hence if a deterministic update strategy is used, it requires $|\mathcal{M}| = 2^n$ in order

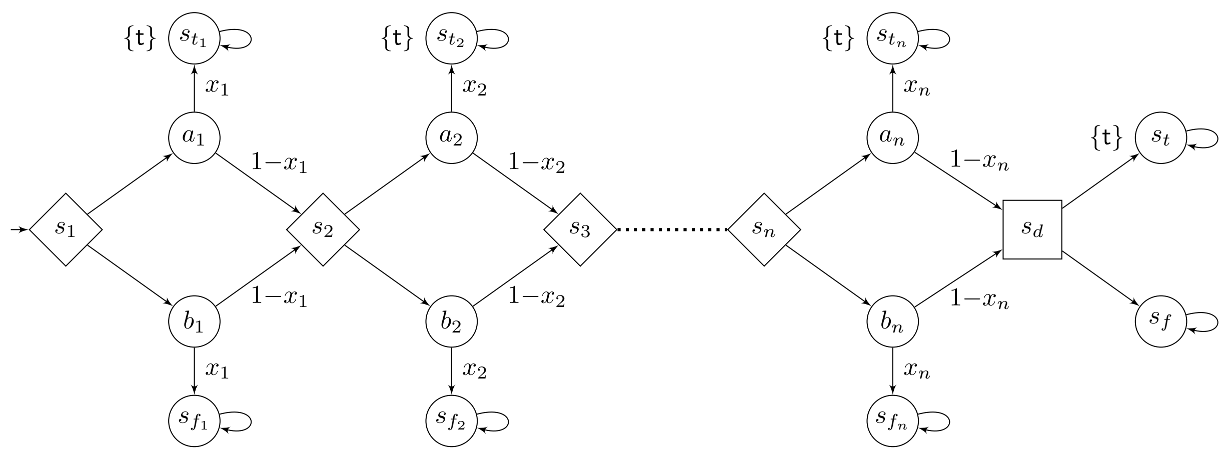


Figure 3.7: A game where player $\Box$ strategy is exponentially more succinct if expressed as stochastic-update rather than deterministic-update.

to achieve the objective. However, we can construct a stochastic-update strategy that has $2 \cdot |S|$ memory states as follows. Let $\text{val}^+(s) = \sup_{\sigma_\Box \in \Sigma_\Box} \inf_{\sigma_\Diamond \in \Sigma_\Diamond} \Pr_s(\mathsf{F}\,\mathsf{t})$ and $\text{val}^-(s) = \inf_{\sigma_\Box \in \Sigma_\Box} \sup_{\sigma_\Diamond \in \Sigma_\Diamond} \Pr_s(\mathsf{F}\,\mathsf{t})$ be the maximum and minimum reachability probabilities that player $\Box$ can achieve in a given state. The winning stochastic-update strategy $\sigma_\Box = \langle \mathcal{M}, \sigma^u_\Box, \sigma^n_\Box, \alpha \rangle$ that achieves the reachability probability exactly 0.5 from state s_1 is defined as follows:

- $\mathcal{M} = \{(s, \text{val}^-(s)), (s, \text{val}^+(s)) \mid s \in S\}$,
- $$\sigma^u_\Box((s,y),t) = \begin{cases} (t,y) & \text{if } s \in S_\Box, \\ [(t,\text{val}^-(t)) \mapsto \beta(y,t),\ (t,\text{val}^+(t)) \mapsto 1-\beta(y,t)] & \text{if } s \in S_\Diamond, \\ (t,\text{val}^-(t)) & \text{if } s \in S_\bigcirc \text{ and } y = \text{val}^-(s), \\ (t,\text{val}^+(t)) & \text{if } s \in S_\bigcirc \text{ and } y = \text{val}^+(s), \end{cases}$$
- $$\sigma^n_\Box(s_d,(s_d,y)) = \begin{cases} s_f & \text{if } y = \text{val}^-(s_d), \\ s_t & \text{otherwise}, \end{cases}$$
- $\alpha(s) = [(s,\text{val}^-(s)) \mapsto \beta(0.5,s),\ (s,\text{val}^+(s)) \mapsto 1-\beta(0.5,s)]$,

for all $s, t \in S$, and $(s,y) \in \mathcal{M}$, where $\beta(y,s) = c$ such that $0 \le c \le 1$ and $y = c \cdot \text{val}^-(s) + (1-c) \cdot \text{val}^+(s)$.

Intuitively, the strategy stores the optimal (minimum and maximum) values that can be achieved in each state, and stochastically updates its memory to ensure that the expectation of the memory element is the same as the intended value (in this case it is 0.5). We can then show that the expectation of the memory element is equal to the probability to reach a state labelled with $\mathtt{t}$, and hence probability to reach such a state in this game is exactly 0.5, and thus the probability to reach a terminal state not labelled with $\mathtt{t}$ is 0.5 as well.

Stochastic-update strategies allow player $\Box$ to react to player $\Diamond$ choices by randomising on the memory elements, and utilising the local relationship between values of the state and its successor. In the case of the game in Figure 3.7 the relationship is that given a player $\Diamond$ state s and two of its successors t, u we have that $\mathrm{val}^{-}(s), \mathrm{val}^{+}(s) \in [\mathrm{val}^{-}(t), \mathrm{val}^{+}(t)]$ and $\mathrm{val}^{-}(s), \mathrm{val}^{+}(s) \in [\mathrm{val}^{-}(u), \mathrm{val}^{+}(u)]$, hence player $\Box$ can always randomise to keep one of the four elements in strategy's memory no matter what is the choice of player $\Diamond$, thus avoiding the exponential explosion in memory size. We show later in the thesis, that this idea of using the local relationship between memory elements to preserve expectation can be extended to more dimensions and can be utilised to construct compact stochastic-update strategies for more general objectives.

3.5 Example

In this section we illustrate how the probabilistic models and properties presented in the previous sections can be applied to the analysis of competitive stochastic systems. We present a brief analysis of the team-formation protocol for sensor networks introduced in [62]. For the full case study, performed using PRISM model checker, we refer the reader to [41].

A short summary of the protocol is as follows. The system is composed of a network of agents (referred to as an agent organisation), each having a specific type of resource ($R = \{r_1, r_2, \ldots, r_k\}$). The tasks, which require a set of resources (e.g., task $T_j = \{r_i :$ "r_i is required by the task T_j"$\}$), are advertised to the network and agents are requested to form teams in order to complete the task. The task is completed successfully if the team has agents covering all resources from the task. The steps of the algorithm can be summarised as follows:

1. A set of tasks is generated according to a probability distribution and advertised to the agents.

2. In random order agents are allowed to act: if an agent has a missing resource for the task, then it can either initiate a new team for the task or join an existing team initiated by one of their neighbours. Only one team per task can be initiated.

3. After all agents have completed their actions, the teams, which have all the resources required by the task, receive a pay-off for successfully completing the task.

The pay-offs for the agents are defined as follows.

- Type $W_1(a)$, which rewards the agent a with 1 point if the agent is in the team that was able to complete its task after team formation is over, and 0 otherwise.
- Type $W_2(a)$, which rewards $\frac{1}{\text{team size}}$ to the agent a that was in a team that was able to complete its task, and 0 otherwise.

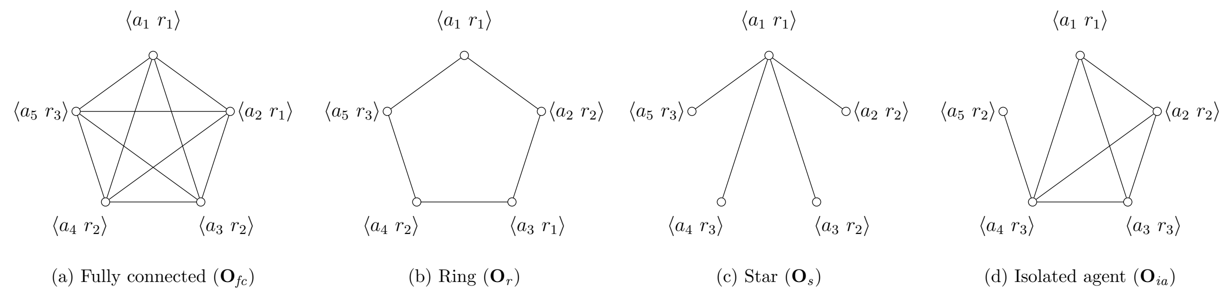
For a team of agents A, the rewards are defined accordingly as the total reward achieved by its members, i.e., $W_1(A) = \sum_{a \in A} W_1(a)$ and $W_2(A) = \sum_{a \in A} W_2(a)$. The underlying idea of these two types of rewards is that W_2 provides incentives for agents to form smaller teams, which can accomplish tasks, whereas W_1 motivates agents to simply be in a successful team. From the organisation's perspective, the W_2 reward should be used when resources are limited, whereas reward W_1 encourages agents to introduce resource redundancy into teams, but this may ensure that tasks are completed with higher probabilities.

In the scenario proposed originally by [62], agents do not make any decisions, but if they have a neighbour who is in a team for which the agent is eligible (i.e., has a missing resource), then it joins that team, and otherwise it initialises a team for the task with the probability given by the ratio of neighbours that are not committed to any team and the total number of neighbours. In this case, the behaviour of the system is entirely probabilistic and can be modelled as a DTMC. If, instead, we allow agents to make *decisions* as to whether join or initiate a team, we obtain an MDP. Finally, if we model every agent as a different player, we have an SMG. We also augment the models with the reward structures representing the two pay-off types, W_1 and W_2.

We now consider model checking of several reachability and expected total reward properties, and show how different models can be used to analyse the system.

Experimental setup. For our experiments we consider organisations consisting of five agents, which are organised into four networks, and each agent is assigned one of the three available resources (see Figure 3.8). We fix seven different tasks that are used in experiments $T = \{\{r_1\}, \{r_2\}, \{r_3\}, \{r_1, r_2\}, \{r_1, r_3\}, \{r_2, r_3\}, \{r_1, r_2, r_3\}\}$. When running the algorithm, two tasks T_1 and T_2 are picked uniformly and independently at random (with replacement) and are advertised to the agent organisation. For the details of the PRISM models of the protocol see Appendix F.1.

Properties. We consider three reachability objectives: probability to form a team to successfully complete task T_1, task T_2, and to complete both tasks (e.g., form two successful teams, one for each task). We also consider two expected total reward objectives defined by W_1 and W_2 reward functions as described earlier.



$\langle a_1\ r_1\rangle$
$\langle a_5\ r_3\rangle$
$\langle a_2\ r_1\rangle$
$\langle a_4\ r_2\rangle$
$\langle a_3\ r_2\rangle$
(a) Fully connected ($\mathbf{O}_{fc}$)
$\langle a_1\ r_1\rangle$
$\langle a_5\ r_3\rangle$
$\langle a_2\ r_2\rangle$
$\langle a_4\ r_2\rangle$
$\langle a_3\ r_1\rangle$
(b) Ring ($\mathbf{O}_r$)
$\langle a_1\ r_1\rangle$
$\langle a_5\ r_3\rangle$
$\langle a_2\ r_2\rangle$
$\langle a_4\ r_3\rangle$
$\langle a_3\ r_2\rangle$
(c) Star ($\mathbf{O}_s$)
$\langle a_1\ r_1\rangle$
$\langle a_5\ r_2\rangle$
$\langle a_2\ r_2\rangle$
$\langle a_4\ r_3\rangle$
$\langle a_3\ r_3\rangle$
(d) Isolated agent ($\mathbf{O}_{ia}$)

Figure 3.8: Experimental configurations of the agent organisations. In parentheses are the abbreviations that use to refer to the organisations throughout this section.

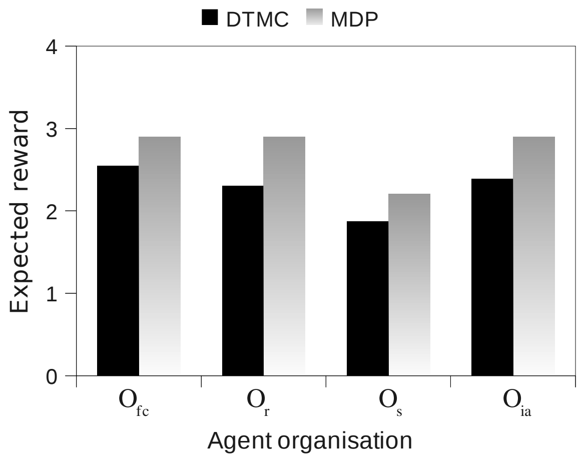
DTMC analysis. For DTMCs, we can compute the precise values for all five objectives to assess the performance of different agent organisations; this provides insight regarding how the algorithm performs in different network configurations.

The model checking results for the reachability probabilities starting from the initial state s_0, i.e., $\mathrm{Pr}_{s_0}(\mathsf{F}\,\mathsf{T1\_completed})$, $\mathrm{Pr}_{s_0}(\mathsf{F}\,\mathsf{T2\_completed})$ and $\mathrm{Pr}_{s_0}(\mathsf{F}\,(\mathsf{T1\_completed} \wedge \mathsf{T2\_completed}))$, are shown in Table 3.1. We can see that, unsurprisingly, in terms of task completion the algorithm performs best in the fully connected network and worst in the star network. However, it is interesting to note that the ring network outperforms the isolated agent network, even though the number of connections in the network is smaller.

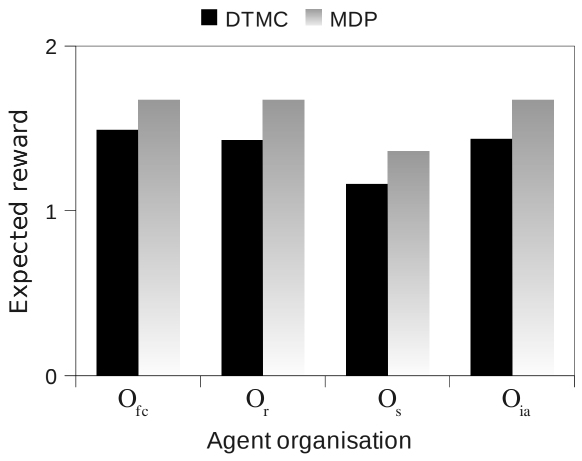
Organisation	T_1 completed	T_2 completed	T_1 and T_2 completed
O_{fc}	0.74562	0.74562	0.49596
O_r	0.71461	0.71461	0.47062
O_s	0.58324	0.58324	0.23639
O_{ia}	0.71799	0.71799	0.44839

Table 3.1: Task completion probabilities for the agent organisations.

MDP analysis. Next, we modify the algorithm so that the agents are allowed to make decisions as to whether to join or initiate the team, instead of picking the action according to some predefined probability. We can use MDP model checking to find the maximum values that can be achieved by agent organisations, i.e., if all agents collaborate, what is the maximum probability to complete task T_1, complete task T_2 or complete both tasks. Similarly, we can verify the maximum expected pay-off that the agents can achieve. Then we can compare these results with the DTMC model and assess how effective is the original algorithm. The comparison of the optimal expected pay-off to the one achieved by the original algorithm for both types of reward is shown in Figure 3.9. The properties verified for DTMCs were expected total reward, i.e., $\mathbb{E}_{s_0}[rew(W_j)]$, and for MDPs the optimal expected total reward: $\max_{\sigma \in \Sigma} \mathbb{E}_{s_0}[rew(W_j)]$ for $j \in \{1, 2\}$. We can see that the performance drop-off is very consistent among agent organisations and reward types, suggesting that the loss of performance is not closely related to the agent network or reward type used.



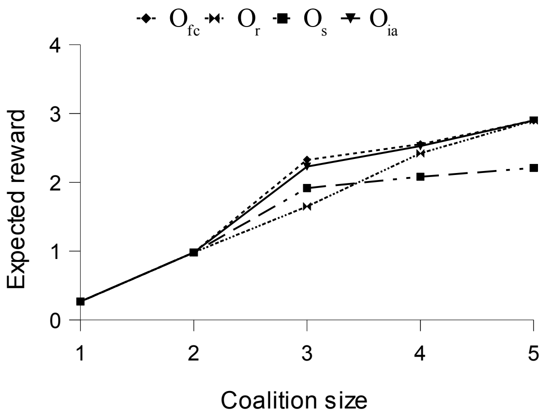
(a) Reward $W_1(O)$.



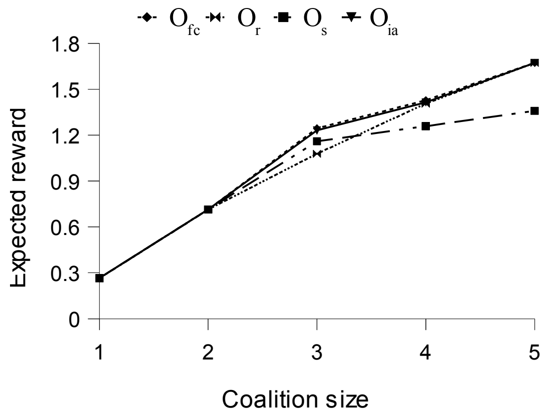
(b) Reward $W_2(O)$.

Figure 3.9: Performance comparison of original algorithm (modelled as a DTMC) with the optimal one (modelled as an MDP).

SMG analysis. Finally, we discuss how to enrich the analysis of the system using stochastic games. The previous discussion about optimal performance modelling using MDPs can be seen as agents forming a *grand coalition* (i.e., coalition consisting of all agents in the network) and trying to be as successful as possible. A natural question arises at this point: "how many agents do we require to be able to guarantee a certain performance of the agent organisation regardless of the actions of the other agents?" To model this problem, we can use stochastic two-player games: one player represents the agents in the coalition and the other represents the remaining players. Now, finding optimal values in this game would yield an answer to the question posed, because the maximum performance obtained for the first player would be in the worst-case scenario, i.e., the agents not in the coalition try to collaborate to minimise this objective. Figure 3.10 shows the comparison of the optimal coalition performance for different coalition sizes for both reward types. The expected total reward properties verified for the games are $\max_{(\sigma_i \in \Sigma_i)_{i \in C}} \min_{(\sigma_i \in \Sigma_i)_{i \in \Pi \setminus C}} \mathbb{E}_{s_0}[rew(W_j)]$ for $j \in \{1, 2\}$. The graphs reveal that, in our setting, the second type of pay-off (W_2) is more sensitive to the coalition size than the first type (W_1) for most agent organisations. This means that, in order to guarantee a certain level of performance for W_2, we would need to control more agents in the network, whereas the optimisation of W_1 can be done with a smaller number of agents.



(a) Reward $W_1(O)$.



(b) Reward $W_2(O)$.

Figure 3.10: Comparison of the optimal coalitions of varying sizes.

Summary. In this section we provided an example of how model checking of the properties presented in Section 3.2 for the probabilistic models from Section 3.1 can be used to conduct an analysis of competitive stochastic systems, in this case, a team formation protocol. We were able to evaluate the performance of the algorithm using DMTCs, compare it to the optimal performance using MDPs, and then, in order to address the competitive aspects of the system, we used stochastic games.

Note that the number of states in the probabilistic model representing the protocol is exponential in the number of agents in the network and thus only relatively small networks can be analysed. For example, the number of states and transitions in the SMG model of a five-agent fully connected agent organisation is 731,233 and 907,992, respectively; but the model with 6 agents already contains 8,155,873 states and 10,040,112 transitions. This is an example of the well-known state space explosion problem in formal verification. However, even the analysis of small models can help to understand the important properties of the algorithm, which would be hard to capture using other methods, such as manual analytic techniques or simulation-based methods. For example, we have been able to establish the optimal value achieved by the algorithm using MDP model checking, and analysed the performance under competitive scenarios using stochastic games.

3.6 Summary

In this chapter we reviewed the material from probabilistic model checking on which we build in this thesis. We presented the key probabilistic models: discrete-time Markov chains, Markov decision processes and (turn-based) stochastic multi-player games and associated concepts. For the aforementioned models, we also reviewed the algorithms for model checking reachability, parity and expected total reward properties. For DTMCs and MDPs properties can be verified in polynomial time. However, there is no known polynomial-time algorithm for model checking these properties for stochastic two-player games. We discussed multi-objective model checking for MDPs, which can be performed in polynomial-time for terminal state reachability and expected total reward objectives. We also presented two logics, PCTL and pATL that are used to express branching-time properties of probabilistic systems and provided their semantics in terms of MDPs (for PCTL) and SMGs (for pATL). We also discussed the models for representing controller strategies proving that, for certain property types, the model of choice for this thesis, stochastic-update strategy, is at least exponentially more succinct than the standard deterministic-update strategy model. Finally, to illustrate how the described models and properties can be applied to the analysis of the competitive stochastic systems, we presented the analysis of a team formation protocol.

Chapter 4

The Logic rPATL

In this chapter we describe a property specification language for competitive stochastic systems and define its semantics for stochastic multi-player games. More specifically, we introduce the Probabilistic Alternating-time Temporal Logic with Rewards (rPATL), which is a temporal logic that allows one to express quantitative objectives that can be achieved in a game by coalitions of players. The logic rPATL is an extension of the logic pATL [43] (see Section 3.3), which is itself a probabilistic extension of ATL [4], a widely used logic for reasoning about multi-player games and multi-agent systems. rPATL allows us to state that a *coalition* of players has a *strategy* which can ensure that either the *probability* of an event's occurrence or an *expected reward* measure meets some threshold, e.g., for a distributed network protocol one could specify: "nodes 1 and 2 can collaborate so that the probability of the protocol terminating within 45 seconds is at least 0.95, whatever nodes 3 and 4 do".

An important part of the logic is the inclusion of the expected *reward* measures. This enables quantitative reasoning about a system's use of resources, such as time spent or energy consumed. rPATL can state, e.g., "sensor 1 can ensure that the expected energy used, *if the algorithm terminates*, is less than $75mJ$, for any actions of sensors 2, 3, and 4". Similar reward-based operators have been introduced to the logic PCTL (see Section 3.3 for discussion), but have not yet been defined for games. In addition to providing model checking algorithms for the logic and discussing their complexity, we also show how to synthesise the strategies for players that achieve the objectives specified in rPATL, which, in the above example, would provide the strategy for sensor 1, which guarantees that "if the algorithm terminates the energy used is less than $75mJ$". This strategy can then be used to implement the controller for the sensor.

The chapter is structured as follows. We define the syntax and semantics of the logic in Section 4.1, provide model checking algorithms in Section 4.2, and discuss the model checking complexity in Section 4.3. Section 4.4 describes how to construct the strategies for the players that satisfy rPATL specifications using the results of the model checking algorithms. Then, before summarising the chapter in Section 4.7, we discuss several extensions of the logic, including rPATL* in Section 4.5, as well as reward-bounded properties in Section 4.6.

The technical results in this chapter are largely based on the results published in [36, 37]. We have adapted the semantics and algorithms of rPATL to be defined over a different (but equivalent) definition of SMG and modified the proofs accordingly, as well as adding Section 4.4 on strategy synthesis.

4.1 Syntax and semantics

We start by presenting the syntax and semantics of rPATL. Throughout the section, we assume a fixed game $\mathcal{G} = \langle \Pi, S, (S_{\bigcirc}, (S_i)_{i\in\Pi}), \Delta, AP, \chi \rangle$. In the presentation we use state reward structures $r : S \to \mathbb{Q}_{\geq 0}$, which are labelling functions mapping each state to a non-negative rational reward, but note that transition rewards can be easily encoded by adding an auxiliary state per transition to the model.

Definition 5 (rPATL) *The syntax of the* Probabilistic Alternating-time Temporal Logic with Rewards (rPATL) *is given by the following grammar:*

$$\begin{aligned} \phi &::= \top \mid a \mid \neg\phi \mid \phi \wedge \phi \mid \langle\langle C \rangle\rangle \mathsf{P}_{\bowtie q}[\psi] \mid \langle\langle C \rangle\rangle \mathsf{R}^{r}_{\bowtie x}[\mathsf{F}^{\star}\phi] \\ \psi &::= \mathsf{X}\,\phi \mid \phi\,\mathsf{U}^{\leq k}\,\phi \mid \phi\,\mathsf{U}\,\phi \end{aligned}$$

where $a \in AP$, $C \subseteq \Pi$, $\bowtie \in \{<, \leq, \geq, >\}$, $q \in \mathbb{Q} \cap [0,1]$, $x \in \mathbb{Q}_{\geq 0}$, $r : S \to \mathbb{Q}_{\geq 0}$ is a reward structure, $\star \in \{0, \infty, c\}$ and $k \in \mathbb{N}$.

rPATL is a branching-time temporal logic having two formula types: state (ϕ) and path (ψ). We adopt the coalition operator $\langle\langle C \rangle\rangle$ of ATL [4], combining it with the probabilistic operator $\mathsf{P}_{\bowtie q}$ and path formulae from PCTL [66, 14], and a generalisation of the expected reward operator $\mathsf{R}^{r}_{\bowtie x}$ from [60]. An example of a typical usage of the coalition operator is $\langle\langle \{agent1, agent2\} \rangle\rangle \mathsf{P}_{\geq 0.9}[\psi]$, which means "agents 1 and 2 have a strategy to ensure that the probability of path formula ψ being satisfied is at least 0.9, no matter what other players do". As path formulae, we allow the standard temporal operators X ("next"), $\mathsf{U}^{\leq k}$ ("bounded until") and U ("until"). As usual, we can directly derive $\bot \equiv \neg\top$, boolean connectives such as $\phi_1 \vee \phi_2 \equiv \neg(\neg\phi_1 \wedge \neg\phi_2)$ and common temporal operators such as $\mathsf{F}\,\phi \equiv \top\,\mathsf{U}\,\phi$ ("eventually"). We can handle the negated path formulae in a $\langle\langle C \rangle\rangle \mathsf{P}_{\bowtie q}$ operator by inverting the probability threshold, e.g., $\langle\langle C \rangle\rangle \mathsf{P}_{\geq q}[\neg\psi] \equiv \langle\langle C \rangle\rangle \mathsf{P}_{\leq 1-q}[\psi]$. This allows us to derive, for example, the G ("globally") and R ("release") operators, using the equivalences $\mathsf{G}\,\phi \equiv \neg(\mathsf{F}\,\neg\phi) \equiv \neg(\top\,\mathsf{U}\,\neg\phi)$ and $\phi_1 \mathsf{R}\,\phi_2 \equiv \neg(\neg\phi_1\,\mathsf{U}\,\neg\phi_2)$. This is done in the same way as for PCTL [66, 14], but, interestingly, cannot be done for ATL, as shown in [83].

Rewards

Before presenting the semantics of rPATL, we discuss the reward operators in the logic. We focus on the *expected total reward* to reach a target, i.e., the expected sum of rewards cumulated along a path until a state from a specified set $T \subseteq S$ is reached. To cope with the variety of different properties encountered in practice, we introduce three variants, which differ in the way they handle the case where T is *not* reached. The three types are denoted by the parameter $\star \in \{0, \infty, c\}$. These indicate that, when T is not reached along the path, the reward is zero, infinite or equal to the cumulated reward along the whole path, respectively.

The motivation for selecting these particular types of rewards stems from our experience of devising rPATL specifications for case studies (which we present in Chapter 6). Each reward type is applicable in different situations. If our goal is, for example, to minimise the expected time for the completion of the algorithm, then it is natural to assume a value of infinity ($\star$=∞) upon non-completion, i.e., we require the algorithm to complete successfully with probability 1 and, in addition, we want to minimise the expected running time. The semantics of this operator are essentially the same as for the $\mathsf{R}^r_{\bowtie q}[\mathsf{F}\,\phi]$ operator of PCTL (see Section 3.3).

Consider, on the other hand, the situation where we are only interested in, for example, minimising energy consumption throughout the lifetime of the device. In this case we might prefer to use type $\star = c$, to compute expected energy used regardless of termination. This operator can be viewed as the expected total reward (see Section 3.2.4) enhanced with a termination condition, i.e., the reward stops accumulating when a certain target set is reached.

The third type of reward, $\star = 0$, is useful in situations when the reward accumulated can only be retrieved when the target is reached, and otherwise it is discarded. For example, consider analysis of the incentive scheme where the user is rewarded £1 for every new user is refers to the company, but the amount is paid-out only if he/she refers at least 100 users, and no payment is received otherwise.

We formalise these notions of rewards by defining *reward functions* that map each possible path in the game $\mathcal{G}$ to a value indicating the total reward cumulated along that path.

Definition 6 (Reward function) *For an SMG $\mathcal{G} = \langle \Pi, S, (S_{\bigcirc}, (S_i)_{i\in\Pi}), \Delta, AP, \chi\rangle$, a reward structure $r : S \to \mathbb{Q}_{\geq 0}$, type $\star \in \{0, \infty, c\}$ and a set $T \subseteq S$ of target states, the* reward function *$rew(r, \star, T) : \Omega_{\mathcal{G}} \to \mathbb{R}_{\geq 0}$ is a random variable defined as follows. For $\lambda \in \Omega_{\mathcal{G}}$:*

$$rew(r,\star,T)(\lambda) \overset{\text{def}}{=} \begin{cases} g(\star) & \text{if } \forall j \in \mathbb{N} : \lambda_j \notin T, \\ \sum_{j=0}^{k-1} r(\lambda_j) & \text{otherwise, where } k = \min\{j \mid \lambda_j \in T\}, \end{cases}$$

and where $g(\star) = \star$ if $\star \in \{0, \infty\}$ and $g(\star) = \sum_{j\in\mathbb{N}} r(\lambda_j)$ if $\star = c$. The expected reward *from a state $s \in S$ of $\mathcal{G}$ under a strategy profile $\sigma_0, \ldots, \sigma_n$ is the expected value of the reward function, i.e., $\mathbb{E}_s^{\sigma_0,\ldots,\sigma_n}[rew(r, \star, T)]$.*

Semantics

Now, we define the semantics of rPATL. Formulae are interpreted over states of a game $\mathcal{G}$; we write $s \models \phi$ to indicate that state s of $\mathcal{G}$ satisfies the formula ϕ and define $Sat(\phi) \overset{\text{def}}{=} \{s \in S \,|\, s \models \phi\}$ as the set of states satisfying ϕ. Also, we use $\Pr_s(\psi)$ as a shorthand for $\Pr_s(\{\lambda \in \Omega_{\mathcal{G},s} \mid \lambda \models \psi\})$, i.e., to denote the probability measure of the paths satisfying formula ψ. For the $\langle\langle C\rangle\rangle \mathsf{P}_{\bowtie q}$ and $\langle\langle C\rangle\rangle \mathsf{R}^r_{\bowtie x}$ operators, we give the semantics via a reduction to a stochastic two-player game called a *coalition game.*

Definition 7 (Coalition game) *For an SMG $\mathcal{G} = \langle \Pi, S, (S_{\bigcirc}, (S_i)_{i\in\Pi}), \Delta, AP, \chi\rangle$ and a* coalition *of players $C \subseteq \Pi$, we define the* coalition game *of $\mathcal{G}$ induced by C as the stochastic two-player game $\mathcal{G}_C = \langle \{\Box, \Diamond\}, S, (S_{\bigcirc}, S_{\Box}, S_{\Diamond}), \Delta, AP, \chi\rangle$ where $S_{\Box} = \cup_{i\in C} S_i$ and $S_{\Diamond} = \cup_{i\in\Pi\setminus C} S_i$.*

Definition 8 (rPATL Semantics) *The* satisfaction relation *$\models$ for rPATL is defined inductively, for each state s of $\mathcal{G}$, as follows:*

$s \models \top$		*always*
$s \models a$	$\Leftrightarrow$	$a \in \chi(s)$
$s \models \neg\phi$	$\Leftrightarrow$	$s \not\models \phi$
$s \models \phi_1 \wedge \phi_2$	$\Leftrightarrow$	$s \models \phi_1$ *and* $s \models \phi_2$
$s \models \langle\langle C\rangle\rangle \mathsf{P}_{\bowtie q}[\psi]$	$\Leftrightarrow$	*In coalition game* $\mathcal{G}_C$, $\exists \sigma_{\Box} \in \Sigma_{\Box}$ *such that* $\forall \sigma_{\Diamond} \in \Sigma_{\Diamond}$, $\Pr_s(\psi) \bowtie q$
$s \models \langle\langle C\rangle\rangle \mathsf{R}^r_{\bowtie x}[\mathsf{F}^\star \phi]$	$\Leftrightarrow$	*In coalition game* $\mathcal{G}_C$, $\exists \sigma_{\Box} \in \Sigma_{\Box}$ *such that* $\forall \sigma_{\Diamond} \in \Sigma_{\Diamond}$ $\mathbb{E}_s[rew(r, \star, Sat(\phi))] \bowtie x$ *for* $\star \in \{0, \infty, c\}$

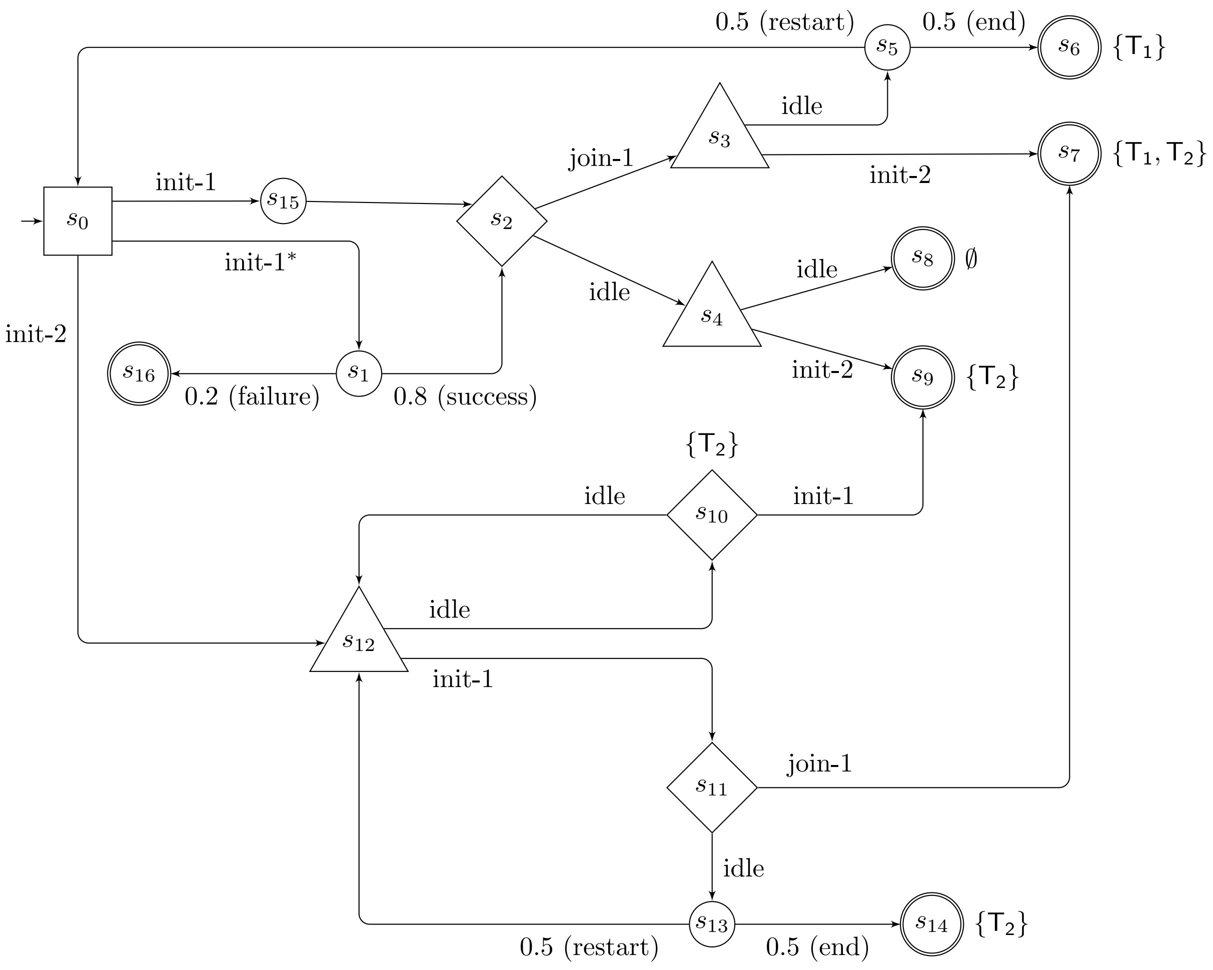


Figure 4.1: SMG modelling a variant of a team formation protocol with three players ($\Pi = \{\Box, \Diamond, \triangle\}$); players $\Box$ and $\triangle$ have resource 2 and player $\Diamond$ has resource 1; there are two tasks advertised: $\mathsf{T}_1 = \{1, 2\}$ and $\mathsf{T}_2 = \{2\}$. Action 'init-i' initialises a team for task i, action 'join-j' joins a team for task j and action 'idle' represents a decision to ignore the task. States with double border are terminal and contain a self-loop. Each state is also labelled with atomic proposition corresponding to its name, e.g., $\chi(s_4) = \{\mathsf{s_4}\}$.

and for any path λ in $\Omega_{\mathcal{G}}$:

$$\begin{array}{lcl}
\lambda \models \mathsf{X}\,\phi & \Leftrightarrow & \lambda_1 \models \phi \\
\lambda \models \phi_1 \,\mathsf{U}^{\leq k}\, \phi_2 & \Leftrightarrow & \lambda_i \models \phi_2 \textit{ for some } i \leq k \\
 & & \textit{and } \lambda_j \models \phi_1 \textit{ for } 0 \leq j < i \\
\lambda \models \phi_1 \,\mathsf{U}\, \phi_2 & \Leftrightarrow & \lambda \models \phi_1 \,\mathsf{U}^{\leq k}\, \phi_2 \textit{ for some } k \in \mathbb{N}.
\end{array}$$

Example 7 *In this example we show how the logic rPATL can be used to analyse an SMG. Consider the game from Figure 4.1 modelling a variant of a team formation protocol presented earlier in Section 3.5. The game has three players $\Box$, $\Diamond$ and $\triangle$ having resources*

2, 1 and 2, respectively. The game starts in s_0 and there are two tasks advertised: $T_1 = \{1, 2\}$ and $T_2 = \{2\}$. The actions that agents can take are 'init-i' (initialise a team for task i), 'join-i' (join a team for task i) and 'idle' (do nothing). The task is successfully completed if a state labelled with the respective atomic proposition is reached (which implies that the team consisting of agents having all the required resources by the task has been formed).

An example path through the game is $s_0 s_{12} s_{11} (s_7)^\omega$, which corresponds to player $\Box$ taking action 'init-2', then player $\triangle$ taking action 'init-1' and, finally, player $\Diamond$ selecting 'join-1', so that both tasks are successfully completed. The above strategies demonstrate that rPATL formula $\langle\langle\{\Box, \triangle, \Diamond\}\rangle\rangle \mathsf{P}_{\geq 1}[\mathsf{F}\,(\mathsf{T}_1 \wedge \mathsf{T}_2)]$ is satisfied in s_0.

Let us consider the rPATL formula $\langle\langle C\rangle\rangle \mathsf{P}_{\geq x}[\top\, \mathsf{U}^{\leq 5}\, \mathsf{T}_1]$, which states that the probability to complete task T_1 within 5 steps is at least x. First, note that, if $C = \{\Box\}$, then formula does not hold for any $x > 0$, because player $\Diamond$ can always pick action 'idle' in its states and hence the task is never be completed. If we have $C = \{\Box, \Diamond\}$, then formula is true for all $x \leq 0.5$ and false for $x > 0.5$, an optimal strategy for player $\triangle$ is to always pick action 'idle'. The optimal strategies for $\Box$ and $\Diamond$ are to choose 'init-1' and 'join-1' actions, respectively, making sure that a state satisfying T_1 is reached along the path $s_0 s_{15} s_2 s_3 s_5 (s_6)^\omega$ or $s_0 s_{15} s_2 s_3 (s_7)^\omega$, which, under deterministic strategies, have probability measures 0.5 and 1, respectively. Note that the unbounded version of the until for this coalition is valid for any x, i.e., formula $\langle\langle\{\Box, \Diamond\}\rangle\rangle \mathsf{P}_{\geq 1}[\top\, \mathsf{U}\, \mathsf{T}_1]$ is true in s_0, because no matter how many times $\triangle$ chooses to 'idle' in s_3, the task is completed with probability 1.

Consider the reward function $cost(s_{15}) = cost(s_{11}) = 50$, $cost(s_1) = 5$ and $cost(s) = 0$ for all other states s. The function represents the initiation cost of the task T_1 and it is 50 if action 'init-1' is executed by player $\Box$ or $\Diamond$ and it is 5 if action 'init-1' is taken (the cost of which is cheaper due to the fact that initiation can fail). The rPATL formula $\langle\langle\{\Box, \Diamond\}\rangle\rangle \mathsf{R}^{cost}_{\leq x}[\mathsf{F}^{\star}\mathsf{T}_1]$ expresses that players $\Box$ and $\Diamond$ have a strategy to complete task T_1 while using at most resource x. If $\star = \infty$, this formula is true for all $x \geq 100$ (achieved using the strategies described in the previous paragraph for satisfying $\langle\langle\{\Box, \Diamond\}\rangle\rangle \mathsf{P}_{\geq 1}[\mathsf{F}\,\mathsf{T}_1]$), but is false for any $x < 100$, because if $\Box$ uses action 'init-1*' to reduce the cost, then a state labelled with T_1 is reached with probability < 1 and thus the expected value is infinite. To circumvent this problem, we can change the reward type to $\star = c$, now the optimal strategy for $\Box$ is to use the action 'init-1*' and the formula is true for $x \geq \frac{25}{3}$.*

To illustrate the usage of the last type of reward, $\star = 0$, let us define the reward function $bonus(s_1) = bonus(s_{15}) = 10$ and $bonus(s_{12}) = 5$ representing the bonuses received by player $\Box$ for initiating the team for tasks one and two, respectively. We want to specify that this bonus is only paid-out if, in the end, at least one task is completed. This can be expressed as rPATL formula $\langle\langle\{\Box\}\rangle\rangle \mathsf{R}^{bonus}_{\geq x}[\mathsf{F}^{0}\,(\mathsf{T}_1 \vee \mathsf{T}_2)]$. The formula is true in s_0 for

$x \leq 5$, but is false for any $x > 5$. This is because if $\Box$ chooses 'init-2', then he receives reward 5 and also guarantees that task 2 is completed for any strategy of other players (and thus reward 5 is received), whereas if he chooses action 'init-1' (where he gets reward at least 10), then players $\Diamond$ and $\triangle$ can pick 'idle' actions and make sure that a state satisfying $\mathsf{T}_1 \vee \mathsf{T}_2$ is never reached (the game terminated in s_8) and thus the reward that was accumulated is never paid-out (i.e., the expectation is 0).

Quantitative queries

Often, instead of asking whether there exists a strategy for a coalition of players to achieve a certain value, we may want to ask a model checker to compute the maximum or minimum such value (also, the model checking algorithms that we present are based on the optimal value computation anyway). To address this issue we consider "quantitative" versions of the $\langle\langle C\rangle\rangle\mathsf{P}_{\bowtie q}$ and $\langle\langle C\rangle\rangle\mathsf{R}^r_{\bowtie x}$ operators, in the style of PRISM [77], which return such numerical values. For the probabilistic operator and a state $s \in S$, we have:

$$\begin{aligned}
\langle\langle C\rangle\rangle\mathsf{P}_{\min=?}[\psi] &\stackrel{\text{def}}{=} \mathrm{Pr}_s^{\min,\max}(\psi) \stackrel{\text{def}}{=} \inf_{\sigma_\Box \in \Sigma_\Box} \sup_{\sigma_\Diamond \in \Sigma_\Diamond} \mathrm{Pr}_s(\psi)\,,\\
\langle\langle C\rangle\rangle\mathsf{P}_{\max=?}[\psi] &\stackrel{\text{def}}{=} \mathrm{Pr}_s^{\max,\min}(\psi) \stackrel{\text{def}}{=} \sup_{\sigma_\Box \in \Sigma_\Box} \inf_{\sigma_\Diamond \in \Sigma_\Diamond} \mathrm{Pr}_s(\psi)
\end{aligned}$$

and for the reward operator:

$$\begin{aligned}
\langle\langle C\rangle\rangle\mathsf{R}^r_{\min=?}[\mathsf{F}^\star\phi] &\stackrel{\text{def}}{=} \mathbb{E}_s^{\min,\max}[\mathit{rew}(r,\star,\mathit{Sat}(\phi))] \stackrel{\text{def}}{=} \inf_{\sigma_\Box \in \Sigma_\Box} \sup_{\sigma_\Diamond \in \Sigma_\Diamond} \mathbb{E}_s[\mathit{rew}(r,\star,\mathit{Sat}(\phi))]\,,\\
\langle\langle C\rangle\rangle\mathsf{R}^r_{\max=?}[\mathsf{F}^\star\phi] &\stackrel{\text{def}}{=} \mathbb{E}_s^{\max,\min}[\mathit{rew}(r,\star,\mathit{Sat}(\phi))] \stackrel{\text{def}}{=} \sup_{\sigma_\Box \in \Sigma_\Box} \inf_{\sigma_\Diamond \in \Sigma_\Diamond} \mathbb{E}_s[\mathit{rew}(r,\star,\mathit{Sat}(\phi))]\,.
\end{aligned}$$

4.2 Model checking

In this section we discuss model checking for rPATL. We start by providing a generic procedure to verify rPATL formulae in Section 4.2.1. We then proceed by providing algorithms based on recursive equations satisfied by the $\langle\langle C\rangle\rangle\mathsf{P}_{\bowtie q}$ and $\langle\langle C\rangle\rangle\mathsf{R}^r_{\bowtie x}$ operators (in Sections 4.2.2 and 4.2.3, respectively), which allow us to derive efficient and practically usable algorithms for model checking rPATL (the pseudocode for the actual algorithms is presented in Chapter 6 when discussing tool implementation), in which computation of numerical values is done by evaluating fixpoints up to a desired level of convergence (in the style of well-known *value iteration* algorithms [48]). The proofs presented here are adapted from [37]; the modifications that we provide are to support a different definition of SMGs.

4.2.1 Model checking algorithm

The basic algorithm for model checking an rPATL formula ϕ on an SMG $\mathcal{G}$ proceeds as for other branching-time logics, such as CTL or PCTL, by determining the set $Sat(\phi)$ recursively, by traversing the parse-tree of the formula ϕ bottom-up. The nodes of the tree represent the sub-formulae of ϕ. At the leaves, we have either the atomic proposition or $\top$. Computation of satisfaction for these and for logical connectives is done by evaluating the resulting propositional logic formula. Thus, the focus of this section is on the evaluation of the $\langle\langle C\rangle\rangle \mathsf{P}_{\bowtie q}$ and $\langle\langle C\rangle\rangle \mathsf{R}^r_{\bowtie x}$ operators.

Computation of states satisfying these reduces to computation of *optimal* probabilities or expected rewards, respectively, in the coalition game $\mathcal{G}_C$. For example, if $\rhd \in \{\geq, >\}$, then:

$$\begin{array}{rcl} s \models \langle\langle C\rangle\rangle \mathsf{P}_{\rhd q}[\psi] & \Leftrightarrow & \mathrm{Pr}_s^{\max,\min}(\psi) \rhd q\,, \\ s \models \langle\langle C\rangle\rangle \mathsf{R}^r_{\rhd x}[\mathsf{F}^\star \phi] & \Leftrightarrow & \mathbb{E}_s^{\max,\min}[rew(r,\star,Sat(\phi))] \rhd x. \end{array}$$

For operators $\leq$ and $<$, we use the following equivalences to convert them into rPATL formula with $\geq, >$ operators, which follows from the determinacy result of [87] for zero-sum stochastic two-player games with Borel measurable payoffs (see Theorem 10 in Section 3.2.4):

$$\begin{array}{rcl} \langle\langle C\rangle\rangle \mathsf{P}_{\leq q}[\psi] & \equiv & \neg\langle\langle \Pi \setminus C\rangle\rangle \mathsf{P}_{>q}[\psi]\,, \\ \langle\langle C\rangle\rangle \mathsf{P}_{<q}[\psi] & \equiv & \neg\langle\langle \Pi \setminus C\rangle\rangle \mathsf{P}_{\geq q}[\psi]. \end{array} \tag{4.1}$$

The following sections describe how to compute the optimal probabilities ($\mathrm{Pr}_s^{\max,\min}(\psi)$) and expected rewards ($\mathbb{E}_s^{\max,\min}[rew(r,\star,Sat(\phi))]$) for the coalition game $\mathcal{G}_C$.

4.2.2 Computation of probabilities

We begin by showing how to compute the probabilities $\mathrm{Pr}_s^{\max,\min}(\psi)$ for states of a coalition game $\mathcal{G}_C = \langle\{\Box,\Diamond\}, S, (S_\bigcirc, S_\Box, S_\Diamond), \Delta, AP, \chi\rangle$. The following results follow in a similar way as the corresponding statements for PCTL model checking for Markov decision processes presented in Section 3.3.

For the X operator and state $s \in S$ the probability can be computed as follows:

$$\mathrm{Pr}_s^{\max,\min}(\mathsf{X}\,\phi) = \begin{cases} 1 & \text{if } s \in S_\Box \text{ and } \Delta(s) \cap Sat(\phi) \neq \emptyset, \\ 1 & \text{if } s \in S_\Diamond \text{ and } \Delta(s) \subseteq Sat(\phi), \\ \sum_{t \in \Delta(s) \cap Sat(\phi)} \Delta(s,t) & \text{if } s \in S_\bigcirc, \\ 0 & \text{otherwise.} \end{cases}$$

Probabilities for the $\mathsf{U}^{\leq k}$ operator in a state $s \in S$ can be computed recursively as follows:

$$\mathrm{Pr}_s^{\max,\min}(\phi_1 \,\mathsf{U}^{\leq k}\, \phi_2) = \begin{cases} 1 & \text{if } s \in \mathit{Sat}(\phi_2), \\ 0 & \text{if } s \notin (\mathit{Sat}(\phi_1) \cup \mathit{Sat}(\phi_2)), \\ 0 & \text{if } k{=}0 \text{ and } s \not\in \mathit{Sat}(\phi_2), \\ \max_{t \in \Delta(s)} \mathrm{Pr}_t^{\max,\min}(\phi_1 \,\mathsf{U}^{\leq k-1}\, \phi_2) & \text{if } s \in S_{\Box}, \\ \min_{t \in \Delta(s)} \mathrm{Pr}_t^{\max,\min}(\phi_1 \,\mathsf{U}^{\leq k-1}\, \phi_2) & \text{if } s \in S_{\Diamond}, \\ \sum_{t \in \Delta(s)} \Delta(s,t) \cdot \mathrm{Pr}_t^{\max,\min}(\phi_1 \,\mathsf{U}^{\leq k-1}\, \phi_2) & \text{if } s \in S_{\bigcirc}. \end{cases} \tag{4.2}$$

Finally, the unbounded until can be computed via value iteration [48], i.e., using:

$$\mathrm{Pr}_s^{\max,\min}(\phi_1 \,\mathsf{U}\, \phi_2) = \lim_{k \to \infty} \mathrm{Pr}_s^{\max,\min}(\phi_1 \,\mathsf{U}^{\leq k}\, \phi_2). \tag{4.3}$$

Also, we can apply the model checking algorithm for pATL and reduce the problem to reachability (see Section 3.3 for details).

4.2.3 Computation of rewards

Next, we show how to compute the optimal values $\mathbb{E}_s^{\max,\min}[\mathit{rew}(r, \star, \mathit{Sat}(\phi))]$ for all three types $\star \in \{0, \infty, c\}$. Our algorithms, like similar ones for computing expected total reward for MDPs (see Section 3.2.3) and games (see Section 3.2.4), rely on computing fixpoints of certain sets of equations. As in the previous section, we assume that this is done by value iteration with an appropriate convergence criterion (for discussion on convergence and experimental results see Section 6.5).

In this section, in addition to a coalition game $\mathcal{G}_C$, we fix a reward structure r and a target set $T = \mathit{Sat}(\phi)$, and make the following modifications to $\mathcal{G}_C$.

- Fresh atomic propositions t and $\mathsf{a}_{\mathrm{rew}}$ are added to target and positive reward states: $AP := AP \cup \{\mathsf{t}, \mathsf{a}_{\mathrm{rew}}\}$, $\forall t \in T : \chi(t) := \chi(t) \cup \{\mathsf{t}\}$ and $\forall s \in S \,.\, r(s) > 0 \Rightarrow \chi(s) := \chi(s) \cup \{\mathsf{a}_{\mathrm{rew}}\}$ (we assume that AP did not originally contain t and $\mathsf{a}_{\mathrm{rew}}$).
- Target states are made terminal: $\forall s \in T : \Delta(s) := \{s\}, r(s) = 0$.

It is easy to see that the above modification does not affect the value that we want to compute, i.e., $\mathbb{E}_s^{\max,\min}[\mathit{rew}(r, \star, \mathit{Sat}(\phi))]$, in the original game, since, after the target state is reached, the reward stops accumulating and adding new atomic propositions does not change the set $\mathit{Sat}(\phi)$.

Before presenting the model checking algorithms, we prove the following lemma about the optimal strategies required to win the games, which we use in several proofs of correctness of the algorithms later on.

Lemma 1 *Memoryless deterministic strategies are sufficient for achieving optimal expected reward $\mathbb{E}_s^{\max,\min}[rew(r,\star,T)]$ for types $\star = \{\infty, c\}$.*

Proof. If the expected value is infinite, then memoryless deterministic strategies suffice by Theorem 4 because this case reduces to the problem of reaching a state where the expected value is infinite with positive probability. States $s \in T$ get value 0 by the game modification presented above. Otherwise, the values $\mathbb{E}_s^{\max,\min}[rew(r,\star,T)]$ satisfy:

$$\mathbb{E}_s^{\max,\min}[rew(r,\star,T)] = \begin{cases} r(s) + \max_{t\in\Delta(s)} \mathbb{E}_t^{\max,\min}[rew(r,\star,T)] & \text{if } s \in S_\Box, \\ r(s) + \min_{t\in\Delta(s)} \mathbb{E}_t^{\max,\min}[rew(r,\star,T)] & \text{if } s \in S_\Diamond, \\ r(s) + \sum_{t\in\Delta(s)} \Delta(s,t)\cdot \mathbb{E}_t^{\max,\min}[rew(r,\star,T)] & \text{if } s \in S_\bigcirc. \end{cases} \quad (4.4)$$

Let $\mathrm{Succ}_{\mathrm{opt}}(s)$ be the set of successors of s that realise the optimum in $s \in S_\Box \cup S_\Diamond$, where opt is max or min, for players $\Box$ and $\Diamond$, respectively. We first analyse the case $\star = \infty$. Any strategy $\sigma_\Box^\infty \in \Sigma_\Box$ that in s picks the successor from $\mathrm{Succ}_{\mathrm{opt}}(s)$ is optimal. For player $\Diamond$, any strategy $\sigma_\Diamond^\infty \in \Sigma_\Diamond$ is optimal if it picks the successor from $\mathrm{Succ}_{\mathrm{opt}}(s)$ in s such that T is reached almost surely under any counter-strategy for player $\Box$ (because otherwise, the value is infinity).

Next, assume $\star{=}c$ and let $S_{=0} = \{s \mid \mathbb{E}_s^{\max,\min}[rew(r,c,T)] = 0\}$. To optimise $rew(r,c,T)$, we fix $\sigma_\Box^c \in \Sigma_\Box$ that uses successors from $\mathrm{Succ}_{\mathrm{opt}}(s)$ in s for all $s \in S_\Box$ and ensures that $S_{=0}$ is reached almost surely. For player $\Diamond$, any strategy $\sigma_\Diamond^c \in \Sigma_\Diamond$ is optimal if it picks a successor from $\mathrm{Succ}_{\mathrm{opt}}(s)$ in s.

We now prove the correctness of the above definitions. Given a state s and a strategy $\sigma_\Box$ for player $\Box$, we denote:

$$err^{\sigma_\Box}(s) = \frac{\min_{\sigma_\Diamond\in\Sigma_\Diamond} \mathbb{E}_s^{\sigma_\Box,\sigma_\Diamond}[rew(r,\star,T)]}{\mathbb{E}_s^{\max,\min}[rew(r,\star,T)]}$$

where we assume $err^{\sigma_\Box}(s) = 1$ if the denominator is 0. Observe that we have:

$$err^{\sigma_\Box}(s)\cdot \mathbb{E}_s^{\max,\min}[rew(r,\star,T)] = \min_{\sigma_\Diamond\in\Sigma_\Diamond} \mathbb{E}_s^{\sigma_\Box,\sigma_\Diamond}[rew(r,\star,T)].$$

Let $\star = c$. We prove that the maximiser's strategy $\sigma = \sigma_\Box^c$ defined above is optimal. Assume, for a contradiction, that it is not, i.e., $err^\sigma(s) < 1$ for some s. For all s, we have:

$$err^{\sigma}(s) \cdot \mathbb{E}_{s}^{\max,\min}[rew(r,c,T)] = r(s) + err^{\sigma}(\sigma(s)) \cdot \mathbb{E}_{\sigma(s)}^{\max,\min}[rew(r,c,T)] \tag{4.5}$$

and, for all $s \in S_{\Diamond}$, there must be a successor t such that:

$$err^{\sigma}(s) \cdot \mathbb{E}_{s}^{\max,\min}[rew(r,c,T)] = r(s) + err^{\sigma}(t) \cdot \mathbb{E}_{t}^{\max,\min}[rew(r,c,T)] \tag{4.6}$$

Fix s such that $err^{\sigma}(s) < 1$ is minimal. Thanks to equations (4.4), (4.5) and (4.6), we get that the value must also be minimal for all successors of s. However, this implies that $S_{=0}$ is not reached with probability 1 because, in every $t \in S_{=0}$, we have $err^{\sigma}(t) = 1$. The other cases ($\sigma_{\Diamond}^{c}$, $\sigma_{\Box}^{\infty}$ and $\sigma_{\Diamond}^{\infty}$) can be proved analogously. $\Box$

The case $\star = c$

First, we solve the following stochastic two-player Büchi game (see Section 3.2.4 for discussion) to identify the states from which the expected reward is infinite:

$$I := \{s \in S \mid \exists \sigma_{\Box} \in \Sigma_{\Box} \,.\, \forall \sigma_{\Diamond} \in \Sigma_{\Diamond} \ \mathrm{Pr}_s(\mathit{inf}(\mathsf{a}_{\mathrm{rew}})) > 0\}$$

where $\mathit{inf}(\mathsf{a}_{\mathrm{rew}})$ is the set of all paths that visit a state satisfying $\mathsf{a}_{\mathrm{rew}}$ infinitely often (and hence have infinite reward). The states in I are assigned infinite reward. To compute values for the remaining states, we remove the states in I from $\mathcal{G}_C$ before continuing. We then compute the *least* fixpoint of the following equations:

$$f(s) = \begin{cases} 0 & \text{if } s \in T, \\ r(s) + \max_{t \in \Delta(s)} f(t) & \text{if } s \in S_{\Box} \setminus T, \\ r(s) + \min_{t \in \Delta(s)} f(t) & \text{if } s \in S_{\Diamond} \setminus T, \\ r(s) + \sum_{t \in \Delta(s)} \Delta(s,t) \cdot f(t) & \text{if } s \in S_{\bigcirc} \setminus T, \end{cases} \tag{4.7}$$

and let $\mathbb{E}_{s}^{\max,\min}[rew(r,c,T)] = f(s)$.

Correctness. To show the correctness of the above algorithm, let us first consider the states with infinite value. Recall that we denote by $\mathit{inf}(\mathsf{a}_{\mathrm{rew}})$ the set of paths that visit a state with positive reward infinitely often (and thus get infinite reward). If, for a state s, there is $\sigma_{\Box} \in \Sigma_{\Box}$ such that the probability $\mathrm{Pr}_s(\mathit{inf}(\mathsf{a}_{\mathrm{rew}}))$ is positive for all $\sigma_{\Diamond} \in \Sigma_{\Diamond}$, then the strategy $\sigma_{\Box}$ itself achieves infinite reward. In the other direction, suppose that for every $\sigma_{\Box} \in \Sigma_{\Box}$ there is some $\sigma_{\Diamond} \in \Sigma_{\Diamond}$ such that $\mathrm{Pr}_s(\mathit{inf}(\mathsf{a}_{\mathrm{rew}}))$ is equal to zero. Because reward is not infinite, it follows that, for every $\sigma_{\Box}$, a strategy $\sigma_{\Diamond}$ exists, which also ensures that the expected number of visits to a state satisfying $\mathsf{a}_{\mathrm{rew}}$ is finite and bounded from above.

The rest follows easily because the rewards assigned to states by the reward function are bounded.

Let us now consider finite values. Because of the modification making target states terminal, no reward is accumulated after visiting a target state and we can change the random variable to use $\sum_{j\in\mathbb{N}} r(\lambda_j)$ instead of $\mathit{rew}(r,c,T)$. It can be shown by induction that the expected value with respect to this variable can be obtained as $\lim_{i\to\infty} f_s(i)$ where:

$$f_s(i) = \begin{cases} 0 & \text{if } i = 0, \\ r(s) + \max_{t\in\Delta(s)} f_t(i-1) & \text{if } i > 0 \text{ and } s \in S_{\Box}, \\ r(s) + \min_{t\in\Delta(s)} f_t(i-1) & \text{if } i > 0 \text{ and } s \in S_{\Diamond}, \\ r(s) + \sum_{t\in\Delta(s)} \Delta(s,t)\cdot f_t(i-1) & \text{if } i > 0 \text{ and } s \in S_{\bigcirc}. \end{cases} \tag{4.8}$$

We can then apply the Kleene fixpoint theorem to obtain that $\lim_{i\to\infty} f_s(i)$ is equal to the least fixpoint of the equations (4.7) (see Section 3.2.4 for discussion on computation of expected total reward using value iteration).

The case $\star = \infty$

We start by identifying and removing states with infinite expected reward; in this case it is $I := \{s \in S \mid s \models \langle\langle\{\Box\}\rangle\rangle \mathsf{P}_{<1}[\,\mathsf{F}\,\mathsf{t}\,]\}$. Then, for all other states $s \in S$, we compute the *greatest* fixpoint, *over* $\mathbb{R}$, of equations (4.7). The need for the greatest fixpoint arises because, in the presence of zero-reward subgames (i.e., the subgames in which all rewards are 0, and which do not contain a target state), multiple fixpoints may exist. For the previous case ($\star = c$), a least fixpoint of (4.7) gives the correct solution for such cases (i.e., because player $\Diamond$ should favour staying in such a cycle in all cases); here, the reward would be infinite if this happens (and hence player $\Diamond$ should prefer leaving such a cycle, but the least fixpoint of the equations (4.7) does not capture this property). The computation is over $\mathbb{R}$ since, e.g., the function mapping all non-target states to ∞ may also be a fixpoint, which is not the one we are interested in.

To find the greatest fixpoint over $\mathbb{R}$, we first compute an over-approximation of the fixpoint by changing all zero rewards to any $\varepsilon > 0$ and then evaluating the *least* fixpoint of (4.7) for the *modified* reward. Starting from the *new* initial values, value iteration now converges from above to the correct fixpoint. For the simpler case of Markov decision processes, an alternative approach based on removal of zero-reward end-components is possible [53], but this cannot be adapted efficiently to stochastic games. On the other hand, our over-approximation-based approach can be used for MDPs.

Correctness. We now prove the correctness of the presented algorithm. First, observe that, if a state $s \in S$ is assigned infinite value in the initial step, then we indeed have $\mathbb{E}_s^{\max,\min}[\mathit{rew}(r,\infty,T)] = \infty$ by definition. We prove the correctness for the other values. Let $u : S \to \mathbb{Q}$ be a function that assigns to each state $s \in S$ a value such that $u(s) \geq \mathbb{E}_s^{\max,\min}[\mathit{rew}(r,\infty,T)]$. Recall that we compute values of equations (4.7) by value iteration, i.e., we compute:

$$f_s(i) = \begin{cases} 0 & \text{if } s \in T, \\ u(s) & \text{if } i = 0, \\ r(s) + \max_{t \in \Delta(s)} f_t(i-1) & \text{if } i > 0 \text{ and } s \in S_\Box, \\ r(s) + \min_{t \in \Delta(s)} f_t(i-1) & \text{if } i > 0 \text{ and } s \in S_\Diamond, \\ r(s) + \sum_{t \in \Delta(s)} \Delta(s,t) \cdot f_t(i-1) & \text{if } i > 0 \text{ and } s \in S_\bigcirc, \end{cases} \tag{4.9}$$

for sufficiently large i, and we show that $\lim_{i\to\infty} f_s(i) = \mathbb{E}_s^{\max,\min}[\mathit{rew}(r,\infty,T)]$. Let us consider auxiliary functions rew_u^i which assign numbers to paths as follows:

$$\mathit{rew}_u^i(\lambda) = \begin{cases} \sum\limits_{j<k} r(\lambda_j) & \exists k \leq i \,.\, \lambda_k \in T, \\ \sum\limits_{j<i} r(\lambda_j) + u(\lambda_i) & \text{otherwise.} \end{cases}$$

Intuitively, the function rew_u^i alters the definition of $\mathit{rew}(r,\infty,T)$ by assigning rewards given by r for the first i steps, and then assigning the reward given by u, if the target has not been reached yet. One can prove by induction that the value of $f_s(i)$ from (4.9) is equal to $\mathbb{E}_s^{\max,\min}[\mathit{rew}_u^i]$.

We need to show that $\lim_{i\to\infty} f_s(i) \geq \mathbb{E}_s^{\max,\min}[\mathit{rew}(r,\infty,T)]$. This can be done inductively by showing that $f_s(i) \geq \mathbb{E}_s^{\max,\min}[\mathit{rew}(r,\infty,T)]$ for every i. The base case $i = 0$ follows from the definitions of f and u, and the inductive step follows by monotonicity of the function f.

Furthermore, we show that $\lim_{i\to\infty} f_s(i) \leq \mathbb{E}_s^{\max,\min}[\mathit{rew}(r,\infty,T)]$. Let $\sigma_{\min} \in \Sigma_\Diamond$ be a memoryless strategy satisfying $\sup_{\sigma\in\Sigma_\Box} \mathbb{E}_s^{\sigma,\sigma_{\min}}[\mathit{rew}(r,\infty,T)] = \mathbb{E}_s^{\max,\min}[\mathit{rew}(r,\infty,T)]$, i.e., $\sigma_{\min}$ is the optimal minimising strategy for player $\Diamond$ (such strategy exists by Lemma 1). Let $\tau(i) = \inf_{\sigma_\Box\in\Sigma_\Box} \Pr_s^{\sigma_\Box,\sigma_{\min}}(\{\lambda \in \Omega_{\mathcal{G}_C,s} \mid \exists j \leq i \,.\, \lambda_j \in T\})$ be the minimal probability with which we end-up in a state in T within i steps when playing according to $\sigma_{\min}$. We have $\lim_{i\to\infty} \tau(i) = 1$, because otherwise player $\Box$ would have a strategy to prevent the target from being reached almost surely and the reward obtained would be infinite. Thus, we have:

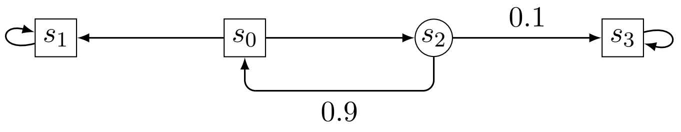
$$\sup_{\sigma_\Box\in\Sigma_\Box} \mathbb{E}_s^{\sigma_\Box,\sigma_{\min}}[\mathit{rew}_u^i] \leq \mathbb{E}_s^{\max,\min}[\mathit{rew}(r,\infty,T)] + (1-\tau(i)) \cdot K$$

where $K = \max_{s \in S} u(s)$. As we let i go to ∞, the second summand diminishes, and so $f_s(i) = \mathbb{E}_s^{\max,\min}[\mathit{rew}_u^i] \leq \mathbb{E}_s^{\max,\min}[\mathit{rew}(r,\infty,T)]$.

The case $\star = 0$

To achieve the optimal value for this reward type, it does not suffice to consider memoryless strategies for player $\Box$. The optimal strategy may depend on the reward accumulated so far, which, for a given history $\lambda \in \Omega_{\mathcal{G}}^{+}$, we denote by $\mathrm{r}(\lambda) \overset{\text{def}}{=} \sum_{i=0}^{|\lambda|} r(\lambda_i)$. However, this is only needed until a certain reward bound B is reached, after which the optimal strategy picks actions that maximise the probability of reaching T (if multiple such actions exist, it picks the one with the highest expected reward). Example 8 below illustrates this concept. The bound B can be computed efficiently using algorithms for $\star = c$ and $\mathrm{Pr}_s^{\max,\min}(\psi)$ and, in the worst-case, can be exponential in the size of $\mathcal{G}$ and the reward structure r (the formula to compute the bound is presented in Equation (4.11)).

Example 8 *In this example we show that memory is required for strategies that achieve optimal expected reward values of type $\star = 0$, i.e., for optimal values of $\mathbb{E}_s^{\max,\min}[\mathit{rew}(r,0,T)]$. Later, we show that the finite memory is indeed sufficient. Let us consider the following example:*



The target set is $T = \{s_1\}$ and the reward structure r assigns 1 to s_0 and 0 to the other states. We analyse the optimal value of $\mathit{rew}(r,0,T)$ in s_0. Let $\sigma_\Box$ be a memoryless randomised strategy that in s_0 picks s_1 as successor with probability x and s_2 with probability $1-x$. The expected reward obtained is then:

$$\sum_{i=1}^{\infty} i \cdot 0.9^{i-1} \cdot (1-x)^{i-1} \cdot x = \frac{x}{(0.1 + 0.9 \cdot x)^2},$$

which, for any x, is lower than $\frac{25}{9}$. Now consider the strategy σ' that is deterministic, and picks s_2 on the first 8 visits to s_0 and then s_1 on the 9th visit. The value under this strategy is: $9 \cdot 0.9^8 \approx 3.8 > \frac{25}{9}$. Hence, strategy with memory can achieve strictly greater expectation than any memoryless strategy.

Remark 1 *Note that, in the above example, optimal (memoryless deterministic) strategy in s_0 for both $\star = \infty$ and $\star = c$ is to pick successor s_2 and thus achieve values ∞ and 10, respectively.*

We now present the model checking algorithm. For clarity, we assume that rewards are integers; rational values can be handled by re-scaling all rewards by the lowest common multiple of the denominators of rewards appearing in the game. Let $R_{(s,k)}$ be the maximum expectation of $\mathit{rew}(r, 0, T)$ in state s after history λ with $\mathrm{r}(\lambda) = k$:

$$R_{(s,k)} \overset{\text{def}}{=} \sup_{\sigma_\Box \in \Sigma_\Box} \inf_{\sigma_\Diamond \in \Sigma_\Diamond} \Big[k \cdot \mathrm{Pr}_s(\mathsf{F}\,\mathsf{t}) + \mathbb{E}_s[\mathit{rew}(r, 0, T)] \Big],$$

and $r_{\max} = \max_{s \in S} r(s)$. The algorithm works as follows:

1. First we solve the Büchi game (see Section 3.2.4) to identify the states that have infinite reward:

$$I := \{ s \in S \mid \exists \sigma_\Box \in \Sigma_\Box \; \forall \sigma_\Diamond \in \Sigma_\Diamond \; \mathrm{Pr}_s(\mathit{inf}^{\mathsf{t}}(\mathsf{a}_{\mathrm{rew}})) > 0 \} \}$$

   where $\mathit{inf}^{\mathsf{t}}(\mathsf{a}_{\mathrm{rew}})$ is the set of all paths that visit a state satisfying $\langle\langle C \rangle\rangle \mathsf{P}_{>0}[\mathsf{F}\,\mathsf{t}] \wedge \mathsf{a}_{\mathrm{rew}}$ infinitely often. Then, assign infinite reward to states in I and remove them from the game.

2. For $B \leq k \leq B + r_{\max} - 1$ and for each state $s \in S$:

   (a) Assign new reward $r'(s) = r(s) \cdot \mathrm{Pr}^{\max,\min}_{\mathcal{G}_C,s}(\mathsf{F}\,\mathsf{t})$;

   (b) For states $s \in S_\Box$, remove from $\Delta(s)$ choices that are sub-optimal, i.e., t such that $\mathrm{Pr}^{\max,\min}_t(\mathsf{F}\,\mathsf{t}) < \mathrm{Pr}^{\max,\min}_s(\mathsf{F}\,\mathsf{t}) \Rightarrow t \notin \Delta(s)$;.

   (c) Compute $R_{(s,k)}$ using the algorithm for $\mathit{rew}(r', c, T)$:

$$R_{(s,k)} = k \cdot \mathrm{Pr}^{\max,\min}_s(\mathsf{F}\,\mathsf{t}) + \mathbb{E}^{\max,\min}_s[\mathit{rew}(r', c, T)].$$

3. Find, for all $0 \leq k < B$ and states s, the *least* fixpoint of the equations:

$$R_{(s,k)} = \begin{cases} k & \text{if } s \in T, \\ \max_{t \in \Delta(s)} R_{(t,k+r(s))} & \text{if } s \in S_\Box, \\ \min_{t \in \Delta(s)} R_{(t,k+r(s))} & \text{if } s \in S_\Diamond, \\ \sum_{t \in \Delta(s)} \Delta(s,t) \cdot R_{(t,k+r(s))} & \text{if } s \in S_\bigcirc. \end{cases} \tag{4.10}$$

4. The required value for the state $s \in S$ is $\mathbb{E}^{\max,\min}_s[\mathit{rew}(r, 0, T)] = R_{(s,0)}$.

Correctness. We prove the correctness of the algorithm by considering each of the steps in turn. To show the correctness of the first step, we prove the following lemma.

Lemma 2 $\sup_{\sigma_\Box \in \Sigma_\Box} \inf_{\sigma_\Diamond \in \Sigma_\Diamond} \mathbb{E}_s[rew(r, 0, T)] = \infty$ *if and only if there is* $\sigma_\Box \in \Sigma_\Box$ *such that for all* $\sigma_\Diamond \in \Sigma_\Diamond$ *we have* $\Pr_s(\mathit{inf}^{\mathsf{t}}(\mathsf{a}_{rew})) > 0$.

Proof. We begin with "$\Leftarrow$" direction. Player $\Box$'s strategy σ to ensure that the expected reward achieved is at least $q \in \mathbb{R}$ works as follows. Suppose $\sigma_\Box$ ensures $\Pr_s(\mathit{inf}^{\mathsf{t}}(\mathsf{a}_{\mathrm{rew}})) > p$ for all $\sigma_\Diamond$ strategies of player $\Diamond$. Since optimal values exist for Büchi games, we can safely assume that $p > 0$. The strategy σ mimics strategy $\sigma_\Box$ if the history λ satisfies $\mathrm{r}(\lambda) < \frac{q}{p \cdot x^{|S|}}$ where x is the minimal probability that occurs in the game. When $\mathrm{r}(\lambda)$ reaches this bound and the formula $\langle\langle C\rangle\rangle \mathsf{P}_{>0}[\mathsf{F}\,\mathsf{t}]$ is satisfied in the last state of λ, the strategy σ changes its behaviour and maximises the probability to reach T. Because memoryless deterministic strategies are sufficient for both players for reachability queries, σ can ensure that T is reached with probability at least $x^{|S|}$ from the last state of λ. The rest is a simple computation.

Let us analyse the direction "$\Rightarrow$". Similarly to the $\star = c$ case, we can show that, if for every $\sigma_\Box \in \Sigma_\Box$ there is $\sigma_\Diamond \in \Sigma_\Diamond$ such that $\Pr_s(\mathit{inf}^{\mathsf{t}}(\mathsf{a}_{\mathrm{rew}}))$ is equal to zero, then there is $\sigma_\Diamond$, which ensures that the expected number of visits to a state satisfying $\mathsf{a}_{\mathrm{rew}}$ is finite. The rest follows as in $\star = c$; we only need to further consider that if the state satisfies $\mathsf{a}_{\mathrm{rew}}$ but not $\mathsf{P}_{>0}[\mathsf{F}\,\mathsf{t}]$ (i.e., it gets non-zero reward but is not labelled with $\mathsf{a}_{\mathrm{rew}}$), then the reward achievable by player $\Box$ in such a state is 0. $\Box$

We continue by proving the correctness of the second step of the algorithm by showing that if player $\Box$ wants to maximise the expected reward with respect to $rew(r, 0, T)$ choosing only successors, which guarantee maximum probability to reach the target in each state, he can do so using a memoryless deterministic strategy. Then we show that this strategy can be played after a certain finite reward (bound B) has been accumulated.

Lemma 3 *Let* $\Sigma_\Box^T \subseteq \Sigma_\Box$ *contain all strategies that in all states* $s \in S_\Box$ *use only successors* $t \in S$ *such that* $\Pr_s^{\max,\min}(\mathsf{F}\,\mathsf{t}) = \Pr_t^{\max,\min}(\mathsf{F}\,\mathsf{t})$ *and for all* $\sigma_\Box \in \Sigma_\Box^T$ *and* $s \in S$ *we have* $\inf_{\sigma_\Diamond \in \Sigma_\Diamond} \Pr_s(\mathsf{F}\,\mathsf{t}) = \Pr_s^{\max,\min}(\mathsf{F}\,\mathsf{t})$. *Then there is a memoryless deterministic strategy* $\sigma_\Box^* \in \Sigma_\Box^T$ *satisfying:*

$$\inf_{\sigma_\Diamond \in \Sigma_\Diamond} \mathbb{E}_s^{\sigma_\Box^*, \sigma_\Diamond}[rew(r, 0, T)] = \sup_{\sigma_\Box \in \Sigma_\Box^T} \inf_{\sigma_\Diamond \in \Sigma_\Diamond} \mathbb{E}_s[rew(r, 0, T)] \,.$$

Proof. Assume the game is restricted so that the only successors available in $s \in S_\Box$ are the ones satisfying the lemma condition. We first create a new reward structure r' defined by $r'(s) = r(s) \cdot \Pr_s^{\max,\min}(\mathsf{F}\,\mathsf{t})$. We show that, for all $\sigma_\Box \in \Sigma_\Box^T$ and $\sigma_\Diamond \in \Sigma_\Diamond$ with $\Pr_s(\mathsf{F}\,\mathsf{t}) = \Pr_s^{\max,\min}(\mathsf{F}\,\mathsf{t})$, we have that $\mathbb{E}_s[rew(r', c, T)] = \mathbb{E}_s[rew(r, 0, T)]$, from which the lemma follows directly, as memoryless deterministic strategies suffice for achieving the optimal value of $rew(r', c, T)$.

Let $\Omega_{\mathcal{G}_C,s}(T) \overset{\text{def}}{=} \{\lambda \in \Omega_{\mathcal{G}_C,s} \mid \exists i \,.\, \lambda_i \in T\}$, and $t(\lambda) = \min_{i\in\mathbb{N}} \lambda_i \in T$. For any strategy profile $\sigma_\Box, \sigma_\Diamond$ such that $\mathrm{Pr}_s^{\sigma_\Box,\sigma_\Diamond}(\mathsf{F}\,\mathsf{t}) = \mathrm{Pr}_s^{\max,\min}(\mathsf{F}\,\mathsf{t})$,

$$\begin{aligned}
\mathbb{E}_s^{\sigma_\Box,\sigma_\Diamond}[\mathit{rew}(r,0,T)] &= \int_{\Omega_{\mathcal{G}_C,s}} \mathit{rew}(r,0,T)(\lambda) d\mathrm{Pr}_s = \int_{\Omega_{\mathcal{G}_C,s}(T)} \sum_{n=0}^{t(\lambda)} r(\lambda_n) d\mathrm{Pr}_s \\
&= \int_{\Omega_{\mathcal{G}_C,s}(T)} \sum_{n=0}^{\infty} r(\lambda_n) d\mathrm{Pr}_s = \sum_{n=0}^{\infty} \int_{\Omega_{\mathcal{G}_C,s}(T)} r(\lambda_n) d\mathrm{Pr}_s \\
&= \sum_{n=0}^{\infty} \sum_{t\in S} r(t) \cdot \mathrm{Pr}_s(\lambda_n = t \wedge \lambda \models \mathsf{F}\,\mathsf{t}) \\
&= \sum_{n=0}^{\infty} \sum_{t\in S} r(t) \cdot \mathrm{Pr}_s(\lambda_n = t) \cdot \mathrm{Pr}_s(\lambda \models \mathsf{F}\,\mathsf{t} \mid \lambda_n = t) \\
&= \sum_{n=0}^{\infty} \sum_{t\in S} r(t) \cdot \mathrm{Pr}_s(\lambda_n = t) \cdot \mathrm{Pr}_t^{\max,\min}(\mathsf{F}\,\mathsf{t}) \\
&= \sum_{n=0}^{\infty} \sum_{t\in S} r'(t) \cdot \mathrm{Pr}_s(\lambda_n = t) = \sum_{n=0}^{\infty} \int_{\Omega_{\mathcal{G}_C,s}} r'(\lambda_n) d\mathrm{Pr}_s \\
&= \int_{\Omega_{\mathcal{G}_C,s}} \mathit{rew}(r',c,T)(\lambda) d\mathrm{Pr}_s = \mathbb{E}_s^{\sigma_\Box,\sigma_\Diamond}[\mathit{rew}(r',c,T)].
\end{aligned}$$

This completes the proof. □

Finally, we prove the existence of the finite bound B on the reward accumulated on the path, after which it becomes optimal to play the strategy described in the previous lemma.

Given a path $h \in \Omega_{\mathcal{G}}^+$, we use $\mathbb{E}_s[\mathit{rew}(r,0,T) \mid h]$ to denote the *conditional* expectation of $\mathit{rew}(r,0,T)$ on infinite paths of which h is a prefix, i.e.:

$$\mathbb{E}_s[\mathit{rew}(r,0,T) \mid h] = \frac{\int_{\{\lambda \mid \lambda \text{ starts with } h\}} \mathrm{r}(\lambda)\, d\mathrm{Pr}_s}{\mathrm{Pr}_s(\{\lambda \mid \lambda \text{ starts with } h\})}.$$

Lemma 4 *For each state $s \in S$, there exists a finite-memory strategy σ^* for player $\Box$, which maximises the expected reward $\mathit{rew}(r,0,T)$ from the state s. In particular, there exists some bound B such that, for $r(h) \geq B$, $\sigma^*(h)$ becomes memoryless (i.e., as in Lemma 3).*

Proof. Fix two strategies $\sigma_\Box \in \Sigma_\Box$ and $\sigma_\Diamond \in \Sigma_\Diamond$. For each state $s \in S$ and a path $h = s_0 s_1 \ldots s_n$ we have that:

$$\begin{aligned}
\mathbb{E}_s^{\sigma_\Box,\sigma_\Diamond}[\mathit{rew}(r,0,T) \mid h] &= \mathbb{E}_s^{\sigma_\Box^h,\sigma_\Diamond^h}[\mathit{rew}(r,0,T) + r(h)] \\
&= \int_{\{\lambda \in \Omega_{\mathcal{G}_C,s} \mid \lambda \models \mathsf{F}\,\mathsf{t}\}} r(h) d\mathrm{Pr}_s^{\sigma_\Box,\sigma_\Diamond} + \mathbb{E}_{s_n}^{\sigma_\Box^h,\sigma_\Diamond^h}[\mathit{rew}(r,0,T)] \\
&= \mathrm{Pr}_{s_n}^{\sigma_\Box^h,\sigma_\Diamond^h}(\mathsf{F}\,\mathsf{t}) \cdot r(h) + \mathbb{E}_{s_n}^{\sigma_\Box^h,\sigma_\Diamond^h}[\mathit{rew}(r,0,T)]
\end{aligned}$$

where $\mathit{rew}(r,0,T) + r(h)$ is a random variable assigning $\mathit{rew}(r,0,T)(\lambda) + r(h)$ to a path λ reaching T, and 0 otherwise; and where $\sigma_i^h(h') = \sigma_i(s_0 s_1 \ldots s_{n-1} \cdot h')$ for $i \in \{\Box, \Diamond\}$.

Given a state $s \in S_\Box$, we use $PR_s^{\underline{\max}}$ to denote the "second" maximal probability to reach T achievable by a deterministic strategy. Below, without loss of generality we assume that such successor always exists (i.e., there is always a suboptimal choice with respect to reaching T). Define:

$$B_s = \frac{\mathbb{E}_s^{\max,\min}[\mathit{rew}(r,c,T)]}{\mathrm{Pr}_s^{\max,\min}(\mathsf{F}\,\mathsf{t}) - PR_s^{\underline{\max}}}\,. \tag{4.11}$$

Let $B = \max_{s \in S} B_s$. We show that, on paths h ending in s and satisfying $\mathrm{r}(h) > B$, no optimal strategy of player $\Box$ can use suboptimal choices and, together with Lemma 3, we obtain the statement of this lemma.

Let $h \cdot s$ be a path ending in $s \in S_\Box$ and satisfying $\mathrm{r}(h) > B$. Assume $\sigma_\Box(h)$ deterministically chooses successor $t \in \Delta(s)$ such that $\mathrm{Pr}_t^{\max,\min}(\mathsf{F}\,\mathsf{t}) < \mathrm{Pr}_s^{\max,\min}(\mathsf{F}\,\mathsf{t})$ (for randomised choices the argument follows analogously). We have that, for any $\sigma_\Diamond \in \Sigma_\Diamond$:

$$\begin{aligned}
& \mathbb{E}_s^{\sigma_\Box,\sigma_\Diamond}[\mathit{rew}(r,0,T) \mid h] \\
= \;& \mathrm{Pr}_s^{\sigma_\Box^h,\sigma_\Diamond^h}(\mathsf{F}\,\mathsf{t}) \cdot \mathrm{r}(h) + \mathbb{E}_s^{\sigma_\Box^h,\sigma_\Diamond^h}[\mathit{rew}(r,0,T)] \\
\le \;& PR_s^{\underline{\max}} \cdot \mathrm{r}(h) + \mathbb{E}_s^{\sigma_\Box^h,\sigma_\Diamond^h}[\mathit{rew}(r,0,T)] \\
= \;& \mathrm{Pr}_s^{\max,\min}(\mathsf{F}\,\mathsf{t}) \cdot \mathrm{r}(h) - (\mathrm{Pr}_s^{\max,\min}(\mathsf{F}\,\mathsf{t}) - PR_s^{\underline{\max}}) \cdot \mathrm{r}(h) + \mathbb{E}_s^{\sigma_\Box^h,\sigma_\Diamond^h}[\mathit{rew}(r,0,T)] \\
< \;& \mathrm{Pr}_s^{\max,\min}(\mathsf{F}\,\mathsf{t}) \cdot \mathrm{r}(h) - (\mathrm{Pr}_s^{\max,\min}(\mathsf{F}\,\mathsf{t}) - PR_s^{\underline{\max}}) \cdot B + \mathbb{E}_s^{\sigma_\Box^h,\sigma_\Diamond^h}[\mathit{rew}(r,0,T)] \\
\le \;& \mathrm{Pr}_s^{\max,\min}(\mathsf{F}\,\mathsf{t}) \cdot \mathrm{r}(h) - \mathbb{E}_s^{\max,\min}[\mathit{rew}(r,c,T)] + \mathbb{E}_s^{\sigma_\Box^h,\sigma_\Diamond^h}[\mathit{rew}(r,0,T)] \\
\le \;& \mathbb{E}_s^{\max,\min}[\mathit{rew}(r,0,T) \mid h]\,,
\end{aligned}$$

which contradicts that $\sigma_\Box$ is optimal. So, the strategy optimising $\mathit{rew}(r,0,T)$ is of finite-memory with upper bound B on the memory needed. $\Box$

By the equalities from the proof of Lemma 3 and by Lemma 4, the procedure described in step 2 of the algorithm is correct. The procedure from step 3 of the algorithm is correct because, for all paths h, we have that:

$$\mathbb{E}_s[rew(r,0,T) \mid h] = \begin{cases} \max_{t\in\Delta(s)} \mathbb{E}_s[rew(r,0,T) \mid h\cdot t] & \text{if } s \in S_\Box, \\ \min_{t\in\Delta(s)} \mathbb{E}_s[rew(r,0,T) \mid h\cdot t] & \text{if } s \in S_\Diamond, \\ \sum_{t\in\Delta(s)} \Delta(s,t)\cdot \mathbb{E}_s[rew(r,0,T) \mid h\cdot t] & \text{if } s \in S_\bigcirc. \end{cases}$$

Summary. In this section we have provided algorithms for rPATL model checking, which are based on the evaluation of the recursive equations. We will use these algorithms in the implementation of the PRISM-games model checker, presented in Chapter 6. We have also analysed memory requirements of the optimal strategies showing that memoryless strategies are sufficient for coalition players to achieve all but one of the rPATL operators. We will use these to provide complexity bounds for rPATL model checking in the next section.

4.3 Complexity

We are now ready to analyse the model checking complexity of rPATL.

Theorem 6 **(a)** *Model checking an rPATL formula with no* $\langle\langle C\rangle\rangle \mathsf{R}^r_{\bowtie x}[\mathsf{F}^0\phi]$ *operator and where* k *for the temporal operator* $\mathsf{U}^{\leq k}$ *is given in unary is in* NP∩coNP*.* **(b)** *Model checking an arbitrary rPATL formula is in* NEXP∩coNEXP*.*

Proof. **(a)** Let φ be a rPATL formula with no $\langle\langle C\rangle\rangle \mathsf{R}^r_{\bowtie x}[\mathsf{F}^0\phi]$ operator and where k for the temporal operator $\mathsf{U}^{\leq k}$ is given in unary. We prove that the problem of deciding whether the formula is satisfied in s is in NP∩coNP. By equivalences from Equations (4.1), we can assume that all probabilistic and reward operators only contain bounds $\geq$ or $>$, so in the proof we assume $\bowtie \in \{>, \geq\}$.

Let $\varphi_1, \varphi_2, \ldots, \varphi_n$ be the sequence of all state formulae occurring in φ. Also, if φ_i's outermost operator is temporal, let C_i denote the outermost coalition in φ_i, and Σ_j^C denote the set of all *memoryless deterministic* strategies for player j in the coalition game $\mathcal{G}_C$ (if the formula contains bounded-until, we expand the model by embedding the steps up to k into the state and transforming it into unbounded-until; this does not increase the size of the model since the bound k is given in unary).

We show that the problem is in NP∩coNP by describing a polynomial-size *certificate* c that allows us to check that a formula is (not) satisfied. The certificate c is a function that assigns an element of $\Sigma_\Box^{C_i} \cup \Sigma_\Diamond^{C_i}$ to each tuple (i, s) where $s \in S$ and φ_i is a formula whose outermost operator is temporal. We have the following.

- If $\varphi_i \equiv \langle\langle C \rangle\rangle \mathsf{P}_{\bowtie q}[\psi]$ and $s \models \varphi_i$, then:
  $c(i,s) = \sigma_\Box$ for $\sigma_\Box \in \Sigma_\Box^C$ such that $\inf_{\sigma_\Diamond \in \Sigma_\Diamond} \mathrm{Pr}_s(\psi) \bowtie q$ holds.

- If $\varphi_i \equiv \langle\langle C \rangle\rangle \mathsf{P}_{\bowtie q}[\psi]$ and $s \not\models \varphi_i$, then:
  $c(i,s) = \sigma_\Diamond$ for $\sigma_\Diamond \in \Sigma_\Diamond^C$ such that $\sup_{\sigma_\Box \in \Sigma_\Box} \mathrm{Pr}_s(\psi) \bowtie q$ does *not* hold.

- If $\varphi_i \equiv \langle\langle C \rangle\rangle \mathsf{R}^r_{\bowtie x}[\mathsf{F}^\star \phi]$ and $s \models \varphi_i$, then:
  $c(i,s) = \sigma_\Box$ for $\sigma_\Box \in \Sigma_\Box^C$ such that $\inf_{\sigma_\Diamond \in \Sigma_\Diamond} \mathbb{E}_s[\mathit{rew}(r, \star, \mathit{Sat}(\phi))] \bowtie x$ holds.

- If $\varphi_i \equiv \langle\langle C \rangle\rangle \mathsf{R}^r_{\bowtie x}[\mathsf{F}^\star \phi]$ and $s \not\models \varphi_i$, then:
  $c(i,s) = \sigma_\Diamond$ for $\sigma_\Diamond \in \Sigma_\Diamond^C$ such that $\sup_{\sigma_\Box \in \Sigma_\Box} \mathbb{E}_s[\mathit{rew}(r, \star, \mathit{Sat}(\phi))] \bowtie x$ does *not* hold.

The existence of the strategies assigned by c follows from Theorem 4 and from Lemma 1. To check the certificate in polynomial time, we compute $\mathit{Sat}(\varphi')$ for all state subformulae φ' of φ, traversing the parse tree of φ bottom-up. Suppose that we are analysing a formula φ' and that we have computed $\mathit{Sat}(\varphi'')$ for all state subformulae φ'' of φ'. If φ' is an atomic proposition or its outermost operator is a boolean connective, we construct $\mathit{Sat}(\varphi'')$ in the obvious way. Otherwise:

$$\mathit{Sat}(\varphi') = \{s \mid c(i,s) \text{ is a strategy for player } \Box \text{ in the coalition game}\}.$$

We verify that our choice of $\mathit{Sat}(\varphi')$ is correct as follows. For all $s \in \mathit{Sat}(\varphi')$, we construct an MDP from the appropriate coalition game by fixing the decisions of player $\Box$ according to $c(i,s)$, and in polynomial time we check that the minimal probability (or reward) in the resulting MDP exceeds the bound given by the outermost operator of φ' (see Theorem 4). If $s \notin \mathit{Sat}(\varphi')$, then we fix the decisions of the second player according to $c(i,s)$ and proceed analogously, computing the maximal probabilities.
(b) The proof is similar to the one above. We only need to extend the certificate from the proof to provide a witnessing strategy for formulae of the form $\langle\langle C \rangle\rangle \mathsf{R}^r_{\bowtie x}[\mathsf{F}^0 \phi]$. This is straightforward since, in Lemma 4, we showed that players need only strategies of exponential size.

In Lemma 4 we showed that, for the optimal strategy, it suffices to play a deterministic memoryless strategy after a certain reward bound B has been reached and, before that, the strategy needs to remember only the reward accumulated along the history. Without loss of generality we assume that rewards are integers, therefore, the strategy in a state may need a different action for each value of reward below B, and one action for reward, which is greater or equal to B. So, the overall size of the memory is $\mathcal{O}(|S| \times B)$. A deterministic strategy suffices in this case; observe that one could 'embed' the memory into the game by constructing a new game where the set of states is $S \times \{0, \ldots, B + r_{\max} - 1\} \cup \{s_f\}$ (where

$r_{\max}$ is the maximum reward assigned by a reward structure in a game). The transition relation is preserved for states (s, k) where $k < B$, and states (s, k) where $k \geq B$ have a transition to s_f only. The reward structure r assigns reward $R_{(s,k)}$ to states where $k \geq B$, which can be computed using step 2 of the algorithm for $\star = 0$, and 0 to all other states. Then, the deterministic memoryless strategy that maximises $rew(r, c, \{s_f\})$ in this new game also is an optimal strategy in the original game (but requiring memory of size B). The size of B can be at most exponential in the size of $\mathcal{G}_C$, i.e., from Lemma 4 it follows that the size of B for a state s is bounded by

$$B_s = \frac{\mathbb{E}_s^{\max,\min}[rew(r, c, T)]}{\Pr_s^{\max,\min}(\mathsf{F}\,\mathsf{t}) - PR_s^{\underline{\max}}}\,.$$

We claim that all $\mathbb{E}_s^{\max,\min}[rew(r, c, T)]$, $\Pr_s^{\max,\min}(\mathsf{F}\,\mathsf{t})$ and $PR_s^{\underline{\max}}$ can be represented as fractions of integers whose binary representation is polynomial in the size of the input, from which the bound on the size of B_s follows. For $\mathbb{E}_s^{\max,\min}[rew(r, c, T)]$ (or $\Pr_s^{\max,\min}(\mathsf{F}\,\mathsf{t})$, $PR_s^{\underline{\max}}$), fixing the optimal strategies for both players we can construct an LP, size of which is polynomial in the size of input and solution is equal to $\mathbb{E}_s^{\max,\min}[rew(r, c, T)]$ (or $\Pr_s^{\max,\min}(\mathsf{F}\,\mathsf{t})$, $PR_s^{\underline{\max}}$, respectively). Because the solution of the linear program can be represented as a fraction of two integers of polynomial binary representations, we get the claim. Therefore, B_s is at most exponential in the size of $\mathcal{G}$. $\square$

Remark 2 *Note that our problem is at least as hard as solving simple stochastic two-player games, which is known to be in* NP∩coNP *[48] and for which the existence of polynomial time algorithms is a long-standing open problem.*

4.4 Strategy synthesis

Previously in the chapter we presented the logic rPATL in order to express the properties of stochastic multi-player games. We also provided the algorithms for *model checking* the logic. In this section we take a different perspective on the rPATL specifications. Consider the rPATL formula $\langle\langle\{controller\}\rangle\rangle \mathsf{R}^{energy}_{\leq 100}[\mathsf{F}^c\,\mathsf{target}]$. If it evaluates to *true* in the given SMG then we have that "player *controller has a strategy* to ensure that the expected energy consumption before reaching target is less or equal to 100, for any strategies of the other players". But what we may be interested in is to actually *obtain the strategy* for the *controller*, which achieves the given objectives in an SMG, in which the formula is true. We refer to the problem of obtaining such a strategy for a coalition of players to satisfy a given rPATL formula as *strategy synthesis*. In this section we show how to construct the strategies for coalitions to satisfy the rPATL specifications for SMGs.

Before we proceed we note that the strategy synthesis only applies to the coalition-quantified rPATL formulae $\langle\langle C\rangle\rangle\mathsf{P}_{\bowtie x}[\psi]$ and $\langle\langle C\rangle\rangle\mathsf{R}^r_{\bowtie x}[\mathsf{F}^\star\phi]$; to construct the strategies we use the results of the model checking algorithms that were presented in Section 4.2. The constructions consider the maximisation problem, i.e., we assume $\bowtie\in\{\geq,>\}$. The minimisation strategies can be constructed in a similar way. To represent the strategies we use the representation defined in Section 3.4, where strategy σ is modelled as a tuple $\sigma=\langle\mathcal{M},\sigma^u,\sigma^n,\alpha\rangle$ where $\mathcal{M}$ is a set of memory elements, σ^u and σ^n are the memory update and next move functions, respectively, and α is the initial distribution on the memory elements.

Probabilities. We provide a winning strategy construction for coalition C in an SMG $\mathcal{G}=\langle\Pi,S,(S_{\bigcirc},(S_i)_{i\in\Pi}),\Delta,AP,\chi\rangle$ to satisfy an rPATL formula $\langle\langle C\rangle\rangle\mathsf{P}_{\bowtie x}[\psi]$. The winning strategies $\sigma_i=\langle\mathcal{M},\sigma_i^u,\sigma_i^n,\alpha\rangle$ for $i\in C$ are constructed as follows.

- Case $\psi=\mathsf{X}\,\phi$. $\mathcal{M}=\{m\}$, and for all $s\in S$ and $i\in C$ we have $\sigma_i^u(m,s)=m$, $\alpha(s)=m$; and for all states $s\in S_i$ we have $\sigma_i^n(s,m)=t$, where $t\in\Delta(s)\cap Sat(\phi)$ if $\Delta(s)\cap Sat(\phi)$ is not empty, and any state in $\Delta(s)$, otherwise.

- Case $\psi=\phi_1\,\mathsf{U}\,\phi_2$. Let X_s be the value $\mathrm{Pr}_s^{\max,\min}(\phi_1\,\mathsf{U}\,\phi_2)$ computed using Equation (4.3) from Section 4.2.2 representing the maximum probability to satisfy the formula for each state s. Memory of the strategy contains a single element, $\mathcal{M}=\{m\}$, and for all $s\in S$ and $i\in C$ we have $\sigma_i^u(m,s)=m$, $\alpha(s)=m$, and for $s\in S_i$ we have $\sigma_i^n(s,m)=[t_1\mapsto\frac{1}{n},\ldots,t_n\mapsto\frac{1}{n}]$, where if $T\overset{\text{def}}{=}\{t\in\Delta(s)|X_s\leq X_t\}\neq\emptyset$ then $\{t_1,\ldots,t_n\}=T$ and $\{t_1,\ldots,t_n\}=\Delta(s)$ otherwise. Intuitively, until the path satisfies the formula ψ, the strategy randomises uniformly among the optimal successor states, at least one of which is guaranteed to exist due to monotonicity [48] of the value iteration from Section 4.2.2 used to compute value X_s. Such a successor may not exist after the path has satisfied the formula ψ, and in this case, the strategy simply randomises uniformly among all successors. This can be done since in such case the probability to satisfy the formula is 1 regardless of any actions of the players.

- Case $\psi=\phi_1\,\mathsf{U}^{\leq k}\,\phi_2$. Let $X_s^0,\ldots,X_s^k$ be the values computed using Equations (4.2) for bounded until from Section 4.2.2 (i.e., $X_s^i=\mathrm{Pr}_s^{\max,\min}(\phi_1\,\mathsf{U}^{\leq i}\,\phi_2)$) representing the maximum probability to satisfy the formula in i steps for $0\leq i\leq k$ from each state s. Memory elements are $\mathcal{M}=\{0,\ldots,k\}$ and for all $s\in S$ and $i\in C$ we have $\sigma_i^u(m,s)=\max\{m-1,0\}$, $\alpha(s)=k$, and for $s\in S_i$ we have $\sigma_s^n(s,m)=t$ where if $m>0$ then $t\in\Delta(s)$ such that if $m>0$ then $X_s^m\leq X_t^{m-1}$, i.e., until the bound has been reached, the strategy chooses one of the optimal successors, at least one of which is guaranteed to exist due to monotonicity of the value iteration from

Equations (4.2) used to compute values $X_s^0, \dots, X_s^k$.

Rewards. Now we provide a winning strategy construction for coalition C in an SMG $\mathcal{G} = \langle \Pi, S, (S_{\bigcirc}, (S_i)_{i \in \Pi}), \Delta, AP, \chi \rangle$ to satisfy an rPATL formula $\langle\langle C \rangle\rangle \mathsf{R}^r_{\bowtie x}[\mathsf{F}^\star \phi]$ for $\star \in \{0, \infty, c\}$. The winning strategies $\sigma_i = \langle \mathcal{M}, \sigma_i^u, \sigma_i^n, \alpha \rangle$ for $i \in C$ are constructed as follows.

- Case $\star = c$. Let X_s be the values $f(s)$ computed using Equations (4.7) from Section 4.2.3 representing the maximum expected reward for each state s. Memory contains a single element $\mathcal{M} = \{m\}$ and for all $s \in S$ and $i \in C$ we have $\sigma_i^u(m, s) = m$, $\alpha(s) = m$, and for $s \in S_i$ we have $\sigma_i^n(s, m) = [t_1 \mapsto \frac{1}{n}, \dots, t_n \mapsto \frac{1}{n}]$ where $\{t_1, \dots, t_n\} = \{t \in \Delta(s) \,|\, X_s - r(s) \leq X_t\}$, i.e., the strategy randomises uniformly among the optimal successors.

- Case $\star = \infty$. Memory contains one memory element $\mathcal{M} = \{m\}$ and for all $s \in S$ and $i \in C$ we have $\alpha(s) = m$. Memory update function is $\sigma_i^u(m, s) = m$ for all $i \in C$, and if there exists a strategy from s to reach a target set with probability less than 1, and hence achieve infinite reward (this is performed by evaluating rPATL formula $\langle\langle C \rangle\rangle \mathsf{P}_{<1}[\mathsf{F}\,\phi]$, then for $s \in S_i$ we have $\sigma_i^n(s, m) = t$ where $X_s \geq X_t$, and X_s, X_t are values $\mathrm{Pr}_s^{\max,\min}(\mathsf{F}\,\phi), \mathrm{Pr}_t^{\max,\min}(\mathsf{F}\,\phi)$ computed using Equations (4.3) from Section 4.2.2. Otherwise, $\sigma_i^n(s, m) = \arg\max_{t \in \Delta(s)} X_t$, where X_t are the values $f(t)$ computed by Equations (4.7) from Section 4.2.3. Intuitively, the strategy is constructed from two strategies, one minimising the probability to reach a target set and another maximising the expected total reward.

- Case $\star = 0$. As in Section 4.2.3, for the ease of presentations we assume rewards assigned by r are integers. First, let us consider the strategies played in the states in which an arbitrarily large reward can be achieved, i.e., the states in the set I from step 1 of the algorithm from Section 4.2.3. Let σ^b be a memoryless deterministic strategy in the Büchi game, which makes sure that a state satisfying $\langle\langle C \rangle\rangle \mathsf{P}_{>0}[\mathsf{F}\,\mathsf{t}] \wedge \mathsf{a}_{\mathrm{rew}}$ is visited infinitely often with probability at least $p_b \geq p_{\min}^{|S|}$, and σ^r be a memoryless deterministic strategy that makes sure that the probability to reach a target set is at least $p_r \geq p_{\min}^{|S|}$ from any state in I, where $p_{\min}$ is the minimum non-zero probability assigned by Δ. Such strategies exist by definition of the set I. Let B be the bound and $R_{(s,k)}$ be the values computed in step 2 of the algorithm for $0 \leq k \leq B + r_{\max} - 1$ by Equations (4.10), and let X_s be the values $f(s)$ for the expected total reward computed using Equations (4.7) for the reward structure r'. Memory of all strategies is defined as $\mathcal{M} = \{0, \dots, \max(\lceil x / p_{\min}^{3 \cdot |S|} \rceil, B + r_{\max} - 1)\}$ and for all $s \in S$ and $i \in C$ we have $\sigma_i^u(m, s) = \min\{m + r(s), B + r_{\max} - 1\}$,

$\alpha(s) = 0$, and if $s \in I \cap S_i$ then $\sigma_i^n(s,m) = \sigma^b(s)$ if $m < \lceil x/p_{\min}^{3\cdot|S|} \rceil$ and otherwise $\sigma_i^n(s,m) = \sigma^r(s)$; else if $s \in S_i$ but $s \notin I$ then

$$\sigma_i^n(s,m) = \begin{cases} [t_1 \mapsto \frac{1}{n}, \ldots, t_n \mapsto \frac{1}{n}] & \text{if } m < B\,, \\ [r_1 \mapsto \frac{1}{m}, \ldots, r_m \mapsto \frac{1}{m}] & \text{otherwise} \end{cases}$$

where the successor states are $\{t_1, \ldots, t_n\} = \{t \in \Delta(s) \mid R_{(s,m)} \leq R_{(t,m+r(s))}\}$ and $\{r_1, \ldots, r_m\} = \{r \in \Delta(s) \mid X_s - r(s) \leq X_r\}$. Intuitively, strategy $\sigma_\Box$ for states $s \in I$ plays a Büchi strategy accumulating the required amount of reward before switching to a strategy which reaches the target set to 'cash-out' the reward and satisfy the property. This amount, after which it can make a switch, is bounded from above by $x/p_{\min}^{3\cdot|S|}$, because once the state $s \in I$ is reached and the accumulated reward at least $R = x/(p_b \cdot p_r)$, and because $R \leq x/p_{\min}^{2\cdot|S|}$, this guarantees that expectation is at least x; another factor of $p_{\min}^{|S|}$ is required to make sure that, if player $\Diamond$ plays sub-optimally from $t \notin I$ to take the game into $s \in I$, the expectation is big enough to be greater than x in the state in which the game started. For the states $s \notin I$, the strategy plays the strategy following the values of equations from step 3. of algorithm in Section 4.2.3 and then switches to following the expected total reward equations after the bound B has been exceeded.

Discussion. In this section we provided methods for strategy synthesis where the strategies can be constructed directly from the results of the (value iteration-based) model checking algorithms from Section 4.2. Strategy constructions provided here can be directly used to implement and analyse the controllers that satisfy the rPATL specifications. PRISM-games (see Chapter 6) implements these strategy constructions and, in Section 6.6.3, we use this functionality to analyse the protocol for user-centric networks.

The advantage of this approach is that strategy synthesis can be performed together with model checking. Furthermore, after the model checking has been performed, strategy construction procedure introduces minimal overhead (can be performed in linear time), except for the cases of bounded-until and reward operator when $\star = 0$, where the strategy synthesis requires the storage of intermediate computation results, which would not be required by otherwise, thus increasing memory requirements. This contrasts with other strategy construction methods (e.g., policy iteration [28]), which may take the number of steps exponential in the size of the model.

4.5 rPATL*

In this section, we discuss the logic rPATL*, which extends rPATL in the same way that PCTL* extends PCTL [14]. In particular, this allows LTL formulae to be provided as path formulae within the $\langle\langle C\rangle\rangle \mathsf{P}_{\bowtie q}[\psi]$ operator. The syntax of rPATL* is given by the following grammar:

$$\phi ::= \top \mid a \mid \neg\phi \mid \phi \wedge \phi \mid \langle\langle C\rangle\rangle \mathsf{P}_{\bowtie q}[\psi] \mid \langle\langle C\rangle\rangle \mathsf{R}^{r}_{\bowtie x}[\mathsf{F}^{\star}\phi]$$
$$\psi ::= \phi \mid \neg\psi \mid \psi \wedge \psi \mid \mathsf{X}\,\psi \mid \psi\,\mathsf{U}\,\psi$$

where $a \in AP$, $C \subseteq \Pi$, $\bowtie \in \{<, \leq, \geq, >\}$, $q \in \mathbb{Q} \cap [0, 1]$, $x \in \mathbb{Q}_{\geq 0}$, r is a reward structure and $\star \in \{0, \infty, c\}$.

Similarly to rPATL, we can derive standard temporal operators like $\mathsf{F}\,\psi \equiv \top\,\mathsf{U}\,\psi$ ("eventually") and $\mathsf{G}\,\psi \equiv \neg\mathsf{F}\,\neg\psi$ ("globally"). The semantics for state formulae are the same as for rPATL. The semantics for path formulae are as follows. For path $\lambda \in \Omega_{\mathcal{G}}$:

$$\begin{array}{lcl}
\lambda \models \phi & \Leftrightarrow & \lambda_0 \models \phi \\
\lambda \models \neg\psi & \Leftrightarrow & \lambda \not\models \psi \\
\lambda \models \psi_1 \wedge \psi_2 & \Leftrightarrow & \lambda \models \psi_1 \text{ and } \lambda \models \psi_2 \\
\lambda \models \mathsf{X}\,\psi & \Leftrightarrow & \text{Suffix}(\lambda, 1) \models \psi \\
\lambda \models \psi_1\,\mathsf{U}\,\psi_2 & \Leftrightarrow & \text{Suffix}(\lambda, i) \models \psi_2 \text{ for some } i \in \mathbb{N} \\
 & & \text{and Suffix}(\lambda, j) \models \psi_1 \text{ for } 0 \leq j < i\,.
\end{array}$$

Examples of rPATL* formulae include:

- $\langle\langle\{1,3\}\rangle\rangle \mathsf{P}_{\geq 1}[\mathsf{GF}\,\mathsf{recharge}]$ - "players 1 and 3 have a strategy to make sure that a recharge state is visited infinitely often with probability 1, no matter what actions other players take";

- $\langle\langle\{4\}\rangle\rangle \mathsf{P}_{>0.5}[(\mathsf{G}\,\mathsf{safe}) \wedge (\mathsf{FG}\,\mathsf{success})]$ - "player 4 has a strategy such that, with probability greater than 0.5, the system ends up and remains in success states, while only visiting safe states, for any strategies of the other players".

Example 9 *Consider again the team formation game presented in Figure 4.1. rPATL* allows us to express richer properties of paths, for example, formula* $\langle\langle\{\Box, \triangle, \Diamond\}\rangle\rangle \mathsf{P}_{\geq x}[(\mathsf{F}\,\mathsf{s}_5) \wedge (\mathsf{F}\,\mathsf{T}_2)]$ *expresses that players have a strategy to complete task* T_2 *along the paths that visit* s_5 *with probability at least* x*. This formula is only true for* $x \leq 0.5$*, and there are two optimal strategies for the coalition: player* $\Box$ *always picks action 'init-1', the first time* $\Diamond$ *has a choice, it has to pick 'join-1' and* $\triangle$ *has to pick 'idle' to make sure the state*

s_5 is visited. In the next round (when the game returns to s_2), $\Diamond$ can pick any action (corresponding to the two optimal strategies) and $\triangle$ has to select 'init-2' to make sure the task is satisfied. We can also specify liveness properties, for example, the rPATL formula $\langle\langle\{\Box,\triangle\}\rangle\rangle\mathsf{P}_{\geq 1}[(\mathsf{GF}\,\mathsf{s}_{12}) \vee (\mathsf{GF}\,\mathsf{s}_9)]$ is true in s_0, because no matter what strategy $\Diamond$ plays, $\Box$ and $\triangle$ can make sure that the game either ends in an infinite loop $(s_{12}s_{10})^\omega$ or in a terminal state s_9.*

Another important use of rPATL specifications is to impose restrictions on strategies that players in the coalition can use, e.g., rPATL* formula $\langle\langle\{\triangle,\Box\}\rangle\rangle\mathsf{P}_{\geq x}[(\mathsf{F}\,\mathsf{T}_1 \vee \mathsf{T}_2) \wedge \mathsf{G}\,\neg(\mathsf{s}_5 \vee \mathsf{s}_{10})]$ restricts actions that player $\triangle$ can take, i.e., he is not allowed to use action 'idle' in any of its states, when the coalition is trying to achieve one of the tasks. Similarly, the formula $\langle\langle\{\triangle,\Box,\Diamond\}\rangle\rangle\mathsf{P}_{\geq x}[(\mathsf{X}\,\mathsf{s}_1) \wedge (\mathsf{F}\,(\mathsf{T}_1 \vee \mathsf{T}_2))]$ forces $\Box$ to use the failure-prone initiation mechanism in the first initiation attempt. This formula is only valid for all $x \leq 0.8$, because if failure is experienced during the first initiation, the game terminates (this happens with probability 0.2), and otherwise players $\Diamond$ and $\triangle$ can ensure that at least one of the tasks is achieved with probability 1, by picking actions 'join-1' and 'init-2', respectively.*

Model checking. We now discuss model checking of the logic rPATL*. This can be done in a similar fashion to the logic PCTL* for Markov decision processes [14]. Model checking for an rPATL* formula ϕ can be performed as follows. Let $\phi_1, \phi_2, \cdots, \phi_n$ be a sequence of all (state) subformulae of ϕ, partially ordered by subsumption and where $\phi_n = \phi$. We compute $\mathit{Sat}(\phi_i)$ for each subformula ϕ_i in turn, starting from ϕ_1. If ϕ_i is an rPATL formula, then we apply the rPATL model checking algorithm described in Section 4.2.1. Otherwise, it must be of the form $\langle\langle C\rangle\rangle\mathsf{P}_{\bowtie q}[\psi]$, where ψ is an LTL formula.

For the latter case, we need to compute the optimal probabilities of satisfying LTL formula ψ for all states of the coalition game $\mathcal{G}_C$ and then compare these values with the bound q. Computing probabilities can be done in the following way. First, we translate ψ into a deterministic parity automaton with $\mathcal{O}(2^{2^{|\psi|}})$ states and $\mathcal{O}(2^{|\psi|})$ indices. Then, we build the product of the game $\mathcal{G}_C$ and the deterministic parity automaton, resulting in a stochastic two-player game with parity winning conditions. Such games can be solved using the methods described earlier in Section 3.2.4, where we enumerate all possible memoryless deterministic strategies. Thus, we can compute the optimal value in $\mathcal{O}((|\mathcal{G}_C|\cdot 2^{2^{|\psi|}})^{2^{|\psi|}}) = \mathcal{O}(|\mathcal{G}_C|^{2^{|\psi|}}\cdot 2^{2^{2|\psi|}})$ time, which entails that model checking ψ can be done in 2EXPTIME. Hence, model checking rPATL* is 2EXPTIME-complete (where the lower bound follows from the fact that model checking LTL formulae for Markov decision processes is 2EXPTIME-hard [51]).

4.6 Reward-bounded properties

So far, with rPATL one was able to reason about branching-time properties of paths of the game and several types of reward. For the reward operators, rPATL can express the existence of a coalition strategy to achieve a given bound on *expected* rewards. This does not, however, rule out the possibility of paths in the game that accumulate arbitrarily large (or even infinite) rewards, which may be undesirable when modelling resource consumption, because, in practice, the resources are bounded, e.g., a device may have finite battery capacity. Here, we take different view on the rewards by asking the question: "does there exist a strategy for the coalition to achieve a given rPATL property using *bounded resources*?". In particular, we extend the syntax of rPATL (see Definition 5) with the *reward-bounded until* temporal operator $\phi_1 \mathsf{U}^{r}_{\leq x}\phi_2$, where r is a reward structure, $x \in \mathbb{Q}_{\geq 0}$, and ϕ_1 and ϕ_2 are rPATL state formulae. Satisfaction for a path λ is defined as:

$$\lambda \models \phi_1 \mathsf{U}^{r}_{\leq x}\phi_2 \quad \Leftrightarrow \quad \text{there exists } k \geq 0 \text{ such that } \lambda_k \models \phi_2,\ \lambda_j \models \phi_1 \text{ for } 0 \leq j < k \text{ and } \textstyle\sum_{i=0}^{k} r(\lambda_i) \leq x.$$

For example, rPATL formula $\langle\langle C\rangle\rangle \mathsf{P}_{>0.9}[\top \mathsf{U}^{r}_{\leq 100}\, \mathsf{success}]$ means that coalition C has a strategy to guarantee that, with probability greater than 0.9, a state satisfying $\mathsf{success}$ is reached whilst consuming no more than 100 units of reward r.

Example 10 *Consider the team formation game from Figure 4.1, and the reward function* $cost(s_{15}) = cost(s_{11}) = 50$, $cost(s_1) = 5$ *and* $cost(s) = 0$ *for all other states* s. *The reward-bounded rPATL formula* $\langle\langle\{\Box, \Diamond\}\rangle\rangle \mathsf{P}_{\geq x}[\top \mathsf{U}^{cost}_{\leq y}\, \mathsf{T}_1]$ *expresses that players* $\Box$ *and* $\Diamond$ *have a strategy to make sure that task* T_1 *is completed with probability at least* x, *while incurring cost at most* y. *We also know from Example 7 that, for the standard until, the formula is valid for any* x, *i.e., the formula* $\langle\langle\{\Box, \Diamond\}\rangle\rangle \mathsf{P}_{\geq 1}[\top \mathsf{U}\, \mathsf{T}_1]$ *is true. Now, let us consider the situation when* $y = 50$, *i.e., players are allowed to incur the cost of at most* 50. *The formula is true for all* $x \leq 0.666597$, *which is achieved by player* $\Box$ *picking 'init-1*' 10 times in a row (and player* $\Diamond$ *picking 'join-1'). Note that picking 'init-1' action would mean that all resources are used-up in the first step and thus the probability to reach* T_1 *is* 0.5. *However, if we can allocate more resources to this goal, e.g., set* $y = 100$, *the optimal strategy becomes to pick 'init-1' in the first instance and then pick 'init-1*' thereafter. This strategy achieves the probability of* $0.5 + 0.5 \cdot 0.666597$ *to complete task* T_1 *before running out of resources.*

Model checking. Now, we consider the problem of model checking the *reward-bounded* rPATL formulae. In a similar fashion to the bounded until operator (see Section 4.2.2), probabilities for the $\phi_1 \mathsf{U}^{r}_{\leq x}\phi_2$ operator can be computed recursively. We have that

$$\mathrm{Pr}_s^{\max,\min}(\phi_1 \mathsf{U}^{r}_{\leq x}\phi_2) = \begin{cases} 1 & \text{if } s \in \mathit{Sat}(\phi_2), \\ 0 & \text{if } s \notin (\mathit{Sat}(\phi_1) \cup \mathit{Sat}(\phi_2)), \\ 0 & \text{if } x < r(s) \text{ and } s \in (\mathit{Sat}(\phi_1) \cup \mathit{Sat}(\phi_2)), \\ \max_{t\in\Delta(s)} \mathrm{Pr}_t^{\max,\min}(\phi_1 \mathsf{U}^{r}_{\leq x-r(s)}\phi_2) & \text{if } s \in S_{\Box}, \\ \min_{t\in\Delta(s)} \mathrm{Pr}_t^{\max,\min}(\phi_1 \mathsf{U}^{r}_{\leq x-r(s)}\phi_2) & \text{if } s \in S_{\Diamond}, \\ \sum_{t\in\Delta(s)} \Delta(s,t)\cdot \mathrm{Pr}_t^{\max,\min}(\phi_1 \mathsf{U}^{r}_{\leq x-r(s)}\phi_2) & \text{if } s \in S_{\bigcirc}. \end{cases}$$

Note that model checking a reward-bounded until operator is equivalent to model-checking an unbounded until operator on an extended model that encodes the accumulated reward (up to x) in its state. The state space of the extended model remains finite since rewards never need to exceed the bound r. This construction allows strategy synthesis to be performed using the algorithm for unbounded-until on the extended game as described in Section 4.4.

4.7 Summary

The contributions of this chapter can be summarised as follows.

Logic. We have introduced the temporal logic rPATL and its extensions to specify quantitative properties of stochastic multi-player games. The logic combines features of logics PCTL and ATL to allow reasoning about what the coalitions of players can achieve in the game. For example, rPATL formula $\langle\langle\{\mathit{sensor}, \mathit{robot}\}\rangle\rangle\, \mathsf{R}^{\mathit{cost}}_{\leq 95}\,[\,\mathsf{F}^{\infty}\,\mathsf{located}\,]$ states that "there exists a strategy for a coalition of *sensor* and *robot* which can guarantee that the target has been located with probability 1 and the expected cost incurred is at most 95".

Algorithms. The model checking is performed via a reduction to a two-player stochastic game, called a *coalition game*, where the states belonging to the coalition are assigned to one player and the remaining states represent the other. The algorithms that we have devised for model checking rPATL are based on the iterative fixpoint computation, also known as *value iteration*, which has proven to work well in practice for other probabilistic models, including Markov chains and Markov decision processes.

Complexity. We have showed that the problem of model checking rPATL lies in the complexity class NEXP∩coNEXP, and if one does not consider the reward operator for type $\star = 0$ (i.e., rPATL formulae free from subformulae of the form $\langle\langle C\rangle\rangle\, \mathsf{R}^{\mathit{cost}}_{\bowtie x}\,[\,\mathsf{F}^{0}\,\phi\,]$) the model checking problem is in NP∩coNP, due to determinacy and existence of memoryless deterministic optimal strategies for the coalition to win the game (whereas reward type $\star = 0$ requires strategy with memory which is at most exponential in the size of the game).

Synthesis. Often one is not only interested in finding out whether there exists a winning strategy for the coalition to win the game with an objective expressed as rPATL formula, but to actually obtain the strategies for the players in the coalition. We have provided algorithms for strategy construction, which can be implemented as controllers for the players of the game. For example, for the rPATL formula mentioned previously, $\langle\langle \{sensor, robot\} \rangle\rangle \, \mathsf{R}^{cost}_{\leq 95} \, [\, \mathsf{F}^{\infty} \, \mathsf{located} \,]$, our methods would produce two strategies, one for player *sensor* and another for player *robot*, which, if implemented in the game, would make sure that, whatever the other players choose to do, the game satisfies the property.

Discussion. The results presented in this chapter are important in several ways. We provided a specification language extending the logic pATL to support several types of expected total reward operators to specify quantitative properties of systems, while keeping the complexity of model checking the logic (with the exception of the $\mathsf{R}^{r}_{\bowtie x}[\mathsf{F}^{0}\,\phi]$ operator) the same as solving simple stochastic games. This was possible, because of existence of memoryless deterministic strategies to achieve optimal values (for the $\mathsf{R}^{r}_{\bowtie x}[\mathsf{F}^{0}\,\phi]$ operator, memory may be required to win). In addition to keeping the complexity the same, we also provided value iteration algorithms with small memory requirements for all rPATL operators and also for the extension with the reward-bounded until. The algorithms that we presented require storing one value per state of the game at any one time, so even for the operators, which require memory for optimal strategies can be efficiently verified.

Strategy synthesis algorithms that we provided here, use results of the value iteration algorithms directly to efficiently construct strategies so that the synthesis functionality could be added to the model checking introducing only small additional overhead (with the exception of bounded-until and $\mathsf{R}^{r}_{\bowtie x}[\mathsf{F}^{0}\phi]$, for which we require that the intermediate results of value iteration are stored, and hence memory requirements are increased).

Finally, note that MDPs are a special case of SMGs, and PCTL properties (as presented in Section 3.3) can be expressed in rPATL by using an empty set of players in the coalition operator, e.g., rPATL formula $\langle\langle \emptyset \rangle\rangle \mathsf{R}^{r}_{\bowtie x}[\mathsf{F}^{\infty}\phi]$ is equivalent to PCTL formula $\langle\langle C \rangle\rangle \mathsf{R}^{r}_{\bowtie x}[\mathsf{F}\,\phi]$, and so model checking algorithms for rPATL can be applied to PCTL model checking for MDPs as well. For example, the algorithm that we have presented for the $\mathsf{R}^{r}_{\bowtie x}[\mathsf{F}^{\infty}\,\phi]$ operator does not require the pre-processing that involves identification and elimination of zero-reward end-components before performing value iteration used by PCTL model checking algorithm presented in [75].

Chapter 5

Multi-Objective rPATL

In the previous chapter we presented the logic rPATL that is able to express properties such as $\langle\langle\{1,2\}\rangle\rangle\mathsf{P}_{\geq 0.5}[\mathsf{F}\,\mathsf{target}]$ or $\langle\langle\{1,2\}\rangle\rangle\mathsf{R}^{cost}_{\leq 10}[\mathsf{F}^{c}\,\mathsf{target}]$, which mean "players 1 and 2 have a strategy to ensure that the probability of reaching target is at least 0.5", and "players 1 and 2 have a strategy to ensure that the expected cost before reaching target is at most 10", respectively. Such properties are single-objective, because they contain one operator of the form $\mathsf{P}_{\bowtie x}[\psi]$ and $\mathsf{R}^{r}_{\bowtie x}[\mathsf{F}^{\star}\phi]$. In practice, however, one is often interested in specifying properties of systems involving several objectives. For example, one may be interested in finding out whether "there exists a strategy for controller, which achieves probability to reach the target state of at least 0.99 while *simultaneously* having expected energy consumption at most 100 units". rPATL is not able to express such properties. [1] Also, each agent in a coalition may have its own objective and one would like to specify that "coalition of agents A and B have a joint strategy which ensures that probability to reach a goal of A is at least 0.5 and the probability to reach a goal of B is at least 0.9".

In this chapter we present multi-objective rPATL, a generalisation of the logic rPATL, which allows us to specify properties containing multiple objectives, for example, the property $\langle\langle\{controller\}\rangle\rangle(\mathsf{P}_{\geq 0.99}[\mathsf{F}\,\mathsf{target}] \wedge \mathsf{R}^{energy}_{\leq 100}[\mathsf{F}^{c}\,\mathsf{target}])$ expresses the aforementioned goal that "player *controller* has a strategy to ensure that a target state is reached with probability 0.99 *and* the expected energy consumption is less than or equal to 100, for any strategies of other players." Note that this formula is not equivalent to the rPATL formula $\langle\langle C\rangle\rangle\mathsf{P}_{\geq 0.99}[\mathsf{F}\,\phi] \wedge \langle\langle C\rangle\rangle\mathsf{R}^{energy}_{\leq 100}[\mathsf{F}^{c}\phi]$, which expresses that coalition C has a strategy to achieve the first objective and also has the strategy to achieve the second objective, but does not imply that there exists a strategy which achieves both of them at the same time. Similarly, we can express the second example with the multi-objective rPATL formula $\langle\langle\{A,B\}\rangle\rangle(\mathsf{P}_{\geq 0.5}[\mathsf{F}\,\mathsf{G_A}] \wedge \mathsf{P}_{\geq 0.9}[\mathsf{F}\,\mathsf{G_B}])$, where $\mathsf{G_A}$ and $\mathsf{G_B}$ represent the goals of agents A and

[1] With the exception of the operator $\mathsf{R}^{r}_{\leq x}[\mathsf{F}^{\infty}\phi]$, the semantics of which implies that the goal is satisfied if and only if the probability to reach ϕ is 1 and expected reward is $\leq x$.

B, respectively.

Multi-objective rPATL permits formulae to contain a boolean combination of conjunctions and disjunctions of probability and expected total reward objectives. We show in this chapter that multi-objective rPATL model checking problem is considerably more difficult than rPATL: the games are not determined already for a conjunction or disjunction of two objectives; optimal strategies for the coalition, if such exist, may require randomisation and infinite memory to satisfy a multi-objective rPATL formula; the model checking problem is PSPACE-hard already for a conjunction of terminal state reachability goals, and is undecidable if the coalition is restricted to deterministic strategies. [2] Nevertheless, for a class of stopping games we provide model checking algorithms that can approximate the answer up to a specified precision, i.e., consider again the multi-objective rPATL formula $\langle\!\langle \{controller\} \rangle\!\rangle(\mathsf{P}_{\geq 0.99}[\mathsf{F}\,\mathsf{target}] \wedge \mathsf{R}^{energy}_{\leq 100}[\mathsf{F}\,\mathsf{target}])$: given precision $\varepsilon > 0$, if the algorithm returns *true* then there exists a strategy for *controller* to achieve the probability to reach target at least $0.99-\varepsilon$ and the expected energy consumption at most $100+\varepsilon$; and if the algorithm returns *false*, then there does not exist a strategy for the *controller* achieving probability $0.99+\varepsilon$ and energy consumption of at most $100-\varepsilon$. As for rPATL, the algorithms that we provide are based on value iteration.

The chapter is structured as follows. We start by defining the syntax and semantics of multi-objective rPATL in Section 5.1, then we present model checking algorithms in Section 5.2. Section 5.3 is dedicated to the complexity analysis, where we discuss the nondeterminacy of the games, strategy memory requirements, and several complexity bounds for the multi-objective rPATL model checking. In Section 5.4 we show how to, using the results of approximation algorithms, construct the winning strategies for the coalition. Finally, we consider multi-objective variant of the logic rPATL* in Section 5.5, before summarising the chapter in Section 5.6.

The technical parts of this chapter are largely based on the results published in [39], [42] and [40]. The results for computing the expected total reward, presented in Section 5.2, are based on [40]; determinacy and non-existence of optimal strategy results of Section 5.3 have been presented in [39] and memory requirements and complexity analysis is based on [40]; the results of Sections 5.5 and 5.4 for strategy construction and multi-objective rPATL* have been adapted from [42].

5.1 Syntax and semantics

We begin by providing the syntax and semantics for the multi-objective version of rPATL.

[2]The decidability of the model checking problem for general strategies remains open.

Definition 9 (Multi-objective rPATL) *The syntax of the multi-objective rPATL is given by the following grammar:*

$$\begin{aligned}\phi &::= \top \mid a \mid \neg\phi \mid \phi \wedge \phi \mid \langle\langle C \rangle\rangle \theta \\ \theta &::= \mathsf{P}_{\bowtie q}[\psi] \mid \mathsf{R}^{r}_{\bowtie x}[\mathsf{F}^{c}\phi] \mid \theta \wedge \theta \mid \theta \vee \theta \mid \neg\theta \\ \psi &::= \mathsf{X}\,\phi \mid \phi\,\mathsf{U}^{\leq k}\,\phi \mid \phi\,\mathsf{U}\,\phi\end{aligned}$$

where $a \in AP$, $C \subseteq \Pi$, $\bowtie \in \{<, \leq, \geq, >\}$, $q \in \mathbb{Q} \cap [0,1]$, $x \in \mathbb{Q}$, and the reward structure r is either $r : S \to \mathbb{Q}_{\geq 0}$ or $r : S \to \mathbb{Q}_{\leq 0}$.

Multi-objective rPATL extends the logic rPATL by allowing a boolean combination of objectives to appear within the coalition operator. We include only one type of the reward operator $\mathsf{R}^{r}_{\bowtie x}[\mathsf{F}^{c}\phi]$ representing the expected total reward. The type of reward operator where $\star{=}\infty$ can be expressed in multi-objective rPATL, e.g., $\mathsf{R}^{r}_{\leq x}[\mathsf{F}^{\infty}\phi] \equiv (\mathsf{P}_{\geq 1}[\mathsf{F}\,\phi] \wedge \mathsf{R}^{r}_{\leq x}[\mathsf{F}^{c}\phi])$. We choose not to include $\star{=}0$ due its high model checking complexity already for the (single-objective) rPATL. As a technical convenience, we allow reward structures to be either non-negative ($r : S \to \mathbb{Q}_{\geq 0}$) or non-positive ($r : S \to \mathbb{Q}_{\leq 0}$). The reason for this is that, as we show later in Section 5.3, the games with multi-objective rPATL goals are not determined, and hence we cannot define the equivalent of Equations (4.1) for rPATL by swapping the coalition to its complement in order to convert a minimisation problem into a maximisation one (i.e., $\leq$ to $\geq$ and $<$ to $>$). Instead, we perform such conversion by changing the sign of the reward structure and the bound, i.e., $\langle\langle C \rangle\rangle \mathsf{R}^{r}_{\leq x}[\mathsf{F}^{c}\phi] \equiv \langle\langle C \rangle\rangle \mathsf{R}^{-r}_{\geq -x}[\mathsf{F}^{c}\phi]$.

Semantics

We define the semantics of the multi-objective rPATL for a stochastic multi-player game $\mathcal{G} = \langle \Pi, S, (S_{\bigcirc}, (S_i)_{i\in\Pi}), \Delta, AP, \chi \rangle$ as follows. For boolean connectives and path formulae the semantics is the same as for rPATL (see Definition 8). The semantics of the $\langle\langle C \rangle\rangle\theta$ operator is defined via the coalition game $\mathcal{G}_C$ (see Definition 7) as follows. For each state $s \in S$ of the game $\mathcal{G}$ we have $s \models \langle\langle C \rangle\rangle\theta$ if and only if in the coalition game $\mathcal{G}_C$ there exists a strategy $\sigma_{\Box} \in \Sigma_{\Box}$ for player $\Box$ such that, for all strategies $\sigma_{\Diamond} \in \Sigma_{\Diamond}$ of player $\Diamond$ we have that the boolean combination of objectives, θ, evaluates to *true*. Every objective that appears in θ evaluates to true if it is $\mathsf{P}_{\bowtie q}[\psi]$ and $\mathrm{Pr}_s(\psi) \bowtie q$ or if it is $\mathsf{R}^{r}_{\bowtie x}[\mathsf{F}^{c}\phi]$ and $\mathbb{E}_s[rew(r, Sat(\phi))] \bowtie x$ and to *false* otherwise, where $rew(r, Sat(\phi)) \overset{\text{def}}{=} rew(r, c, Sat(\phi))$. Some example multi-objective rPATL formulae are listed below.

- $\langle\langle \{printer, assembler\} \rangle\rangle (\mathsf{P}_{\geq 1}[\mathsf{F}\,\mathsf{item}] \wedge \mathsf{R}^{cost}_{\leq 50}[\mathsf{F}^{c}\,\mathsf{item}] \wedge \mathsf{R}^{quality}_{\geq 100}[\mathsf{F}^{c}\,\mathsf{item}])$ – "*printer* and *assembler* have a joint strategy to produce an item with probability 1 at an expected cost at most 50 and expected quality at least 100."

- $\langle\langle\{robot1, robot2\}\rangle\rangle(\mathsf{P}_{\geq 1}[\mathsf{F}\,\mathsf{target}] \vee (\mathsf{P}_{\geq 0.9}[\mathsf{F}\,\mathsf{target}] \wedge \mathsf{R}^{energy}_{\leq 50}[\mathsf{F}^c\,\mathsf{target}]))$ – "robots 1 and 2 have a strategy that either locates the target with probability 1, or locates the target with probability at least 0.9 but has expected energy expenditure of at most 50."

We refer to multi-objective rPATL restricted to conjunctions only within $\langle\langle C\rangle\rangle\theta$ operator as *conjunctive rPATL*, and the variant restricted to disjunctions only is referred to as *disjunctive rPATL*.

Additional notions

Before presenting the main results of the chapter we introduce some notions that we use throughout. Given a vector $\vec{x} \in \mathbb{R}^n$, we use x_i to refer to its i-th component, where $1 \leq i \leq n$, and define the norm $\|\vec{x}\| \overset{\text{def}}{=} \sum_{i=1}^{n} |x_i|$. Given a number $y \in \mathbb{R}$, we use $\vec{x} \pm y$ to denote the vector $(x_1 \pm y, x_2 \pm y, \ldots, x_n \pm y)$. Given two vectors $\vec{x}, \vec{y} \in \mathbb{R}^n$, the *dot product* of $\vec{x}$ and $\vec{y}$ is defined by $\vec{x} \cdot \vec{y} = \sum_{i=1}^{n} x_i \cdot y_i$, and the comparison operator $\leq$ on vectors is defined to be the elementwise ordering. The sum of two sets of vectors $X, Y \subseteq \mathbb{R}^n$ is defined by $X + Y = \{\vec{x} + \vec{y} \mid \vec{x} \in X, \vec{y} \in Y\}$. Given a set X, we define the *downward closure* of X as $\mathsf{dwc}(X) \overset{\text{def}}{=} \{\vec{y} \mid \exists \vec{x} \in X \,.\, \vec{y} \leq \vec{x}\}$ and the *upward closure* as $\mathsf{up}(X) \overset{\text{def}}{=} \{\vec{y} \mid \exists \vec{x} \in X \,.\, \vec{x} \leq \vec{y}\}$, and *convex closure* as $\mathsf{conv}(X) \overset{\text{def}}{=} \{\vec{z} \mid \exists \alpha \in [0,1] \,.\, \exists \vec{x}, \vec{y} \in X \,.\, \vec{z} = \alpha \cdot \vec{x} + (1 - \alpha) \cdot \vec{y}\}$. We denote by $\mathbb{R}_{\pm\infty}$ the set $\mathbb{R} \cup \{\infty, -\infty\}$, and we define the operations $\cdot$ and $+$ in the expected way, defining $0 \cdot x = 0$ for all $x \in \mathbb{R}_{\pm\infty}$ and leaving $-\infty + \infty$ undefined. We also define the function $\mathsf{sign}(x) : \mathbb{R}_{\pm\infty} \to \mathbb{N}$ to be 1 if $x > 0$, -1 if $x < 0$ and 0 if $x = 0$.

Parametrised formulae. We introduce the parametrised notation for multi-objective rPATL formulae, denoted by $\varphi(\vec{r}, \vec{v})$ where $\vec{r}$ is a vector of reward functions and $\vec{v}$ is a vector of bounds that appear in the formula, e.g., $\varphi(\vec{r}, \vec{v}) = \langle\langle C\rangle\rangle(\mathsf{R}^{r_1}_{\geq v_1}[\mathsf{F}^c\mathsf{goal}] \wedge \mathsf{P}_{\leq v_2}[\mathsf{F}\,\mathsf{failure}])$ or $\varphi(\vec{r}, \vec{v}) = \langle\langle C\rangle\rangle(\bigwedge_{i=1}^{n} \mathsf{R}^{r_i}_{\geq v_i}[\mathsf{F}^c\bot])$. We assume that each element of $\vec{r}$ and $\vec{v}$ of $\varphi(\vec{r}, \vec{v}) = \langle\langle C\rangle\rangle\theta$ appears at most once in the formula θ. We also assume that formula θ contains no negations, these can be easily removed by switching the comparison operator, e.g., $\neg\mathsf{P}_{\geq v}[\mathsf{F}\,\phi] \equiv \mathsf{P}_{<v}[\mathsf{F}\,\phi]$. If the query does not contain reward operators, we omit $\vec{r}$ and use $\varphi(\vec{v})$ only. We also use $\mathsf{dir}(\vec{v}) = (d_1, \ldots, d_n)$ where d_i denotes the "direction" of the objective, where $d_i = 1$ if the objective in which v_i appears is maximisation (i.e., $\geq$ or $>$), and $d_i = -1$ if the objective in which v_i appears is minimisation (i.e., $\leq$ or $<$), e.g., for $\varphi(\vec{r}, \vec{v}) = \langle\langle C\rangle\rangle(\mathsf{R}^{r_1}_{\geq v_1}[\mathsf{F}^c\mathsf{goal}] \wedge \mathsf{P}_{\leq v_2}[\mathsf{F}\,\mathsf{failure}])$ we have $\mathsf{dir}(\vec{v}) = (1, -1)$.

Pareto sets. Given an SMG $\mathcal{G}$ and a multi-objective rPATL formula $\varphi(\vec{r}, \vec{v})$ we say that a vector $\vec{v}$ is a *Pareto vector* for a vector of reward functions $\vec{r}$ if and only if for all $\varepsilon > 0$ we have that $\varphi(\vec{r}, \vec{v} - \varepsilon \cdot \mathsf{dir}(\vec{v}))$ is true and $\varphi(\vec{r}, \vec{v} + \varepsilon \cdot \mathsf{dir}(\vec{v}))$ is false. The *Pareto set* is the set of all such vectors $\vec{v}$. Given $\varepsilon > 0$, an *ε-approximation of a Pareto set* is the set of vectors Q satisfying that, for any $\vec{w} \in Q$, there is a vector $\vec{v}$ in the Pareto set such that $\|\vec{v} - \vec{w}\| \leq \varepsilon$, and for every $\vec{v}$ in the Pareto set there is a vector $\vec{w} \in Q$ such that $\|\vec{v} - \vec{w}\| \leq \varepsilon$.

Optimal and $\varepsilon-$optimal strategies. Given an SMG $\mathcal{G}$ and a multi-objective rPATL formula $\varphi(\vec{r}, \vec{v}) = \langle\langle C \rangle\rangle \theta$ we say that *optimal* strategies exist for coalition C for the query if, for every point $\vec{v}$ in its Pareto set, $\varphi(\vec{r}, \vec{v})$ is true; and we say that $\varepsilon-$optimal strategies exist if for every such point and for all $\varepsilon > 0$ the formula $\varphi(\vec{r}, \vec{v} - \varepsilon \cdot \mathsf{dir}(\vec{v}))$ is true. Note that $\varepsilon-$strategies exist for all points in the Pareto set by definition.

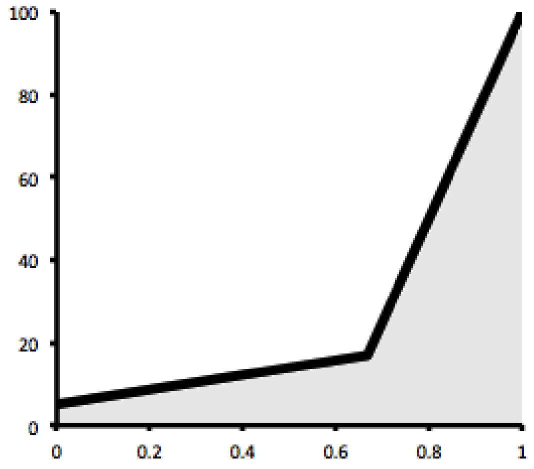
Vector reward functions. We extend the notion of reward function to vectors. Given a multi-objective rPATL formula $\varphi(\vec{r}, \vec{v})$, for a state $s \in S$ we define a vector reward structure by $\vec{r}(s) = (r_1(s), \ldots, r_n(s))$, where n is the length of $\vec{r}$. We can now define a vector reward function for a path $\lambda \in \Omega_{\mathcal{G}}$ and a set of target states $T \subseteq S$ of the SMG $\mathcal{G}$ by $rew(\vec{r}, T)(\lambda) \overset{\text{def}}{=} \sum_{j=0}^{k-1} \vec{r}(\lambda_j)$ where $k = \min\{j \mid \lambda_j \in T\}$. The *expected reward* from state $s \in S$ of $\mathcal{G}$ under a strategy profile $\sigma_0, \ldots, \sigma_n$ is the expected value of this reward function, i.e., $\mathbb{E}_s^{\sigma_0,\ldots,\sigma_n}[rew(\vec{r}, T)]$.

Example 11 *In this example we show how the multi-objective rPATL can be used to analyse the SMG from Figure 4.1. Consider the rPATL formula* $\langle\langle C \rangle\rangle \mathsf{P}_{\geq 1}[\top \mathbin{\mathsf{U}} \mathsf{T}_1]$ *from Example 7 which is satisfied for coalition* $C = \{\Box, \Diamond\}$ *in state* s_0*. Now if we want also to bound the resources used to complete the task (modelled by the 'cost' reward function, where* $cost(s_{15}) = cost(s_{11}) = 50$ *and* $cost(s_1) = 5$*), we can express this in multi-objective rPATL as* $\langle\langle C \rangle\rangle(\mathsf{P}_{\geq 1}[\mathsf{F}\, \mathsf{T}_1] \wedge \mathsf{R}^{cost}_{\leq 50}[\mathsf{F}^c\, \mathsf{T}_1])$*. This formula no longer holds for* $C = \{\Box, \Diamond\}$*. To see this first observe that player* $\Box$ *cannot take action 'init-1*' and player* $\Diamond$ *cannot take action 'idle' because the first objective would fail, and player* $\triangle$ *can always take action 'idle' in a state* s_3*, making sure that, in order to satisfy the objective* $\mathsf{P}_{\geq 1}[\mathsf{F}\, \mathsf{T}_1]$*, player* $\Box$ *has to take action 'init-1' every time the game returns to* s_0*, and thus the expected value for cost is* $50 + 0.5 \cdot 50 + 0.5^2 \cdot 50 + \cdots = 100$*. The grand coalition (i.e.,* $C = \{\Box, \Diamond, \triangle\}$*) is required to achieve the formula, because* $\triangle$ *can take action 'init-2' in* s_3 *making sure that the objective* $\mathsf{P}_{\geq 1}[\mathsf{F}\, \mathsf{T}_1]$ *is satisfied in 4 steps using exactly 50 units of the resource 'cost' and thus satisfying the objective* $\mathsf{R}^{cost}_{\leq 50}[\mathsf{F}^c\, \mathsf{T}_1]$ *as well.*

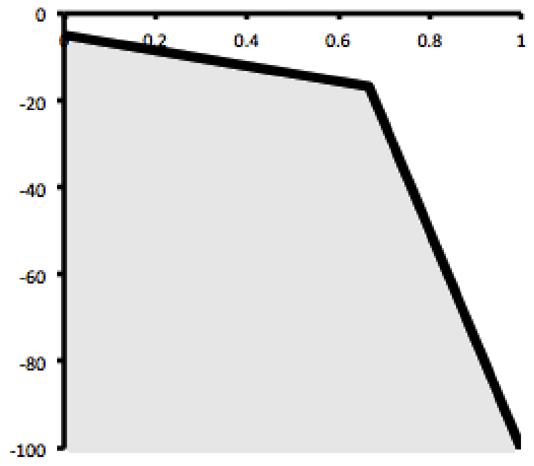
We note that resource bounding using multiple objectives is not the same as using the reward-bounded until discussed in Section 4.6, for example, multi-objective rPATL formula $\langle\langle \{\Box, \Diamond\} \rangle\rangle(\mathsf{P}_{\geq 1}[\mathsf{F}\, \mathsf{T}_1] \wedge \mathsf{R}^{cost}_{\leq 100}[\mathsf{F}^c\, \mathsf{T}_1])$ *holds in* s_0*, but* $\langle\langle \{\Box, \Diamond\} \rangle\rangle \mathsf{P}_{\geq 1}[\top \mathbin{\mathsf{U}}^{cost}_{\leq 100} \mathsf{T}_1]$ *does not. This is because the reward-bounded until does not allow to accumulate more than 100 of reward 'cost' along any path (hence, explicitly bounding the expectation), which is not restricted by the reward objective* $\mathsf{R}^{cost}_{\leq 100}[\mathsf{F}^c\, \mathsf{T}_1]$.

Also, in multi-objective rPATL we can consider several reward functions in the same formula. Let us define a reward function representing the number of steps to termination of the game, i.e., steps$(s) = 1$ is $s \notin \mathsf{Term}$. Now we can use this to relax the multi-objective formula considered earlier so that the restriction on resources only applies if the expected number of steps executed by the protocol is less than 6, and otherwise it is ignored; this can be expressed by the following multi-objective rPATL formula: $\langle\langle\{\Box,\Diamond\}\rangle\rangle(\mathsf{P}_{\geq 1}[\mathsf{F}\,\mathsf{T}_1] \wedge (\mathsf{R}^{cost}_{\leq 50}[\mathsf{F}^c\,\mathsf{T}_1] \vee \mathsf{R}^{steps}_{>5}[\mathsf{F}^c\,\bot]))$. *Suppose players $\Box$ and $\Diamond$ always play actions 'init-1' and 'join-1', respectively (to make sure that $\mathsf{P}_{\geq 1}[\mathsf{F}\,\mathsf{T}_1]$ is satisfied). When the game reaches s_3, if player $\triangle$ chooses action 'init-2', then the expected cost is exactly 50 and thus objective $\mathsf{R}^{cost}_{\leq 50}[\mathsf{F}^c\,\mathsf{T}_1]$ is true; if he chooses action 'idle', then the expected cost becomes greater than 50, but, at the same time, the expected number of steps becomes greater than 5 and thus the objective $\mathsf{R}^{steps}_{>5}[\mathsf{F}^c\,\bot]$ is satisfied. So, no matter which strategy $\triangle$ picks, one of the objectives in the disjunction is satisfied, in addition to the reachability objective, and thus the formula holds in s_0.*

Finally, we consider the parametric variant of the formula in a game starting from s_0: $\varphi(\vec{v}) = \langle\langle\{\Box,\Diamond\}\rangle\rangle(\mathsf{P}_{\geq v_1}[\mathsf{F}\,\mathsf{T}_1] \wedge \mathsf{R}^{cost}_{\leq v_2}[\mathsf{F}^c\,\mathsf{T}_1])$. *The Pareto set for the values of $\vec{v}$ is shown in Figure 5.1a. For any vector $\vec{u}$ in the area shaded in grey (including the black line), the*



(a) $\langle\langle\{\Box,\Diamond\}\rangle\rangle(\mathsf{P}_{\geq v_1}[\mathsf{F}\,\mathsf{T}_1] \wedge \mathsf{R}^{cost}_{\leq v_2}[\mathsf{F}^c\,\mathsf{T}_1])$



(b) $\langle\langle\{\Box,\Diamond\}\rangle\rangle(\mathsf{P}_{\geq v_1}[\mathsf{F}\,\mathsf{T}_1] \wedge \mathsf{R}^{-cost}_{\geq -v_2}[\mathsf{F}^c\,\mathsf{T}_1])$

Figure 5.1: Pareto sets for multi-objective rPATL formulae. Horizontal axis represents values for v_1 and vertical one represents values for v_2 ($-v_2$ in (b)).

formula $\varphi(\vec{u})$ is true, i.e., players $\Box$ and $\Diamond$ have a strategy to guarantee (at least) such a trade-off between objectives. Conversely, for any vector outside this set the formula is false, i.e., player $\triangle$ has a counter-strategy to any strategy proposed by the coalition of $\Box$ and $\Diamond$.

We use the computation of such sets as a key tool for multi-objective rPATL model *checking. The algorithms that we develop in this chapter require the sets to be convex and, in order to achieve this, we use negative rewards. For example, the multi-objective rPATL formula $\varphi(\vec{v})$ presented above is equivalent to the formula $\varphi'(\vec{v}) = \langle\langle\{\Box, \Diamond\}\rangle\rangle(\mathsf{P}_{\geq v_1}[\mathsf{F}\,\mathsf{T}_1] \wedge \mathsf{R}^{-cost}_{\geq -v_2}[\mathsf{F}^c\,\mathsf{T}_1])$, where the 'cost' reward function is replaced by the one which negates it, and the bound is also made negative (see Section 5.2 for a detailed discussion). The Pareto set for such formula is convex and is shown in Figure 5.1b.*

5.2 Model checking

In this section we present model checking algorithms for the multi-objective rPATL, which guarantee correctness of the result up to a specified precision. We restrict the class of games considered to *stopping* SMGs, i.e., the games that end up in a terminal state with probability 1 for any strategy profile of the players (see Section 3.1.3 for discussion about stopping games).

The general model checking algorithm is the same as for single-objective rPATL (see Section 4.2.1). The only difference is in the evaluation of the operator $\langle\langle C\rangle\rangle\theta$, for which we also need to solve a coalition game, but this time with multiple objectives. This section focuses on model checking of this operator.

The high-level strategy for model checking the operator $\langle\langle C\rangle\rangle\theta$ is similar to the one for (single-objective) rPATL. In the algorithms for rPATL, we computed the optimal values for probability and expected reward in the coalition game and then compared them to the given bound. For multi-objective rPATL we compute the Pareto sets for the objectives in the formula, and then check whether there is a point in this set, which makes the formula *true.*

The remainder of the section is structured follows. We begin by showing, in Section 5.2.1, that model checking of $\langle\langle C\rangle\rangle\theta$ in a game $\mathcal{G}$ can be reduced to model checking of a formula $\langle\langle C\rangle\rangle\theta'$ in a game $\mathcal{G}'$ where θ' contains only objectives that maximise the expected total reward with no target states, i.e., of the form $\mathsf{R}^r_{\geq x}[\mathsf{F}^c\bot]$ or $\mathsf{R}^r_{>x}[\mathsf{F}^c\bot]$; and hence, in the following sections we can focus on model checking formulae in this form. In Section 5.2.2 we establish the relationship between the strategies for multi-objective rPATL and conjunctive rPATL that allows us to combine the model checking algorithms for conjunctive and disjunctive rPATL to obtain the approximation algorithm for model checking general multi-objective rPATL. We present these algorithms in Section 5.2.3.

5.2.1 Reduction to maximisation of expected reward

In this section we show that model checking $\langle\langle C\rangle\rangle\theta$ in a game $\mathcal{G}$ can be reduced to model checking of formula $\langle\langle C\rangle\rangle\theta'$ in a game $\mathcal{G}'$ where θ' contains only expected total reward objectives with no target states, i.e., of the form $\mathsf{R}^{r}_{\rhd x}[\mathsf{F}^{c}\bot]$, where $\rhd \in \{\geq, >\}$. We present four steps, which can be successively applied to perform the construction of $\mathcal{G}'$ and θ':

1. Replace $\mathsf{R}^{r}_{\bowtie x}[\mathsf{F}^{c}\phi]$ with $\mathsf{R}^{r'}_{\bowtie x}[\mathsf{F}^{c}\bot]$;

2. Replace $\mathsf{P}_{\bowtie x}[\psi]$ with $\mathsf{R}^{r}_{\bowtie x}[\mathsf{F}^{c}\bot]$;

3. Replace $\neg\mathsf{R}^{r}_{\bowtie x}[\mathsf{F}^{c}\phi]$ with $\mathsf{R}^{r}_{\bowtie' x}[\mathsf{F}^{c}\bot]$;

4. Replace $\mathsf{R}^{r}_{\lhd x}[\mathsf{F}^{c}\bot]$ with $\mathsf{R}^{r'}_{\rhd x'}[\mathsf{F}^{c}\bot]$, where $\lhd \in \{\leq, <\}$ and $\rhd \in \{\geq, >\}$.

We now proceed with describing the transformation steps.

"$\mathsf{R}^{r}_{\bowtie x}[\mathsf{F}^{c}\phi]$ to $\mathsf{R}^{r'}_{\bowtie x}[\mathsf{F}^{c}\bot]$"

We start by showing how to, given a game $\mathcal{G}$ and a multi-objective rPATL formula $\langle\langle C\rangle\rangle\theta$ containing the $\mathsf{R}^{r}_{\bowtie x}[\mathsf{F}^{c}\phi]$ operator, to construct a game $\mathcal{G}'$ and a formula $\langle\langle C\rangle\rangle\theta'$ where $\mathsf{R}^{r}_{\bowtie x}[\mathsf{F}^{c}\phi]$ is replaced by $\mathsf{R}^{r'}_{\bowtie x}[\mathsf{F}^{c}\bot]$. Let us define $\mathcal{G}'$ as follows. Let $S' = \{(s, I) \mid s \in S \text{ and } I \in \{0, 1\}\}$, where (s, I) is assigned to the player that controlled state s in $\mathcal{G}$. The set of atomic propositions remains the same (i.e., $AP'{=}AP$) and the labelling function is defined by $\chi'((s, I)) = \chi(s)$. The transition function Δ' is defined as follows. $\Delta'((s, I), (s', I')) = x$ whenever the conditions below are satisfied:

- $\Delta(s, s') = x$;

- if $s \in Sat(\phi)$ and $I = 0$ then $I' = 1$;

- if $s \notin Sat(\phi)$ and $I = 0$ then $I' = 0$;

- if $I = 1$ then $I' = 1$.

For all other values Δ' returns 0. The reward function r' is defined by $r'((s, I)) = 0$ if $s \in Sat(\phi)$ or $I = 1$, and $r'((s, I)) = r(s)$ otherwise. The above construction ensures that, when starting from $(s, 0)$, the reward is accumulated only until the state satisfying ϕ is reached and, from then on, no reward is accumulated. It follows from the correspondence of strategies between the two games that, for a state s of $\mathcal{G}$, $\langle\langle C\rangle\rangle\theta$ is satisfied if and only if $\langle\langle C\rangle\rangle\theta'$ is satisfied in state $(s, 0)$ of $\mathcal{G}'$.

“$\mathsf{P}_{\bowtie x}[\psi]$ to $\mathsf{R}^r_{\bowtie x}[\mathsf{F}^c\bot]$”

We continue by showing how to, given a game $\mathcal{G}$ and a multi-objective rPATL formula $\langle\langle C\rangle\rangle\theta$ containing $\mathsf{P}_{\bowtie x}[\psi]$ operator, construct a game $\mathcal{G}'$ and formula $\langle\langle C\rangle\rangle\theta'$ where $\mathsf{P}_{\bowtie x}[\psi]$ is replaced by $\mathsf{R}^r_{\bowtie x}[\mathsf{F}^c\bot]$. Let us define $\mathcal{G}'$ as follows. Let $S' = \{(s, I, J) \mid s \in S \text{ and } I, J \in \{0, 1\}\}$, where (s, I, J) is assigned to the player that controlled state s in $\mathcal{G}$. The set of atomic propositions remains the same (i.e., AP'=AP) and labelling function is defined by $\chi'((s, I, J)) = \chi(s)$. There are two cases to consider for the transition function.

1. If $\psi = \mathsf{X}\,\phi$, then transition function Δ' is defined as follows. $\Delta'((s, I, J),(s', I', J')) = x$ whenever the conditions below are satisfied:

   - $\Delta(s, s') = x$;
   - $I' = 1$;
   - if $I = J = 0$ and $s' \in \mathit{Sat}(\phi)$ then $J' = 0$ else $J' = 1$.

   For all other values Δ' returns 0.

2. If $\psi = \phi_1 \,\mathsf{U}\, \phi_2$, then Δ' is defined as follows. $\Delta'((s, I, J),(s', I', J')) = x$ whenever the following conditions hold:

   - $\Delta(s, s') = x$;
   - if $I = J = 0$ and $s' \in \mathit{Sat}(\phi_2)$, then $I' = 1$ and $J' = 0$;
   - if $I = J = 0$ and $s' \notin \mathit{Sat}(\phi_1) \cup \mathit{Sat}(\phi_2)$ then $I' = J' = 1$.
   - if $I = J = 0$ and $s' \notin \mathit{Sat}(\phi_2)$ and $s' \in \mathit{Sat}(\phi_1)$ then $I' = J' = 0$;
   - if $I \neq 0$ or $J \neq 0$ then $I' = J' = 1$.

3. For $\psi = \phi_1 \,\mathsf{U}^{\leq k}\, \phi_2$ we assume that the step counter has been embedded in the state space and the formula has been transformed into the unbounded-until as in Section 4.3, and hence the transformation of step 2 can be applied.

The reward function r is defined by $r(s, I, J) = 1$ if $I = 1$ and $J = 0$, and otherwise $r(s, I, J) = 0$. For a state s of $\mathcal{G}$, $\langle\langle C\rangle\rangle\theta$ is satisfied if and only if $\langle\langle C\rangle\rangle\theta'$ is satisfied in $(s, 0, 0)$ of $\mathcal{G}'$. To see this it suffices to observe that, due to the constructions above, the reward function r assigns reward 1 to the state in $\mathcal{G}'$ if and only if the formula ψ is satisfied for the first time and assign 0 otherwise.

“$\neg \mathsf{R}^{r}_{\bowtie x}[\mathsf{F}^c\phi]$ to $\mathsf{R}^{r}_{\bowtie' x}[\mathsf{F}^c\bot]$”

We can remove the negation by switching the direction of optimisation, i.e., we have $\neg\mathsf{R}^r_{\geq x}[\mathsf{F}^c\phi] \equiv \mathsf{R}^r_{<x}[\mathsf{F}^c\phi]$, $\neg\mathsf{R}^r_{>x}[\mathsf{F}^c\phi] \equiv \mathsf{R}^r_{\leq x}[\mathsf{F}^c\phi]$, $\neg\mathsf{R}^r_{\leq x}[\mathsf{F}^c\phi] \equiv \mathsf{R}^r_{>x}[\mathsf{F}^c\phi]$ and $\neg\mathsf{R}^r_{<x}[\mathsf{F}^c\phi] \equiv \mathsf{R}^r_{\geq x}[\mathsf{F}^c\phi]$.

“$\mathsf{R}^{r}_{\lhd x}[\mathsf{F}^c\phi]$ to $\mathsf{R}^{r'}_{\rhd x'}[\mathsf{F}^c\phi]$”

To perform the transformation, we use the change of sign for the reward function and the bound. If $\langle\langle C\rangle\rangle\theta$ contains an objective $\mathsf{R}^r_{\leq x}[\mathsf{F}^c\phi]$, we can replace it by a goal $\mathsf{R}^{r'}_{\geq x'}[\mathsf{F}^c\phi]$, where $x' = -x$ and the reward function is defined by $r'(s) = -r(s)$. The above construction can be performed because, for any strategy profile of the players, we have that $\mathbb{E}_s[rew(\vec{r}, Sat(\phi))] \leq x \equiv -\mathbb{E}_s[rew(\vec{r}, Sat(\phi))] \geq -x \equiv \mathbb{E}_s[rew(\vec{r}, Sat(\phi))] \geq -x = \mathbb{E}_s[rew(\vec{r}, Sat(\phi))] \geq x'$. The same construction can be applied for the strict inequalities.

Remark 3 *Note that the transformation provided above may cause the number of states in the resulting game $\mathcal{G}'$ to be of the order $\mathcal{O}(|S|{\cdot}2^n)$ where S is the set of states in the original game $\mathcal{G}$ and n is the number of objectives in $\langle\langle C\rangle\rangle\theta$. This effect, however, is already observed for the multi-objective optimisation of non-terminal state reachability in Markov decision processes [57].*

5.2.2 Reduction to conjunctive rPATL

In this section we consider model checking of the multi-objective rPATL formulae expressed in the positive conjunctive normal form (CNF), i.e., formulae having the form $\langle\langle C\rangle\rangle \bigwedge_{i=1}^n \bigvee_{j=1}^m \mathsf{R}^{q_{i,j}}_{\rhd_{i,j} u_{i,j}}[\mathsf{F}\,\bot]$, where $\rhd_{i,j} \in \{\geq, >\}$. Note that, using the transformations from the previous section together with standard rules of propositional logic, any multi-objective rPATL formula can be transformed into one in such form. In order to conveniently reference the reward functions and bounds of the formula expressed in CNF as above we introduce a few notations. Since we only consider approximation algorithms here, we assume that formula contain only non-strict inequalities (i.e., $\geq$). For a vector $\vec{v}$ we denote by $v_{i,j}$ its $v_{(i-1)n+j}$th element. Also by $\vec{v}_i$ we denote the sub-vector $(v_{i,1}, \ldots, v_{i,m})$ of $\vec{v}$. The latter corresponds to a vector of reward functions or bounds within a single conjunct. The results presented in this section have been published in [40].

The aim of this section is to prove the following theorem.

Theorem 7 *Given a stochastic multi-player game $\mathcal{G}$ and a multi-objective rPATL formula $\varphi(\vec{q}, \vec{u}) = \langle\langle C\rangle\rangle \bigwedge_{i=1}^n \bigvee_{j=1}^m \mathsf{R}^{q_{i,j}}_{\geq u_{i,j}}[\mathsf{F}^c\bot]$, let $\sigma_\Box$ be a strategy of player $\Box$ in coalition game $\mathcal{G}_C$. The following two conditions are equivalent.*

- *The strategy $\sigma_\Box$ achieves $\varphi(\vec{q},\vec{u})$.*

- *For all $\varepsilon > 0$ there are non-zero vectors $\vec{x}_1, \dots \vec{x}_n \in \mathbb{R}^m_{\geq 0}$, such that $\sigma_\Box$ achieves $\vartheta(\vec{r},\vec{v}) = \langle\langle C\rangle\rangle \bigwedge_{i=1}^n \mathsf{R}^{r_i}_{\geq v_i}[\mathsf{F}\,\bot]$, where $r_i(s) = \vec{x}_i \cdot \vec{q}_i(s)$ and $v_i = \vec{x}_i \cdot (\vec{u}_i - \varepsilon)$ for all $1 \leq i \leq n$.*

Before presenting the proof of the theorem we discuss how it can be used to design the approximation algorithm for multi-objective rPATL model checking. Let us assume that we have an algorithm that can compute the Pareto set for the formula $\vartheta(\vec{r},\vec{v}) = \langle\langle C\rangle\rangle \bigwedge_{i=1}^n \mathsf{R}^{r_i}_{\geq v_i}[\mathsf{F}\,\bot]$ (this algorithm is presented in the next section). Let Q be this Pareto set. Then, every combination of vectors $\vec{x}_1, \dots \vec{x}_n \in \mathbb{R}^m_{\geq 0}$ provides us with the set of vectors $U_{\vec{x}_1,\dots\vec{x}_n} \stackrel{\text{def}}{=} \{\vec{v} \in \mathbb{Q}^{n\times m}_{\geq 0} \mid \exists\, \vec{d} \in Q\,.\,\forall\, 0 \leq i \leq n\,.\,\vec{x}_i \cdot \vec{v}_i \leq d_i\}$, such that for every vector $\vec{v}$ in this set, we have that $\varphi(\vec{q},\vec{v}) = \langle\langle C\rangle\rangle \bigwedge_{i=1}^n \bigvee_{j=1}^m \mathsf{R}^{q_{i,j}}_{\geq v_{i,j}}[\mathsf{F}\,\bot]$ is satisfied. The more vectors different combinations of vectors we take, the more Pareto points we obtain, and, as we show in the next section, we can construct a set X containing combinations of vectors $\vec{x}_1, \dots \vec{x}_n \in \mathbb{R}^m_{\geq 0}$ such that a set $\bigcup_{\vec{x}_1,\dots\vec{x}_n \in X} U_{\vec{x}_1,\dots\vec{x}_n}$ is an $\varepsilon-$approximation of the Pareto set for $\varphi(\vec{q},\vec{v})$. So, effectively, Theorem 7 enables us to approximate the Pareto set for any multi-objective rPATL formula with the algorithm computing the Pareto set for the conjunctive rPATL.

In order to prove Theorem 7 we use the following reformulation of the separating hyperplane theorem.

Lemma 5 *Let $W \subseteq \mathbb{R}^m_{\pm\infty}$ be a convex set satisfying the following. For all j, whenever there is $\vec{x} \in W$ such that $\mathsf{sign}(x_j) \geq 0$ (resp. $\mathsf{sign}(x_j) \leq 0$), then $\mathsf{sign}(y_j) \geq 0$ (resp. $\mathsf{sign}(y_j) \leq 0$) for all $\vec{y} \in W$. Let $\vec{z} \in \mathbb{R}^m$ be a point which does not lie in the closure of $\mathsf{up}(W)$. Then there is a* non-zero *vector $\vec{x} \in \mathbb{R}^m$ such that the following conditions hold.*

1. *For all $1 \leq j \leq m$ we have $x_j \geq 0$.*

2. *For all $1 \leq j \leq m$, if there is $\vec{w} \in W$ satisfying $w_j = -\infty$, then $x_j = 0$.*

3. *For all $\vec{w} \in W$, the product $\vec{w} \cdot \vec{x}$ is defined and satisfies $\vec{w} \cdot \vec{x} \geq \vec{z} \cdot \vec{x}$.*

Proof. Let $I \subseteq \{1, \dots, m\}$ be the set of indices such that all $\vec{w} \in W$ satisfy $\mathsf{sign}(w_i) \leq 0$. Let U be the closure of $\mathsf{up}(W) \cap \mathbb{R}^m$. If $U = \emptyset$, then we define $\vec{x}$ by $x_i = 0$ if $i \in I$ and $x_i = 1$ otherwise. For any $\vec{w} \in W$, we have

$$\vec{w} \cdot \vec{x} = \sum_{i \in I} w_i \cdot x_i + \sum_{i \notin I} w_i \cdot x_i$$

where the left summand is 0. We argue that the right summand must be positive. Suppose otherwise, then it must be the case that all w_i for $i \notin I$ are real numbers. But then we can replace any $-\infty$ in $\vec{w}$ by any real number, and get a vector which by the definition lies in U, contradicting the property that $U = \emptyset$.

Suppose $U \neq \emptyset$. Note that U is convex and let $\tau > 0$ be the smallest number such that $\vec{z} + \tau$ lies in the closure of U. Denote $\bar{\vec{z}} := \vec{z} + \tau$. By the separating hyperplane theorem [64], there is some non-zero vector $\vec{y} \in \mathbb{R}^m$, s.t., for all $\vec{w} \in U$, we have $\vec{w} \cdot \vec{y} \geq \bar{\vec{z}} \cdot \vec{y}$. We show that the vector $\vec{y}$ satisfies the condition 1, i.e., that all components of $\vec{y}$ are non-negative. Assume, for the sake of contradiction, that for some $1 \leq j \leq m$ we have $y_j < 0$. Let $\vec{w}$ be any point from U. Let $d = \vec{w} \cdot \vec{y} - \bar{\vec{z}} \cdot \vec{y}$, and let $\vec{w}'$ be the vector which is obtained from $\vec{w}$ by replacing jth coordinate with $w_j + \frac{d+1}{-y_j}$. Since $\frac{d+1}{-y_j}$ is positive and U is upwards closed in $\mathbb{R}^m_{\pm\infty}$, we have $\vec{w}' \in U$. So, since $\bar{\vec{z}} \cdot \vec{y} \leq \vec{w}' \cdot \vec{y}$, we have

$$\begin{aligned} \vec{w}' \cdot \vec{y} = \sum_h w'_h \cdot y_h &= \frac{d+1}{-y_j} \cdot y_j + \sum_h w_h \cdot y_h \\ &= -(d+1) + \vec{w} \cdot \vec{y} = \bar{\vec{z}} \cdot \vec{y} - 1, \end{aligned}$$

which is a contradiction.

Let $\varepsilon := \bar{\vec{z}} \cdot \vec{y} - \vec{z} \cdot \vec{y}$, we have $\varepsilon > 0$. We define $\vec{x}$ by putting $x_i = y_i$ for $i \in I$, and $x_i = y_i + \frac{\varepsilon}{|\sum_{j=1}^m z_j| + 1}$ for $i \notin I$. Because $\vec{y} \geq 0$, the vector $\vec{x}$ satisfies the condition 1. We show that $\vec{x}$ satisfies the condition 2. of the lemma. Let $L \subseteq \{1, \ldots, m\}$ be the set of indices such that $l \in L$ if and only if there is $\vec{u} \in W$ with $u_l = -\infty$. Note that $L \subseteq I$. Since $x_l = y_l$ for all $l \in L$, it suffices to show that $y_l = 0$ for all $l \in L$. If $L = \emptyset$, there is nothing we need to prove. Otherwise, because W is convex, there is a vector $\vec{u} \in W$ with $u_l = -\infty$ for all $l \in L$, and so for arbitrary $\alpha \in \mathbb{R}^m$ the set U contains the vector $\vec{u}^\alpha$ defined by $u^\alpha_l = u_l$ if $l \in L$ and $u^\alpha_l = \alpha$ otherwise. Then $\lim_{\alpha \to -\infty} \vec{x} \cdot \vec{u}^\alpha = -\infty$ if $y_l > 0$ for any $l \in L$, contradicting that $\vec{y} \cdot \vec{u}^\alpha \geq \vec{y} \cdot \vec{z}$ for all α.

Finally, we prove the condition 3. of the lemma. Let $\vec{w} \in W$. The product $\vec{w} \cdot \vec{x}$ is defined by the condition 2. of the lemma. Also,

$$\begin{aligned} \vec{w} \cdot \vec{x} &= \sum_{i \in I} w_i \cdot x_i + \sum_{i \notin I} w_i \cdot x_i = \sum_{i \in I} w_i \cdot y_i + \sum_{i \notin I} w_i \cdot (y_i + \frac{\varepsilon}{|\sum_{j=1}^m z_j| + 1}) \\ &= \vec{w} \cdot \vec{y} + \sum_{i \notin I} w_i \cdot \frac{\varepsilon}{|\sum_{j=1}^m z_j| + 1} \\ &\geq \vec{w} \cdot \vec{y} \geq \bar{\vec{z}} \cdot \vec{y} = \vec{z} \cdot \vec{y} + \varepsilon \\ &= \Big(\sum_{i \in I} z_i \cdot x_i + \sum_{i \notin I} z_i \cdot (x_i - \frac{\varepsilon}{|\sum_{j=1}^m z_j| + 1}) \Big) + \varepsilon \\ &= \vec{z} \cdot \vec{x} - \Big(\sum_{i \notin I} z_i \frac{\varepsilon}{|\sum_{j=1}^m z_j| + 1} \Big) + \varepsilon \geq \vec{z} \cdot \vec{x}. \end{aligned}$$

where the first inequality follows because all w_i are positive for $i \notin I$. $\square$

Proof of Theorem 7. We are now in the position to prove the main theorem. Consider the coalition game $\mathcal{G}_C$ and fix player $\Box$ strategy $\sigma_\Box \in \Sigma_\Box$. For the "$\Rightarrow$" direction, for all $1 \le i \le n$ define $R_s[i] \overset{\text{def}}{=} \{\vec{y} \in \mathbb{R}^m_{\pm\infty} \mid \exists \sigma_\Diamond \in \Sigma_\Diamond . \mathbb{E}_s[rew(\vec{q_i}, \emptyset)] = \vec{y}\}$. Fix $\varepsilon > 0$ and $1 \le i \le n$. Because $\sigma_\Box$ achieves $\varphi(\vec{u}, \vec{q})$ it must also achieve $\langle\langle C \rangle\rangle (\bigvee_{j=1}^m \mathsf{R}^{q_{i,j}}_{\ge u_{i,j}}[\mathsf{F}\,\bot])$. Hence, for every $\vec{y} \in \mathsf{up}(R_s[i])$ there is a j satisfying $y_j > u_{i,j} - \frac{\varepsilon}{2}$, and so $\vec{u}_i - \frac{\varepsilon}{2} \notin R_s[i]$. By Lemma 5, since $\vec{u}_i - \varepsilon$ is not in the closure of $\mathsf{up}(R_s[i])$, and since $R_s[i]$ satisfies the conditions of the lemma, we can obtain a vector $\vec{x}_i$ for $\mathsf{up}(R_s[i])$ and $\vec{u}_i - \varepsilon$. Fix any player $\Diamond$ strategy $\sigma_\Diamond \in \Sigma_\Diamond$. We have $\mathbb{E}_s[rew(\vec{q_i}, \emptyset)] \in \mathsf{up}(R_s)$, and it follows that

$$\mathbb{E}_s[rew(r_i, \emptyset)] = \int_{\lambda \in \Omega_{\mathcal{G},s}} \sum_{k=0}^{\infty} \sum_{j=1}^{m} x_{i,j} \cdot q_{i,j}(\lambda_k)\, d\mathrm{Pr}_s \overset{(*)}{=} \int_{\lambda \in \Omega_{\mathcal{G},s}} \sum_{j=1}^{m} x_{i,j} \cdot \sum_{k=0}^{\infty} q_{i,j}(\lambda_k)\, d\mathrm{Pr}_s$$
$$= \sum_{j=1}^{m} x_{i,j} \cdot \int_{\Omega_{\mathcal{G},s}} \sum_{k=0}^{\infty} q_{i,j}\, d\mathrm{Pr}_s = \vec{x}_i \cdot \mathbb{E}_s[rew(\vec{q_i}, \emptyset)] \ge \vec{x}_i \cdot (\vec{u}_i - \varepsilon) = v_i.$$

The equality marked with $(*)$ holds because for almost every λ we have $\sum_{k=0}^{\infty} \sum_{j=1}^{m} x_{i,j} \cdot q_{i,j}(\lambda_k) = \sum_{j=1}^{m} x_{i,j} \cdot \sum_{k=0}^{\infty} q_{i,j}(\lambda_k)$; this is true because for every j we either have $x_{i,j} = 0$, or the sum $\sum_{k=0}^{\infty} q_{i,j}(\lambda_k)$ is strictly greater than $-\infty$ for almost all λ.

For the "$\Leftarrow$" direction, for each $\varepsilon > 0$ we have non-zero vectors $\vec{x}_1, \ldots, \vec{x}_n \in \mathbb{R}^m_{\ge 0}$, such that $\sigma_\Box$ achieves $\vartheta(\vec{r}, \vec{v})$. Assume for the sake of contradiction that this $\sigma_\Box$ does not achieve $\varphi(\vec{u}, \vec{q})$. Then there exists a player $\Diamond$ strategy $\sigma_\Diamond$ and an index i such that for all j we have that $\mathbb{E}_s[rew(q_{i,j}, \emptyset)] = u_{i,j} - \tau_j < u_{i,j}$ for some $\tau_j > 0$ (and possibly $\tau_j = \infty$). Now, fix such a strategy $\sigma_\Diamond$, a corresponding index i, and let $\varepsilon = \frac{\min_j \tau_j}{2} < \infty$. By assumption, we can pick $\vec{x}_i$ such that $\sigma_\Box$ achieves $\vartheta(\vec{r}, \vec{v})$, and hence $\mathbb{E}_s[rew(r_i, \emptyset)] \ge \vec{x}_i \cdot (\vec{u}_i - \varepsilon)$. Consequently, $\mathbb{E}_s[rew(r_i, \emptyset)] = \vec{x}_i \cdot \mathbb{E}_s[rew(\vec{q_i}, \emptyset)]$ by the same argument as above. Thus, $\vec{x}_i \cdot \mathbb{E}_s[rew(\vec{q_i}, \emptyset)] \ge \vec{x}_i \cdot (\vec{u}_i - \varepsilon)$, and because $\vec{x}_i$ is non-zero and has no negative components, there must be a j such that $\mathbb{E}_s[rew(q_{i,j}, \emptyset)] \ge u_{i,j} - \varepsilon > u_{u,j} - \tau_j = \mathbb{E}_s[rew(q_{i,j}, \emptyset)]$, which implies a contradiction. $\square$

Remark 4 *Note that, in the case of disjunctive rPATL, i.e., formulae having the form* $\varphi(\vec{q}, \vec{u}) = \bigvee_{j=1}^m \mathsf{R}^{q_j}_{\ge u_j}[\mathsf{F}^c\bot]$*, Theorem 7 establishes a relationship between disjunctive rPATL and single-objective expected total reward: if a coalition strategy achieves* $\varphi(\vec{q}, \vec{u})$*, then for every* $\varepsilon > 0$ *there exists a non-zero vector* $\vec{x} \in \mathbb{R}^m_{\ge 0}$*, such that the strategy achieves* $\langle\langle C \rangle\rangle \mathsf{R}^r_{\ge v}[\mathsf{F}^c\bot]$*, where* $r(s) = \vec{x} \cdot \vec{q}(s)$ *and* $v = \vec{x} \cdot (\vec{u} - \varepsilon)$*. We later use this property to show that memoryless deterministic strategies are sufficient to achieve a disjunction of objectives (see Theorem 13).*

5.2.3 Computation of Pareto sets

In this section we present the algorithms for approximating Pareto sets for multi-objective rPATL in stopping games. We start the section by presenting the algorithm for conjunctive rPATL, followed by the algorithm for disjunctive rPATL. Finally, we describe how, using Theorem 7, the two can be combined to model check general multi-objective rPATL. The results presented in this section have been published in [40].

Given a stopping SMG $\mathcal{G}$, we denote by M the upper bound of the absolute value of the expected total reward for any objective, i.e., $M = |S| \cdot \frac{\max_{s \in S_i} |r_i(s)|}{p_{\min}^{|S|}}$ where $p_{\min}$ is the smallest positive probability assigned by Δ in $\mathcal{G}$. Also note that, in the theorem below, we relax the reward structure to assign any rational numbers to states.

Theorem 8 (Pareto sets: conjunctive rPATL) *For a stopping game $\mathcal{G}$ and a multi-objective rPATL formula $\varphi(\vec{r}, \vec{v}) = \langle\langle C \rangle\rangle \bigwedge_{i=1}^{n} \mathsf{R}_{\geq v_i}^{r_i} [\, \mathsf{F}^c \perp]$, an $\varepsilon-$approximation of the Pareto sets for all states can be computed in $k = |S| + \lceil |S| \cdot \frac{\ln(\varepsilon \cdot (n \cdot M)^{-1})}{\ln(1-\delta)} \rceil$ iterations of the operator F defined by*

$$F(X)(s) \stackrel{\text{def}}{=} \begin{cases} \mathit{dwc}(\mathit{conv}(\bigcup_{t \in \Delta(s)} X_t) + \vec{r}(s)) & \text{if } s \in \bigcup_{i \in C} \\ \mathit{dwc}(\bigcap_{t \in \Delta(s)} X_t + \vec{r}(s)) & \text{if } s \in \bigcup_{i \in \Pi \setminus C} \\ \mathit{dwc}(\sum_{t \in \Delta(s)} \Delta(s,t) \cdot X_t + \vec{r}(s)) & \text{if } s \in S_{\bigcirc}, \end{cases}$$

starting from the initial sets $X_s^0 \stackrel{\text{def}}{=} \{\vec{x} \in \mathbb{R}^n \,|\, \vec{x} \leq \vec{r}(s)\}$ for all $s \in S$.

Proof. We only provide a high level intuition behind the result, and the full proof of the Theorem 8 is presented in Appendix B. We start by explaining the intuition the operations of the functional (see Figure 5.2 for illustrations). For $s \in \bigcup_{i \in C}$, the coalition player can randomise between successor states to achieve any convex combination of the points achievable in the successor states, so the achievable points in s are in the convex closure of the union of successors. For $s \in \bigcup_{i \in \Pi \setminus C}$, a value in s is achievable by the coalition, if it is achievable in all successors, and hence we take the intersection. Finally, stochastic states $s \in S_{\bigcirc}$ are like the coalition-controlled states with a fixed probability distribution, and the set of achievable points is characterised by the weighted Minkowski sum of successor points. Having defined this one step relationship, we can show that performing k iterations of the functional F, the obtained set X_s^k for state s is exactly the set of all value that can be guaranteed by the coalition C along the paths of length k. Finally, because the game $\mathcal{G}$ is stopping, by picking k large enough, we can guarantee that Pareto points of X_s^k are within ε of the Pareto points for $\varphi(\vec{r}, \vec{v})$. $\square$

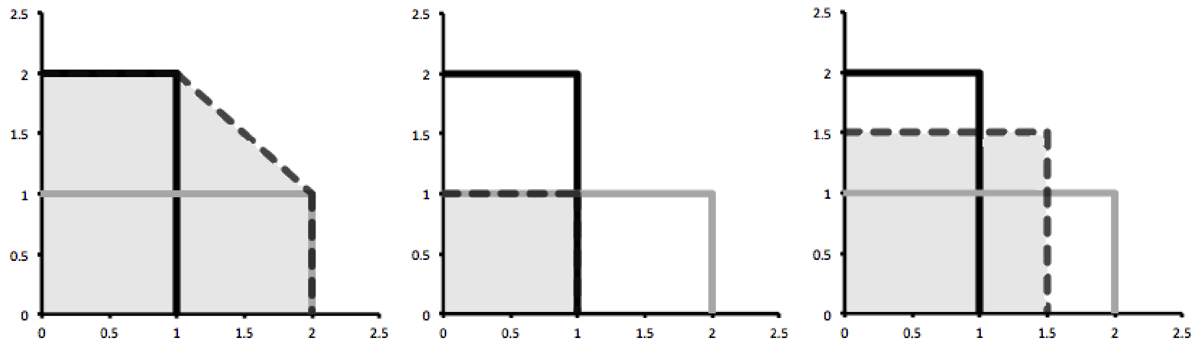
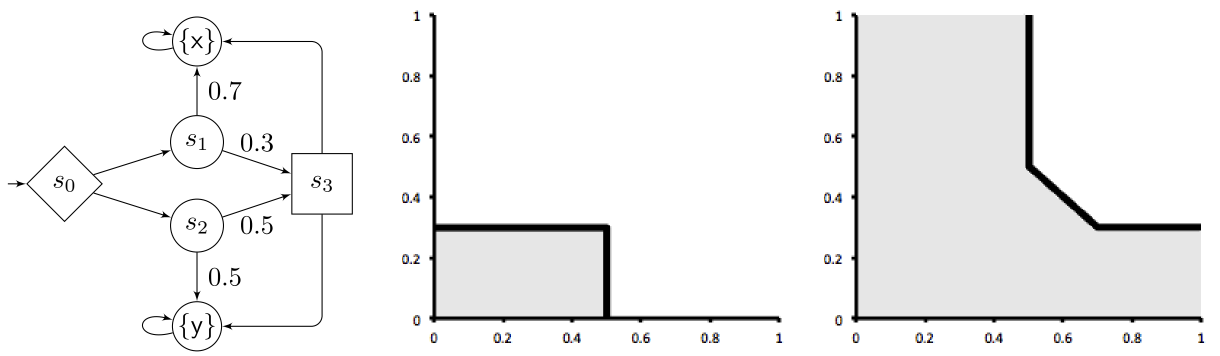

(a) $X_s = \mathsf{dwc}(\mathsf{conv}(X_t \cup X_u))$. (b) $X_s = \mathsf{dwc}(X_t \cap X_u)$. (c) $X_s = \mathsf{dwc}(\frac{1}{2}\cdot X_t + \frac{1}{2}\cdot X_u)$.

Figure 5.2: Operations of the functional for a state s with two successors, t and u. X_t is the downward closure of the black line and X_u is the downward closure of the grey line. Set X_s, is a downward closure of the dashed line (area shaded in grey).

Next, we present the algorithm for computing Pareto set approximations for disjunctive rPATL. The algorithm uses the relationship between the disjunctive rPATL objectives and the single-objective expected total reward games established in Theorem 7 (see Remark 4). It makes repeated calls to the algorithm computing optimal values for such games to compute the approximation of the Pareto set for a disjunctive rPATL formula. The optimal value of the single-objective expected total reward in the coalition game is $\mathbb{E}_s^{\max,\min}[rew(\vec{r},\emptyset)]$, computation of which was discussed in Section 4.2.3.

Theorem 9 (Pareto sets: disjunctive rPATL) *Given a stopping SMG $\mathcal{G}$ and a multi-objective rPATL formula, $\varphi(\vec{r},\vec{v}) = \langle\langle C\rangle\rangle(\bigvee_{j=1}^{m} \mathsf{R}_{\geq v_j}^{r_j}[\mathsf{F}\perp])$ an ε-approximation of the Pareto set is the set of Pareto points of $U_X \overset{\text{def}}{=} \bigcup_{\vec{x}\in X}\{\vec{p} \mid \vec{x}\cdot\vec{p} \leq d_{\vec{x}}\}$, where X is a set of all non-zero vectors $\vec{x}$ such that $\|\vec{x}\| = 1$, and where all x_i are of the form $\tau \cdot k_i$ for some $k_i \in \mathbb{N}$ and $\tau = \frac{\varepsilon}{2\cdot m^2\cdot(M+1)}$; and $d_{\vec{x}}$ is the optimal value for expected total reward for the reward function $\vec{x}\cdot\vec{r}$ in the coalition game $\mathcal{G}_C$, i.e., $\mathbb{E}_s^{\max,\min}[\vec{x}\cdot rew(\vec{r},\emptyset)]$.*

Proof. By Theorem 7 we have that if $\mathbb{E}_s^{\max,\min}[\vec{x}\cdot rew(\vec{r},\emptyset)] \geq \vec{x}\cdot\vec{v}$, then $\varphi(\vec{r},\vec{v})$ is achievable. Hence, it is not difficult to see that U_X yields an under-approximation of achievable points for $\varphi(\vec{r},\vec{v})$. We show that, for any point in the Pareto set, there is an ε-close point in U_X. Consider any point $\vec{p}$ in the Pareto set for $\varphi(\vec{r},\vec{v})$ and let $\sigma_\Box$ be a strategy in a coalition game $\mathcal{G}_C$, which achieves this point. Note that, for some $\vec{y} \in \mathbb{R}_{\geq 0}^m$ such that $\|\vec{y}\| = 1$, we have $\vec{p}\cdot\vec{y} = d_{\vec{y}}$, since otherwise $\vec{p}$ would not be a Pareto point. Let $\vec{x} = \operatorname{argmin}_{\vec{z}\in X}\|\vec{z}-\vec{y}\|$ be a vector in X, which is closest to $\vec{y}$. Note that $d_{\vec{y}}-d_{\vec{x}} \leq m\cdot M\cdot\tau$ and thus $d_{\vec{x}} \geq d_{\vec{y}}-m\cdot M\cdot\tau$.



(a) SMG. (b) $\langle\langle\{\Box\}\rangle\rangle(\mathsf{P}_{\geq v_1}[\mathsf{F}\,\mathsf{x}] \wedge \mathsf{P}_{\geq v_2}[\mathsf{F}\,\mathsf{y}])$ (c) $\langle\langle\{\Box\}\rangle\rangle(\mathsf{P}_{\geq v_1}[\mathsf{F}\,\mathsf{x}] \vee \mathsf{P}_{\geq v_2}[\mathsf{F}\,\mathsf{y}])$

Figure 5.3: Sets of achievable vectors for the queries $\varphi(\vec{v})$ in state s_0 of the SMG (player $\Box$ has a strategy to achieve any point in the area shaded in grey). Horizontal axis represents values for v_1 and vertical one represents the values for v_2.

For the point $\vec{q} = \vec{p} - \frac{\varepsilon}{m}$, we have $\vec{q} \cdot \vec{x} \leq d_{\vec{x}}$ because

$$\begin{aligned}\vec{q} \cdot \vec{x} &= \vec{p} \cdot \vec{x} - \frac{\vec{\varepsilon}}{m} \cdot \vec{x} \leq \vec{p} \cdot \vec{y} + \vec{p} \cdot \vec{\tau} - (\frac{\vec{\varepsilon}}{m} \cdot \vec{y} - \frac{\vec{\varepsilon}}{m} \cdot \vec{\tau}) \leq d_{\vec{y}} + \vec{M} \cdot \vec{\tau} - \frac{\varepsilon}{m} + m \cdot \tau \\ &\leq d_{\vec{y}} + m \cdot (M+1) \cdot \tau - \frac{\varepsilon}{m} \leq d_{\vec{y}} - m \cdot M \cdot \tau \leq d_{\vec{x}},\end{aligned}$$

and so $\vec{q} \in U_X$. Since $\|\vec{p} - \vec{q}\| \leq \varepsilon$, this concludes the proof. The other direction can be proved similarly. $\Box$

The examples of Pareto sets for conjunctive and disjunctive queries are shown in Figure 5.3. We consider the game from Figure 5.3a starting in state s_0. The conjunctive query is $\varphi(\vec{v}) = \langle\langle\{\Box\}\rangle\rangle(\mathsf{P}_{\geq v_1}[\mathsf{F}\,\mathsf{x}] \wedge \mathsf{P}_{\geq v_2}[\mathsf{F}\,\mathsf{y}])$ and the disjunctive query is $\varphi(\vec{v}) = \langle\langle\{\Box\}\rangle\rangle(\mathsf{P}_{\geq v_1}[\mathsf{F}\,\mathsf{x}] \vee \mathsf{P}_{\geq v_2}[\mathsf{F}\,\mathsf{y}])$, results for which are shown in Figure 5.3b and Figure 5.3c, respectively.

Finally, using Theorem 7 we can combine the methods described above from Theorem 8 and Theorem 9 to provide an algorithm to compute an $\varepsilon-$approximation of the Pareto set for any multi-objective rPATL formula. Given a stochastic multi-player game $\mathcal{G}$, a multi-objective rPATL formula $\varphi(\vec{q}, \vec{u}) = \langle\langle C\rangle\rangle(\bigwedge_{i=1}^{n} \bigvee_{j=1}^{m} \mathsf{R}^{q_{i,j}}_{\geq u_{i,j}}[\mathsf{F}\perp])$ and $\varepsilon > 0$ the following procedure computes $\varepsilon-$approximation of the Pareto set for $\varphi(\vec{q}, \vec{u})$.

1. Construct sets $X_1, X_2, \ldots, X_n$ for each conjunct ($1 \leq i \leq n$) containing unit vectors as in Theorem 9, such that each set contains the number of vectors that suffice for $\frac{\varepsilon}{2}-$approximation for the disjunction of objectives.

2. For each $\vec{x} = (\vec{x}_1, \ldots, \vec{x}_n) \in \prod_{i=1}^{n} X_i$ construct a vector of reward functions $\vec{r} = (\vec{x_1} \cdot \vec{q_1}, \ldots, \vec{x_n} \cdot \vec{q_n})$ as in Theorem 7 and compute the $\frac{\varepsilon}{2}-$approximation of the Pareto set for $\vartheta(\vec{r}, \vec{v}) = \langle\langle C\rangle\rangle \bigwedge_{i=1}^{n} \mathsf{R}^{r_i}_{\geq v_i}[\,\mathsf{F}\perp\,]$, $P_{\vec{x}}$, using algorithm from Theorem 8.

3. Similarly to Theorem 9, for each $\vec{x}$ and its Pareto set $P_{\vec{x}}$ we compute the set of points that are achievable for $\varphi(\vec{q}, \vec{u})$ with $\vec{x}$ as $U_{\vec{x}} = \bigcup_{y \in P_{\vec{x}}} \prod_{i=1}^{n} \{p \mid \vec{x_i} \cdot p \le y_i\}$.

4. The $\varepsilon-$approximation of the Pareto set Q for φ is then the Pareto set of $\bigcup_{\vec{x} \in \prod_{i=1}^{n} X_i} U_{\vec{x}}$.

We have the following:

- Take a vector $\vec{u}$ from the Pareto set of $\varphi(\vec{q}, \vec{u})$ and a strategy $\sigma_\Box$ which achieves it in the coalition game. Then, by Theorem 7, this strategy also achieves vector $\vec{v} = (\vec{x_1} \cdot \vec{u_1} - \frac{\varepsilon}{2}, \ldots, \vec{x_n} \cdot \vec{u_n} - \frac{\varepsilon}{2})$ for reward function vector $\vec{r} = (\vec{x_1} \cdot \vec{q_1}, \ldots, \vec{x_n} \cdot \vec{q_n})$ for some $\vec{x}$. By construction of sets X_i in step 1 we have that there exists $\vec{x}' \in \prod_{i=1}^{n} X_i$ which differs from $\vec{x}$ by at most $\frac{\varepsilon}{2}$, and in step 2 we get all achievable points, that are at most another $\frac{\varepsilon}{2}$ away from optimal ones, hence the points obtained in step 3 are at most ε away from the optimal and so, $\vec{u} - \varepsilon \in Q$.

- For any vector $\vec{u} \in Q$, there exists a strategy $\sigma_\Box$ in a coalition game that achieves at least $\vec{u} - \varepsilon$, that is, the Pareto set of $\varphi(\vec{q}, \vec{u})$ has to contain vector which is greater or equal to $\vec{u} - \varepsilon$. But also, by the above construction, we have that any vector $\vec{u} + \delta$ for $\delta > \varepsilon$ is not achievable, and hence the Pareto set of $\varphi(\vec{q}, \vec{u})$ must contain a vector $\vec{v}$ such that $\vec{u} - \varepsilon \le \vec{v} \le \vec{u} + \varepsilon$.

Discussion

In this section we have shown how to reduce the model checking of multi-objective rPATL to model checking the formula of the form $\langle\langle C \rangle\rangle \bigwedge_{i=1}^{n} \bigvee_{j=1}^{m} \mathsf{R}^{q_{i,j}}_{\triangleright u_{i,j}} [\, \mathsf{F}^c \perp]$, where $\triangleright \in \{>, \geq\}$, and presented algorithms to approximate the Pareto sets for such a query up to a specified precision for a class of stopping games. The approximation algorithms that we have developed here are based on value iteration (for conjunctive rPATL) and solution of single-objective games (for disjunctive rPATL), which allows us to compute successive approximations of the solutions. For example, in [42] the prototype implementation of the algorithm from Theorem 8 (dealing with positive expected rewards only) has been developed and was applied to models having up to 100,000 states.

Note that the result of Theorem 7 applies to general (i.e., non-stopping) games and also equations from Theorem 8 can be used to compute successive approximations of the Pareto sets for such games, but the provision of convergence criteria for such approximation remains an open problem. The computation of *exact* Pareto sets for multi-objective rPATL is an open problem for both stopping and non-stopping games. However, in Theorem 13 in the next section, we use the result Theorem 7 to prove the decidability for disjunctive rPATL.

5.3 Complexity

In this section we turn our attention to the analysis of the complexity of multi-objective rPATL model checking. All results that we present here are for terminal state reachability or for the expected total reward properties, and thus are among the simplest of multi-objective rPATL properties.

The section is structured as follows. In Section 5.3.1 we show that, in contrast with rPATL, games with multi-objective rPATL objectives are not determined already for two objectives and optimal strategies may not exist to achieve a point in the Pareto set. Then, in Section 5.3.2, we discuss the memory requirements for strategies of coalitions showing that, in general, infinite memory may be required, but if the formulae are restricted to disjunctions of expected total reward objectives, then memoryless deterministic strategies suffice. Finally, in Section 5.3.3 we prove several complexity lower bounds showing that multi-objective rPATL model checking is PSPACE-hard in general, and undecidable if player strategies are not allowed to use randomisation. For the case where formulae are restricted to disjunctions of expected total reward objectives, we show that the problem is NP-complete.

The results of nondeterminacy and non-existence of optimal strategies from Section 5.3.1 are have been published in [39]; results on memory requirements and complexity lower bounds from Sections 5.3.2 and 5.3.3 have been published in [40].

5.3.1 Determinacy and optimal strategies

We begin by showing that games with multi-objective rPATL goals are not determined.

Theorem 10 *Stochastic multi-player games with multi-objective rPATL goals are not determined.*

Proof. Consider the game $\mathcal{G} = \langle\{\Box, \Diamond\}, S, (S_{\bigcirc}, S_{\Box}, S_{\Diamond}), \Delta, AP, \chi\rangle$ from Figure 5.4. To show nondeterminacy we first prove that player $\Box$ does not have a strategy $\sigma_{\Box}$ to satisfy the multi-objective rPATL formula $\langle\langle\{\Box\}\rangle\rangle(\mathsf{P}_{\geq 0.5}[\mathsf{F}\,\mathsf{t}] \wedge \mathsf{P}_{\leq 0.5}[\mathsf{F}\,\mathsf{t}])$ in a state s_0, i.e., $\forall\sigma_{\Box} \in \Sigma_{\Box} . \exists\sigma_{\Diamond} \in \Sigma_{\Diamond} . (\Pr_{s_0}(\mathsf{F}\,\mathsf{t}) < 0.5 \vee \Pr_{s_0}(\mathsf{F}\,\mathsf{t}) > 0.5)$. To see this, consider player $\Diamond$'s strategy $\sigma_{\Diamond}$, which, if player $\Box$'s strategy assigns positive probability to go to s_3, $\sigma_{\Diamond}$ chooses to go to s_3, and to s_4 otherwise. In the first case, $\sigma_{\Diamond}$ ensures that the probability to reach s_4 is less than 0.5 (and so $\Pr_{s_0}(\mathsf{F}\,\mathsf{t}) < 0.5$), and in the second case it ensures that the probability to go to s_4 is greater than 0.5 (and so $\Pr_{s_0}(\mathsf{F}\,\mathsf{t}) > 0.5$).

On the other hand, for any player $\Diamond$ strategy in this game, there exists a player $\Box$ strategy which ensures that the formula holds, i.e., $\forall\sigma_{\Diamond} \in \Sigma_{\Diamond} . \exists\sigma_{\Box} \in \Sigma_{\Box} . (\Pr_{s_0}(\mathsf{F}\,\mathsf{t}) \geq$

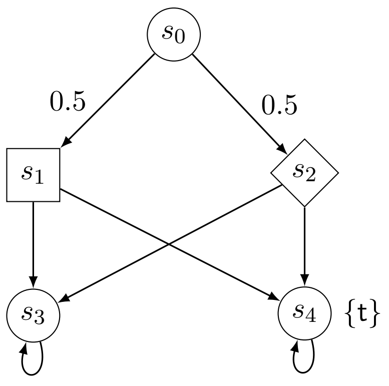


Figure 5.4: Game demonstrating nondeterminacy.

$0.5 \wedge \Pr_{s_0}(\mathsf{F}\,\mathsf{t}) \leq 0.5)$. Consider player $\lozenge$ strategy, which assigns probability p to choose s_3 as a successor. Define a player $\Box$ strategy $\sigma_\Box$ such that is assigns probability $1-p$ to choose s_3, making sure that it is reached with probability exactly 0.5, and hence probability to reach s_4 is also 0.5. Thus $\sigma_\Box$ is a winning strategy. □

Next, we show that optimal strategies may not exist for a point in the Pareto set of the multi-objective rPATL query.

Theorem 11 *In stochastic multi-player games optimal strategies may not exist for coalition to achieve a point in the Pareto set for a multi-objective rPATL formula.*

Proof. Consider the state s_0 of the game $\mathcal{G} = \langle \{\Box, \lozenge\}, S, (S_\bigcirc, S_\Box, S_\lozenge), \Delta, AP, \chi \rangle$ from Figure 5.5. The multi-objective rPATL formula that we consider is $\varphi(\vec{v}) = \langle\langle \{\Box\} \rangle\rangle (\mathsf{P}_{\geq v_1}[\mathsf{F}\,\mathsf{t}] \wedge \mathsf{P}_{\leq v_2}[\mathsf{F}\,\mathsf{t}])$. First, we claim that there does not exist player $\Box$ strategy to satisfy this for-

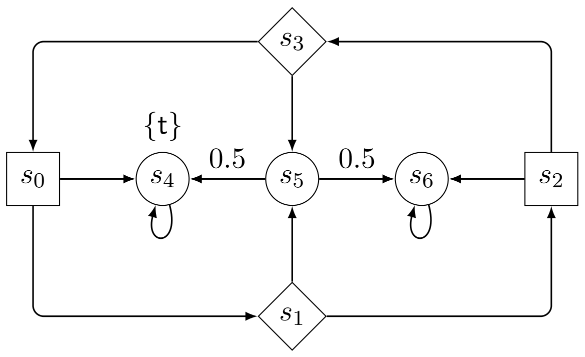


Figure 5.5: Game demonstrating non-existence of optimal strategies.

mula for $\vec{v} = (0.5, 0.5)$ in s_0. To see this, consider any player $\Box$ strategy $\sigma_\Box$ and define the player $\lozenge$ strategy $\sigma_\lozenge$ as follows. If $\sigma_\Box$ assigns positive probability to go to s_4 in s_0, then $\sigma_\lozenge$ goes to s_5 in s_1 and ensures that probability to reach s_4 is greater than 0.5 (and so

$\mathrm{Pr}_{s_0}(\mathsf{F}\,\mathsf{t}) > 0.5$). Otherwise, $\sigma_\Diamond$ chooses s_2 as successor in s_1. Now, if $\sigma_\Box$ assigns positive probability to go to s_6 in s_2, then $\sigma_\Diamond$ picks s_5 as successor ensuring that probability to reach s_4 is less than 0.5 (and so $\mathrm{Pr}_{s_0}(\mathsf{F}\,\mathsf{t}) < 0.5$). If $\sigma_\Box$ never picks states s_4 and s_6 as successors with positive probability, then the probability to reach s_4 is 0 (and hence $\mathrm{Pr}_{s_0}(\mathsf{F}\,\mathsf{t}) < 0.5$). So, player $\Diamond$ strategy $\sigma_\Diamond$ as defined above, is a winning strategy for player $\Diamond$ for any player $\Box$ strategy.

We have shown that $\varphi(\vec{v})$ is not achievable. Now to show that the point $\vec{v}$ is indeed in the Pareto set, we show that for any $\varepsilon > 0$, $\varphi(\vec{v} - \mathsf{dir}(\vec{v}) \cdot \varepsilon)$ is achievable (i.e., formula $\langle\langle\{\Box\}\rangle\rangle(\mathsf{P}_{\geq 0.5-\varepsilon}[\mathsf{F}\,\mathsf{t}] \wedge \mathsf{P}_{\leq 0.5+\varepsilon}[\mathsf{F}\,\mathsf{t}])$ holds for any ε). To see this consider player $\Box$ strategy $\sigma_\Box$, which in s_0 picks s_4 as successor with probability ε, and, in s_2, $\sigma_\Box$ picks s_6 with probability $\frac{\varepsilon}{1-\varepsilon}$; this ensures that for all player $\Diamond$ strategies $\sigma_\Diamond$ we have $0.5-\varepsilon \leq \mathrm{Pr}_{s_0}(\mathsf{F}\,\mathsf{t}) \leq 0.5+\varepsilon$, and hence the formula is satisfied. $\Box$

Discussion. The results proved in this section provide several important implications for the model checking of multi-objective games. Firstly, the nondeterminacy and non-existence of optimal strategies already for terminal state reachability objectives are in sharp contrast with single-objective games, which are determined and optimal strategies exist. This means that we cannot leverage the determinacy in the way it is done for single-objective games, e.g., it no longer suffices to consider only algorithms for maximisation (i.e., we cannot utilise equations similar to the ones use for rPATL in Equations (4.1)) and hence we need to deal with minimisation objectives explicitly. Secondly, non-existence of optimal strategies means that, in some cases, approximation algorithms (e.g., like the ones presented in Section 5.2) are the best that we can do.

5.3.2 Memory requirements

In this section, we prove that infinite memory is required for a coalition to achieve a conjunctive rPATL formula and that memoryless strategies are sufficient for disjunctive rPATL with expected total reward objectives.

Theorem 12 (Conjunctive rPATL) *Infinite memory may be required for a coalition to win a stochastic multi-player game with a conjunctive rPATL objective.*

Proof. We provide a sketch of the proof here and the full proof is presented in Appendix C. To prove the theorem we use the game $\mathcal{G} = \langle\{\Box, \Diamond\}, S, (S_\bigcirc, S_\Box, S_\Diamond), \Delta, AP, \chi\rangle$ from Figure 5.6. We start by explaining the intuition behind the need of infinite memory before presenting the formal proof. First, we note that it is sufficient to consider deterministic counter-strategies for player $\Diamond$, since, after player $\Box$ has proposed his strategy the resulting model is an MDP with finite branching [87].

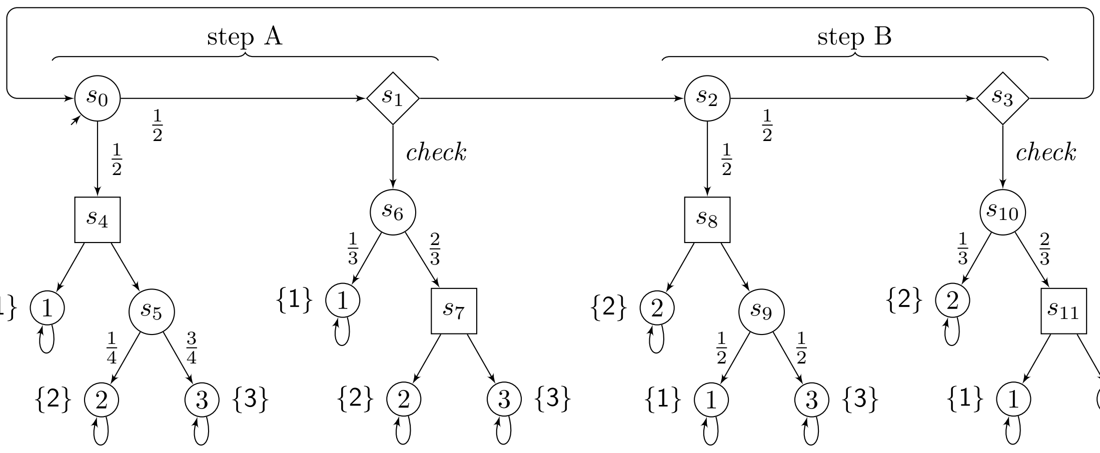


Figure 5.6: SMG in which infinite memory is required to by player $\Box$ to satisfy multi-objective rPATL formula $\langle\langle\{\Box\}\rangle\rangle(\mathsf{P}_{\geq\frac{1}{3}}[\mathsf{F}\,1] \wedge \mathsf{P}_{\geq\frac{1}{3}}[\mathsf{F}\,2] \wedge \mathsf{P}_{\geq\frac{1}{3}}[\mathsf{F}\,3])$ in state s_0.

Consider the game starting in the initial state s_0 and a multi-objective rPATL formula $\langle\langle\{\Box\}\rangle\rangle(\mathsf{P}_{\geq\frac{1}{3}}[\mathsf{F}\,1] \wedge \mathsf{P}_{\geq\frac{1}{3}}[\mathsf{F}\,2] \wedge \mathsf{P}_{\geq\frac{1}{3}}[\mathsf{F}\,3])$. We note that all target states (1, 2 and 3) are terminal and disjoint, and hence for any $\sigma_\Box$ and $\sigma_\Diamond$ we have that $\sum_{\mathsf{i}\in\{1,2,3\}} \mathrm{Pr}_{s_0}(\mathsf{F}\,\mathsf{i}) = 1$, and therefore, for any winning player $\Box$ strategy $\sigma_\Box$, it must be the case that, for any $\sigma_\Diamond$, $\mathrm{Pr}_{s_0}(\mathsf{F}\,\mathsf{i}) = \frac{1}{3}$ for $\mathsf{i} \in \{1,2,3\}$. Let $E \subseteq \Omega_{\mathcal{G},s_0}$ be the set of paths, which never take any transition '*check*' (i.e., do not contain states s_6 and s_{10}). The game proceeds by alternating between the two steps, A and B, as indicated in Figure 5.6. In step A, player $\Box$ chooses a probability to go to 1 from state s_4, and then player $\Diamond$ gets an opportunity to "verify" that the probability $\mathrm{Pr}_{s_0}(\mathsf{F}\,1|E)$ of paths reaching 1 conditional on the event that no '*check*' action was taken is $\frac{1}{3}$. He can do this by taking the action '*check*' and so ensuring that $\mathrm{Pr}_{s_0}(\mathsf{F}\,1|\Omega_{\mathcal{G},s_0} \setminus E) = \frac{1}{3}$. If player $\Diamond$ again does not choose to take '*check*', the game continues in step B, where the same happens for 2, and so on.

When first performing step A, player $\Box$ has to pick probability $\frac{1}{3}$ to go to 1. But since the probability of going from s_4 to 2 is $< \frac{1}{3}$, when step B is performed for the first time, player $\Box$ must go to 2 with probability $y_0 > \frac{1}{3}$ to compensate for the "loss" of the probability in step A. However, this decreases the probability of reaching 1 at step B, and so player $\Box$ must compensate for it in the subsequent step A by taking probability $> \frac{1}{3}$ of going to 1. This decreases the probability of reaching 2 in the second step B even more (compared to first execution of step A), for which player $\Box$ must compensate by picking $y_1 > y_0 > \frac{1}{3}$ in the second execution of step B, and so on. So, in order to win, player $\Box$ has to play infinitely many different probability distributions in states s_4 and s_8. Note that, if player $\Diamond$ takes action '*check*', player $\Box$ can always randomise appropriately in states s_7 and s_{11} to achieve probabilities exactly $\frac{1}{3}$ for all objectives. $\Box$

We already mentioned in Remark 4 following the proof of Theorem 7, that if a coalition strategy achieves disjunctive rPATL formula $\langle\langle C\rangle\rangle \bigvee_{j=1}^{m} \mathsf{R}_{\geq v_j}^{r_j}[\,\mathsf{F}^c \perp]$, then for every $\varepsilon > 0$ there exists a non-zero vector $\vec{x} \in \mathbb{R}_{\geq 0}^{m}$, such that this strategy achieves single-objective rPATL formula $\langle\langle C\rangle\rangle\, \mathsf{R}_{\geq v}^{r}[\,\mathsf{F}^c \perp]$, where $r(s) = \vec{x} \cdot \vec{q}(s)$ and $v = \vec{x} \cdot (\vec{u} - \varepsilon)$. We now use this relationship to prove that memoryless deterministic strategies are sufficient to achieve such disjunctive rPATL objectives.

Theorem 13 (Disjunctive rPATL) *Memoryless deterministic strategies are sufficient for the coalition to achieve a disjunctive rPATL formula* $\varphi(\vec{r}, \vec{v}) = \langle\langle C\rangle\rangle \bigvee_{j=1}^{m} \mathsf{R}_{\geq v_j}^{r_j}[\,\mathsf{F}^c \perp]$.

Proof. Assume that there exists a strategy for player $\Box$ in coalition game $\mathcal{G}_C$ achieving $\varphi(\vec{r}, \vec{v})$. Then by Theorem 7 we know that for all $\varepsilon > 0$ there exists a winning strategy which achieves the single-objective $\varphi_\varepsilon = \forall \sigma_\Diamond \in \Sigma_\Diamond \,.\, \mathbb{E}_s[\vec{x}_\varepsilon \cdot rew(\vec{r}, \emptyset)] \geq \vec{x}_\varepsilon \cdot (\vec{v} - \varepsilon)$ for some $\vec{x}_\varepsilon \in \mathbb{R}^m$. We can assume such strategy is memoryless deterministic (MD), because such strategies suffice to achieve a single-objective expected total reward in stochastic games [59]. Define a set $\Gamma = \{k^{-1} \mid k \in \mathbb{N}\}$. We know that for every $\varepsilon \in \Gamma$ there exists an MD strategy achieving φ_ε. Because the number of MD strategies if finite, there must exists some $\sigma_\Box$, which is MD and winning for infinitely many $\varepsilon \in \Gamma$. We prove that this $\sigma_\Box$ actually achieves φ_ε for all $\varepsilon > 0$. Assume for a contradiction that there is some $\delta > 0$ such that

$$\forall \vec{x}_\delta \in \mathbb{R}_{\geq 0}^{m} \,.\, \exists \sigma_\Diamond \in \Sigma_\Diamond \,.\, \mathbb{E}_s[\vec{x}_\delta \cdot rew(\vec{r})] < \vec{x}_\delta \cdot (\vec{v} - \delta). \tag{5.1}$$

Pick $\varepsilon \in \Gamma$ such that $\varepsilon < \delta$ and pick $\vec{x}_\varepsilon \in \mathbb{R}_{\geq 0}^{m}$ such that $\forall \sigma_\Diamond \in \Sigma_\Diamond \,.\, \mathbb{E}_s[\vec{x}_\varepsilon \cdot rew(\vec{r}, \emptyset)] \geq \vec{x}_\varepsilon \cdot (\vec{v} - \varepsilon)$. But we have that $(\vec{v} - \varepsilon) > (\vec{v} - \delta)$, and hence $\forall \sigma_\Diamond \in \Sigma_\Diamond \,.\, \mathbb{E}_s[\vec{x}_\varepsilon \cdot rew(\vec{r}, \emptyset)] > \vec{x}_\varepsilon \cdot (\vec{v} - \delta)$, which contradicts (5.1). $\Box$

Discussion. The result of Theorem 12 for infinite memory for conjunctive rPATL proved in this section is important in several ways. First, it contrasts multi-objective games with other models. For example, memoryless deterministic strategies are sufficient for most single-objective games including reachability, expected total reward and parity objectives (see Theorem 4 in Section 3.2.4). Also, memoryless deterministic strategies are sufficient for controller strategy to achieve multi-objective reachability and expected total reward objectives in MDPs (see Theorem 2 in Section 3.2.3), so it shows that adding another source of nondeterminism makes the problem more difficult. Note that the proof showed that player $\Box$ might need to use infinitely many probability distributions, which, for the deterministic-update strategy model, implies that infinite memory is required. It does not rule out the case, that finite memory stochastic-update strategy can be constructed, which can generate infinitely many probability distributions using finitely many memory elements (e.g., in Section 3.4 we have shown how such strategy can generate exponentially many probability distributions with linear number of memory elements).

The result of existence of memoryless deterministic strategies for disjunctive rPATL objectives proved in Theorem 13 effectively provides an NP algorithm for model checking such formulae: one can guess such a strategy and then check that the multi-objective formula holds for any strategy in the resulting MDP; this check can be done in polynomial time (see Section 3.2.3 and Theorem 2 for discussion).

5.3.3 Complexity bounds

In this section we discuss the complexity bounds of model checking multi-objective rPATL. We start by showing that the model checking problem of multi-objective rPATL is PSPACE-hard in general and undecidable, if coalition is restricted to deterministic strategies only. We finish by proving that the model checking problem for disjunctive rPATL with expected total reward goals is NP-complete.

Theorem 14 (PSPACE-hardness) *The problem of deciding whether a multi-objective rPATL formula is satisfied in a state of an SMG is* PSPACE-*hard.*

Proof. We prove the PSPACE-hardness of the problem of deciding the existence of a winning coalition in an SMG to achieve a given multi-objective rPATL formula by reduction from satisfiability of quantified boolean formula (QBF), which is known to be PSPACE-complete [92]. Consider QBF with n variables and m clauses

$$\psi = \exists x_1 \forall x_2 \exists x_3 \ldots \forall x_n \, . \, c_1 \wedge c_2 \wedge \cdots \wedge c_m,$$

where each $c_i = (l_1^i \vee l_2^i \vee l_3^i)$ and $l_j^i \in \{x_1, \neg x_1, \ldots, x_n, \neg x_n\}$. We assume that every clause contains at most one literal for any given variable. For the reduction we use the game $\mathcal{G} = \langle \{\Box, \Diamond\}, S, (S_{\bigcirc}, S_{\Box}, S_{\Diamond}), \Delta, AP, \chi \rangle$ from Figure 5.7, where the shape of the state indicates, which player controls it. A state x_j^+ is labelled with $\mathsf{C_i}$ (i.e., $\mathsf{C_i} \in \chi(x_j^+)$) if clause c_i of ψ contains literal x_j, and same for the state x_j^- if c_i contains literal $\neg x_j$, for all j. Other labellings are as follows: $\chi(x_j) = \emptyset$, $\chi(p_j) = \{\mathsf{p_j}\}$, and $\chi(n_j) = \{\mathsf{n_j}\}$ for all $1 \leq j \leq n$.

Consider the following multi-objective rPATL formula in a state x_1:

$$\varphi = \langle\!\langle \{\Box\} \rangle\!\rangle \, (\bigwedge_{\mathsf{i}=1}^{\mathsf{m}} \mathsf{P}_{\geq \frac{1}{2^{2 \cdot n}}} [\mathsf{F} \, \mathsf{C_i}] \wedge \tag{5.2}$$

$$\bigwedge_{\mathsf{i}=\{1,3,\ldots,\mathsf{n}-1\}} (\mathsf{P}_{\leq 0}[\mathsf{F} \, \mathsf{p_i}] \vee \mathsf{P}_{\leq 0}[\mathsf{F} \, \mathsf{n_i}]) \,) \, . \tag{5.3}$$

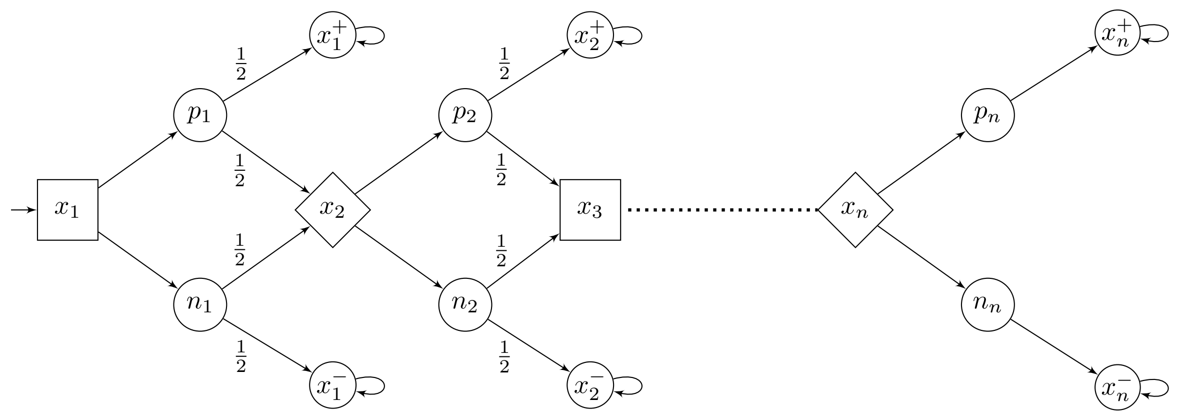


Figure 5.7: Game illustrating PSPACE-hardness.

First observe that, in order to win the game, player $\Box$ has to use a deterministic strategy. This is ensured by the conjunction in (5.3), which makes sure that if player $\Box$ has a winning strategy in state x_1, then this strategy has to pick either p_i or n_i in a state x_i for all i. We show that ψ is true if and only if player $\Box$ has a winning strategy for φ.

For the "$\Rightarrow$" direction, let us assume there are functions $q_i : \mathbb{B}^{i-1} \to \mathbb{B}$ for $i \in \{1, 3, \dots n-1\}$ such that for any $v_2, v_4 \dots, v_n \in \mathbb{B}$ the formula $c_1 \wedge \dots \wedge c_m$ is satisfied under the assignment ν defined inductively by

$$\nu(x_i) = \begin{cases} q_i(\nu(x_1), \dots, \nu(x_{i-1})) & \text{if } i \in \{1, 3, \dots, n-1\} \\ v_i & \text{if } i \in \{2, 4, \dots, n\} \end{cases}$$

This can be directly transformed into a player $\Box$ strategy in the game from Figure 5.7 where $\star_i \in \{p_i, n_i\}$, $b(p_i) = 1$ and $b(n_i) = 0$:

$$\sigma_\Box(x_1 \star_1 \dots x_{i-1} \star_{i-1}) = \begin{cases} p_i & \text{if } q_i(b(\star_1), \dots, b(\star_{i-1})) = 1 \\ n_i & \text{otherwise.} \end{cases}$$

Let $\sigma_\Diamond$ be an arbitrary strategy for player $\Diamond$. Let us consider a path $x_1 \star_1 \dots x_n \star_n$ such that, for every $i \in \{1, 3, \dots n-1\}$ we have $\sigma_\Box(x_1 \star_1 \dots x_i) = \star_i$, and for every $i \in \{2, 4, \dots n\}$ we have $\sigma_\Diamond(x_1 \star_1 \dots x_i)(\star_i) \geq 0.5$. Note that such a path always exsits since player $\Diamond$ has exactly two choices in every state it controls. By the properties of the functions q_i and by the construction of $\sigma_\Box$ we have that the valuation μ which to x_i assigns $b(\star_i)$ satisfies every $c_1 \wedge \dots \wedge c_m$. Fix c_j for $1 \leq j \leq m$, there must be a literal which makes c_j satisfied under μ, let x_k be such a variable. By the definition of the game we have that the state x_k^+ (resp. x_k^-) is labelled by $\mathsf{C_j}$ if c_j contains x_k (resp. $\neg x_k$). Thus, a state labelled with

$\mathsf{C_j}$ is reached at least with probability

$$\Big(\prod_{i=1}^{k}\frac{1}{2}\Big)\cdot\Big(\prod_{i\in\{2,4,\ldots e(k)\}}\sigma_\Diamond(x_1\star_1\ldots x_i)(\star_i)\Big) \quad\geq\quad \frac{1}{2^{2\cdot k}}\geq\quad\frac{1}{2^{2\cdot n}}$$

which is the probability of the path $x_1\star_1\ldots x_kp_kx_k^+$ (resp. $x_1\star_1\ldots x_kn_kx_k^-$), where $e(k)=k$ if k is even and $e(k)=k-1$ if k is odd.

The other direction "$\Leftarrow$" can be proved by directly constructing assignment functions from the winning strategy $\sigma_\Box$ for player $\Box$. This can be achieved because the winning strategy must be deterministic, as discussed above. $\Box$

We now show that if one restricts a coalition to use deterministic strategies in the game, multi-objective rPATL model checking problem becomes undecidable. We build on the idea presented in the proof for infinite memory (see Theorem 12), to construct gadgets, which encode counter values of a two-counter machine as probabilities that need to be played by the coalition in order to achieve a multi-objective rPATL goal. The proof idea is inspired by the undecidability of existence of a winning strategy in Markov decision processes with objectives expressed in a variant of the branching-time logic PCTL [19].

Theorem 15 (Undecidability) *The problem of whether a multi-objective rPATL formula is satisfied in a state of an SMG, where coalition strategies are restricted to be deterministic, is undecidable.*

Proof. We present the high level idea for a proof here; the full details can be found in Appendix D. We show the undecidability of the problem via a reduction to the termination problem of a two-counter machine. We show that a two-counter machine $\mathcal{M}$ does not terminate if and only if multi-objective rPATL formula is satisfied in the game $\mathcal{G}(\mathcal{M})$ constructed using the reduction presented here.

A two-counter machine $\mathcal{M}$ consists of a sequence of instructions $l_1 : ins_1, \cdots, l_n : ins_n$, where each ins_i has one of the following forms:

1. $c_1 := c_2 := 0$ and goto l_j;
2. $c_1 = c_1 + 1$ and goto l_j;
3. $c_2 = c_2 + 1$ and goto l_j;
4. if $c_1 = 0$ then goto l_j else $c_1 = c_1 - 1$ and goto l_k;
5. if $c_2 = 0$ then goto l_j else $c_2 = c_2 - 1$ and goto l_k;
6. Terminate.

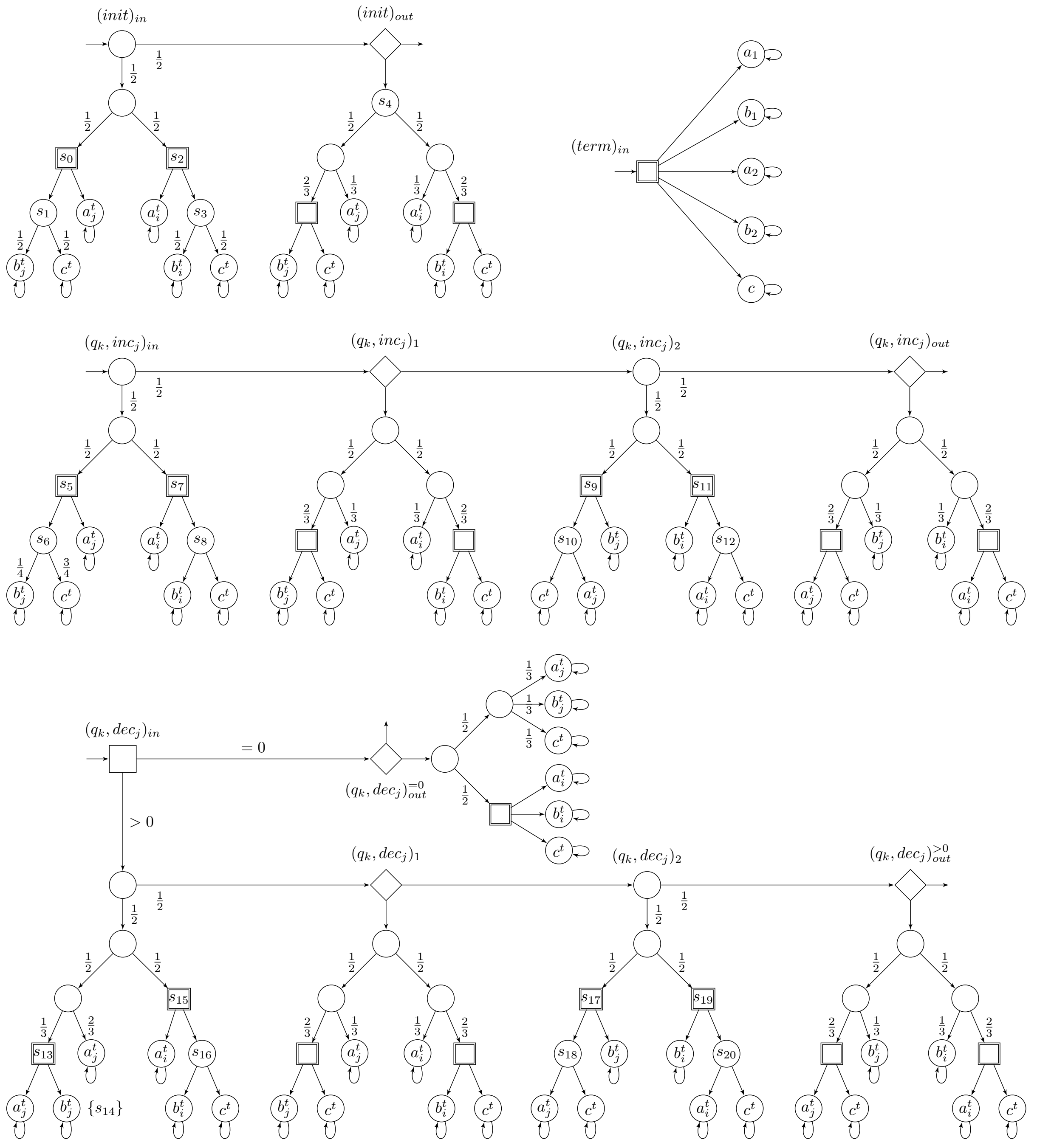
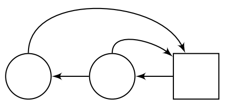


Figure 5.8: Operations for counter j. In the order left to right and top to bottom we have gadgets Init, Terminate, Increment and Decrement. Player $\Box$ states with double border contain a gadget allowing to select arbitrary probability distributions with deterministic strategies (see text for details).

The *state* of the two-counter machine is encoded by a location l and two counter values $c_1, c_2 \in \mathbb{N}$, i.e., $\langle l, c_1, c_2 \rangle$. Given an initial location l_0 with both counter values 0, the *termination problem* asks to determine whether a terminal location l_t is reached. The problem is known to be undecidable [67].

Let $\mathcal{M}$ be a two-counter machine. We define the game $\mathcal{G}(\mathcal{M})$ incrementally. The game has two players, $\Box$ and $\Diamond$. For each type of instruction, we have a corresponding gadget, i.e., Init, Terminate, Increment, and Decrement, which are shown in Figure 5.8.

In this figure, player $\Box$ states with double border are of the form  (where transitions probabilities in stochastic states are uniform), which allows player $\Box$ to simulate any probability distribution with deterministic strategies.

Note that, for Increment and Decrement gadgets, $j \in \{1, 2\}$ refers to the counter (c_1 or c_2), on which the operation is applied in the instruction and i is the other counter, the value of which remains unchanged. The game $\mathcal{G}(\mathcal{M})$ is then constructed by connecting the instructions together in the following way.

- For the initial instruction "$l_0 : c_1 := c_2 := 0$ and goto l_k;", we use the Init gadget and link $(init)_{out}$ to $(q_k, op)_{in}$, if l_k is not terminal location where op is the operation type, and we link $(init)_{out}$ to $(term)_{in}$ if l_k is a terminal location;
- For the increment instruction "$l_i : c_j := c_j + 1$ and goto l_k;", we use the Increment gadget and link $(q_i, inc_j)_{out}$, if l_k is not terminal location where op is the operation type, and we link $(q_i, inc_j)_{out}$ to $(term)_{in}$ if l_k is a terminal location;
- For the decrement instruction if "$c_j = 0$ then goto l_k else $c_j = c_j - 1$ and goto $l_{k'}$;", we use the Decrement gadget and link $(q_i, dec_j)^{=0}_{out}$ to $(q_k, op_k)_{in}$ (resp. $(q_i, dec_j)^{>0}_{out}$ to $(q_{k'}, op_{k'})_{in}$), if l_k (resp. $l_{k'}$) is not a terminal location where op_k (resp. $op_{k'}$) is the operation type, and we link $(q_i, dec_j)^{=0}_{out}$ (resp. $(q_i, dec_j)^{>0}_{out}$) to $(term)_{in}$ if l_k (resp. $l_{k'}$) is a terminal location.

Note that the $(init)_{in}$ is also the initial state of the whole game and we use $s_{init} = (init)_{in}$ as an alias for it. We label the states as "targets" as follows. States a^t_j, a_j are labelled with atomic proposition $\mathsf{T_{a_j}}$, states b^t_j, b_j are labelled with atomic proposition $\mathsf{T_{b_j}}$ for $j \in \{1, 2\}$, states c, c^t are labelled with $\mathsf{T_c}$, and states $a^t_1, a^t_2, b^t_1, b^t_2, c^t$ are all labelled with $\mathsf{T_t}$. We consider the following multi-objective rPATL formula

$$\varphi = \langle\langle\{\Box\}\rangle\rangle \left(\mathsf{P}_{\geq\frac{1}{6}}[\mathsf{F\,T_{a_1}}] \wedge \mathsf{P}_{\geq\frac{1}{6}}[\mathsf{F\,T_{b_1}}] \wedge \mathsf{P}_{\geq\frac{1}{6}}[\mathsf{F\,T_{a_2}}] \wedge \mathsf{P}_{\geq\frac{1}{6}}[\mathsf{F\,T_{b_2}}] \wedge \mathsf{P}_{\geq\frac{1}{3}}[\mathsf{F\,T_c}] \wedge \mathsf{P}_{\geq 1}[\mathsf{F\,T_t}]\right).$$

We show that player $\Box$ has a winning strategy in $\mathcal{G}(\mathcal{M})$ if and only if $\mathcal{M}$ does *not* terminate. The intuition behind the result is that if there exists a winning strategy for player $\Box$, it has to play probability distributions corresponding to gadget-specific counter updates and, in addition, never reach a terminal gadget. For example, consider the Init gadget, where in states s_0 and s_2 player $\Box$ has to pick probability $\frac{2}{3 \cdot 2^0}$ to go to s_1 and s_3 states, corresponding to the initialisation of counter values to 0. Similarly, the Increment gadget has the property that if the probability of selecting edge (s_5, s_6) for player $\Box$ is $\frac{2}{3 \cdot 2^{c_j}}$, then the probability to select the edge (s_9, s_{10}) must be $\frac{2}{3 \cdot 2^{c_j+1}}$, corresponding to a increment of counter j. In our case if Increment gadget follows the Init gadget, player $\Box$ has to play $\frac{2}{3 \cdot 2^0}$ to go to s_6 and s_8 states, and thus in s_9 and s_{11} he has to pick probabilities $\frac{2}{3 \cdot 2^1}$ and $\frac{2}{3 \cdot 2^0}$ to go to s_{10} and s_{12}, respectively, corresponding to the increment of the first counter.

Essentially, this means that if there is a winning strategy for player $\Box$, there is a unique one having a specific form, i.e., picking probability distributions corresponding to the counter updates induced by the gadgets, which in turn correspond to the instructions of the two-counter machine. $\Box$

We note that the question whether the multi-objective rPATL model checking problem is decidable when coalition player strategies are not restricted to deterministic remains open. Similar is results are known for the problem of finding Nash equilibrium in SMGs, where the problem is undecidable for deterministic strategies and open in general [107]. In the previous section, we have shown in Theorem 13, that memoryless deterministic strategies are sufficient for disjunctive rPATL with expected total reward objectives. We further show that finding such a strategy is not an easy problem by proving its NP-completeness. We also note that it contrasts with the problem of finding the optimal strategy in the single-objective expected reward games, which is known to be in NP$\cap$coNP.

Theorem 16 (NP-completness) *The problem of deciding whether the multi-objective rPATL formula $\langle\langle C \rangle\rangle(\bigvee_{j=1}^{m} \mathsf{R}^{r_j}_{\geq q_j}[\mathsf{F}^c \bot])$ is satisfied in a state of an SMG is* NP*-complete.*

Proof. We start with NP membership. Since memoryless deterministic strategies suffice for achieving such disjunctive rPATL objective (see Theorem 13), to determine whether such a formula is achievable we can guess such a strategy for player $\Box$ in the coalition game $\mathcal{G}_C$, which uniquely determines an MDP (obtained from $\mathcal{G}$ by resolving nondeterminism in player $\Box$ states). We can then use the polynomial time algorithm to verify that there exists no winning player $\Diamond$ strategy in this MDP (see Section 3.2.3 for details).

We prove NP-hardness by reducing 3SAT to the problem. Let Ψ be a 3CNF formula with clauses $c_1, \ldots, c_n$ and variables $x_1, \ldots, x_m$. We assume that each variable appears at most once in each clause. For clause j, we denote by v_1^j, v_2^j, v_3^j the variables that appear in the clause, e.g., $v_k^j \in \{x_1, \ldots, x_m, \neg x_1, \ldots, \neg x_m\}$ for $1 \leq k \leq 3$, $1 \leq j \leq n$.

We construct the game shown in Figure 5.9, where the terminal states are labelled with x_i^+ and v_i^- for all $1 \leq i \leq m$, corresponding to the valuations of the variables. We further construct $2m$ target sets, each a singleton containing either x_i^+ or x_i^-. The disjunctive

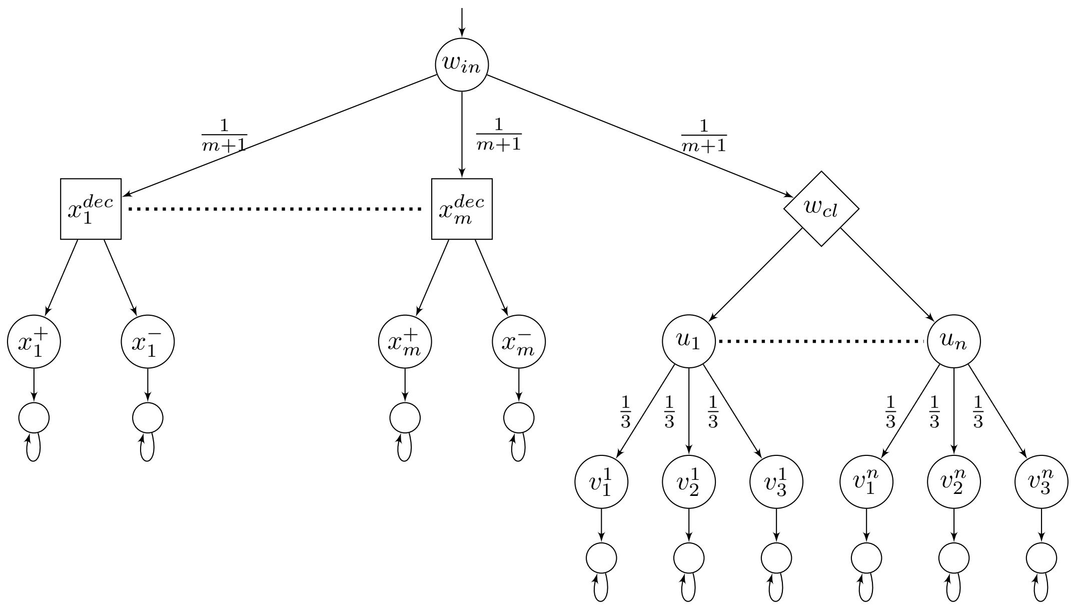


Figure 5.9: Game illustrating NP-hardness. The set of atomic proposition is $AP = \{\mathsf{x}_1^+, \mathsf{x}_1^-, \dots, \mathsf{x}_m^+, \mathsf{x}_m^-\}$, and the labelling function is defined as follows $\chi(x_i^+) = \{\mathsf{x}_i^+\}$, $\chi(x_i^-) = \{\mathsf{x}_i^-\}$, $\chi(v_k^j) = \{\mathsf{x}_i^+\}$ if $v_k^j = x_i$ and $\chi(v_k^j) = \{\mathsf{x}_i^-\}$ if $v_k^j = \neg x_i$ for $1 \le i \le m$, $1 \le j \le n$, $1 \le k \le 3$. For all other states χ returns an empty set.

rPATL formula that we consider is

$$\varphi = \langle\langle\{\Box\}\rangle\rangle(\mathsf{R}_{\ge q}^{r_1^+}[\mathsf{F}^c\bot] \vee \cdots \vee \mathsf{R}_{\ge q}^{r_m^+}[\mathsf{F}^c\bot] \vee \mathsf{R}_{\ge q}^{r_1^-}[\mathsf{F}^c\bot] \vee \cdots \vee \mathsf{R}_{\ge q}^{r_m^-}[\mathsf{F}^c\bot])\,,$$

where $q = \frac{1}{m+1} + \frac{1}{m+1} \cdot \frac{1}{n} \cdot \frac{1}{3}$ and reward functions are defined as follows. $r_i^+(s) = 1$ if $\mathsf{x}_i^+ \in \chi(s)$ and 0 otherwise, and $r_i^-(s) = 1$ if $\mathsf{x}_i^- \in \chi(s)$ and 0 otherwise. Note that reward functions assign 1 exactly to the paths that reach the states labelled with the appropriate atomic proposition; hence the value of the reward function is exactly the probability to reach such states. We claim that there is a satisfying assignment to Ψ if and only if there is a strategy $\sigma_\Box$ achieving φ.

For the "$\Rightarrow$" direction, given a satisfying assignment μ, we define a strategy $\sigma_\Box$ that goes to x_i^+ from x_i^{dec} if and only if $\mu(x_i) = 1$ and to x_i^- otherwise, for all $1 \le i \le m$. Consider any strategy $\sigma_\Diamond$ for player $\Diamond$, and let j be such that $\sigma_\Diamond$ picks u_j with probability at least $\frac{1}{n}$ in w_{cl} (such j surely exists). There must be a literal in c_j which is satisfied under μ. Let x_i be a variable in this literal. If the literal is of the form x_i, then we get that the state x_i^+ is reached on a path $w_{in}x_i^{dec}x_i^+$ with probability $\frac{1}{m+1}$ and on a path $w_{in}w_{cl}u_jv_k^j$, where $\mathsf{x}_i^+ \in \chi(v_k^j)$, with probability at least $\frac{1}{m+1} \cdot \frac{1}{n} \cdot \frac{1}{3}$, and so the objective is satisfied. Similarly, if the literal is of the form $\neg x_i$, we get the same line of argument, replacing x_i^+ with x_i^-.

For the "$\Leftarrow$" direction, we assume that $\sigma_\Box$ is memoryless deterministic (see Theorem 13). Define a valuation μ by $\mu(x_i) = 1$ if and only if successor x_i^+ is picked in x_i^{dec}. Let c_j be an arbitrary clause in Ψ, and consider a strategy $\sigma_\Diamond$ of player $\Diamond$, which goes deterministically to u_j in w_{cl}. There must be a target label $\mathsf{t} \in AP$ satisfying $\Pr_{w_{in}}(\mathsf{F}\,\mathsf{t}) \geq q$. Fix one such label t, and suppose that $\mathsf{t} = \mathsf{x_i^+}$. This set can be reached by the path $w_{in}v_i^{dec}x_i^+$, and the paths starting with $w_{in}w_{cl}$. Since the first path has probability only $\frac{1}{m+1}$, the other paths must have a non-zero probability. But since $\sigma_\Diamond$ is deterministic and selects u_j, there must be a path $w_{in}w_{cl}u_jv_k^j$, where $\mathsf{x_i^+} \in \chi(v_k^j)$, which means that the literal x_i is set to true in c_j under μ. Since this literal is true under μ, c_j is satisfied. For $\mathsf{t} = \mathsf{x_i^-}$ we proceed similarly. $\Box$

Remark 5 *Note that terminal state reachability objectives can be transformed into expected total reward ones (i.e., the ones used in Theorem 16) in linear time, and hence the* NP*-completeness result also holds for such objectives.*

Discussion. In this section we have shown several bounds for complexity of model checking multi-objective rPATL. Note that, to prove all the theorems, we used terminal state reachability and the expected total reward objectives, which are among the simplest in the logic. This way we make sure the complexity results are not due to the transformations as the ones presented in Section 5.2.1, and thus the results are tight in this respect.

The NP-hardness result proved in Theorem 16 showed that finding the memoryless deterministic strategy, existence of which was shown in Theorem 13, is a hard problem. For single-objective games with similar objectives (i.e., reachability, parity, expected total reward), the problem of finding such strategy is in the complexity class NP∩coNP. Such result cannot be obtained here because, due to nondeterminacy (shown in Theorem 10), player $\Diamond$ counter-strategy may be dependent on the strategy proposed by player $\Box$ (the idea leveraged by our NP-hardness proof). Another important consequence of this theorem lies in the game used to prove NP-hardness. Note that it is a tree of depth fixed depth, so if any value iteration-based algorithm would be derived, the cost of single iteration would be very expensive (unless P=NP), this justifies our method of approximation where we use the repeated solution of single-objective stochastic game to compute the approximation of the Pareto set (see Section 5.2).

5.4 Strategy synthesis

In this section we present a construction of the strategy for coalition C of players to satisfy the multi-objective rPATL formula $\varphi(\vec{q}, \vec{u}) = \langle\langle C \rangle\rangle \bigwedge_{i=1}^{n} \bigvee_{j=1}^{m} \mathsf{R}_{\geq u_{i,j}}^{q_{i,j}} [\,\mathsf{F}\ \bot\,]$ in a stopping game $\mathcal{G}$. Note that, as we have shown in Section 5.2.1, any multi-objective rPATL formula $\langle\langle C\rangle\rangle\theta$ can be converted into such form. The strategy construction extends the similar construction presented in [42] to deal with negative rewards.

We construct the strategy from the model checking algorithm presented in Section 5.2.3. Let $\vec{u}$ be a point in the $\varepsilon-$approximation of the Pareto set Q for $\varphi(\vec{q},\vec{u})$ and let $\vec{x} = (\vec{x}_1,\ldots,\vec{x}_n)$ be vectors for which $\vec{v} = (\vec{x}_1{\cdot}\vec{u}_1,\ldots,\vec{x}_n{\cdot}\vec{u}_n) \in X^k_s$, where the sets X^k are the ones computed using the iteration from Theorem 8, for a reward function vector $\vec{r} = (\vec{x}_1\cdot\vec{q}_1,\ldots,\vec{x}_n\cdot\vec{q}_n)$ up to $\frac{\varepsilon}{2}-$precision. We show how to construct a strategy that, for a given state s, guarantees that at least vector $\vec{v}-\mathsf{dir}(\vec{v}){\cdot}\delta$ is achieved where $\delta = \frac{\varepsilon}{2}{\cdot}|S|{\cdot}p_{\min}^{-|S|}$.

The strategy construction uses the idea of stochastic update in a similar way to the one we used to construct a linear size winning strategy to achieve precise reachability probability in the game from Figure 5 in Section 3.4. This time, instead of keeping only two values per state, the strategy keeps the corner points of the polytope X^k_s for each state s. The memory updates are executed in a way where the memory element is picked stochastically to ensure that, at every step, the expected value of the memory element (which is a lower bound for the expected value of the total reward) is kept within $\frac{\varepsilon}{2}$ of the target vector $\vec{v}$.

We denote the set of vertices (corner points) of a polytope X as $Cnr(X)$. The strategy $\sigma_\Box = \langle\mathcal{M},\sigma^u_\Box,\sigma^n_\Box,\alpha\rangle$ is defined as follows.

- $\mathcal{M} = \bigcup_{s\in S'}\{(s,\vec{p})\,|\,\vec{p}\in Cnr(X^k_s)\}$.
- $\sigma^u_\Box((s,\vec{p}),t) = [(t,\vec{q^t_0})\mapsto\beta^t_0,\ldots,(t,\vec{q^t_l})\mapsto\beta^t_l]$, where where for all $0\le i\le l$, $\vec{q^t_i}\in Cnr(X^k_t)$, $\beta^t_i\in[0,1]$, and $\sum_i\beta^t_i = 1$, such that
  - for $s\in S_\Box\cup S_\Diamond$ we have $\sum_i\beta^t_i\cdot\vec{q^t_i}\ge\vec{p}-\vec{r}(s)-\frac{\varepsilon}{2}$,
  - for $s\in S_\bigcirc$, $\vec{q^t_i}$ and β^t_i have to be chosen together with the respective values $\vec{q^v_i}$, and β^v_i assigned by $\sigma^u_\Box((s,\vec{p}),v)$ for the remaining successors $v\in S\setminus\{t\}$ of s, so that they satisfy
  $$\Delta(s,t)\cdot\sum_i\beta^t_i\cdot\vec{q^t_i}+\sum_{v\in S\setminus\{t\}}\Delta(s,v)\cdot\sum_i\beta^v_i\cdot\vec{q^v_i}\ge\vec{p}-\vec{r}(s)-\frac{\varepsilon}{2},$$
  which ensures that the expected total reward is kept larger than the current memory element.
- $\sigma^n_\Box(s,(s,\vec{p})) = [t\mapsto 1]$ for some $t\in S$ such that for all $0\le i\le l$ there exist $\vec{q^t_i}\in Cnr(X^k_t)$, and $\beta^t_i\in[0,1]$, such that $\sum_i\beta^t_i = 1$ and $\sum_i\beta^t_i\cdot\vec{q^t_i}\ge\vec{p}-\vec{r}(s)-\frac{\varepsilon}{2}$.

- $\alpha(s) = [(s, \vec{q_0^s}) \mapsto \beta_0^s, \dots, (s, \vec{q_l^s}) \mapsto \beta_l^s]$, where s is the respective initial state of $\mathcal{G}$, and $\vec{q_i^s} \in Cnr(X_s^k)$, $\beta_i^s \in [0,1]$ (for all $0 \le i \le l$), and $\sum_i \beta_i^s = 1$ such that $\sum_i \beta_i^s \cdot \vec{q_i^s} \ge \vec{v}$.

Note that, for all $t \in S$, it is always possible to choose $l \le n$, i.e., the number of points $\vec{q_i^t}$ and respective coefficients β_i^t may be less than the number of objectives. Also, the points $\vec{q_i^s}$ can indeed be picked from X_s^k because from Theorem 8 we have that $X_s^k \supseteq X_s^{k-1} - \frac{\varepsilon}{2}$. The correctness of the construction can be proved in a similar way as for Theorem 17 showing that the expectation of the memory element if preserved within $\frac{\varepsilon}{2}$ precision in every step and we can obtain the claimed error bound δ in the following way. In every step, the strategy, may introduce an error of at most $\frac{\varepsilon}{2}$, and, because the game is stopping, in $|S|$ steps the probability for the game to terminate is at least $p_{\min}^{|S|}$, so the upper bound on the error introduced into the expectation is $\frac{\varepsilon}{2} \cdot |S| + \frac{\varepsilon}{2} \cdot |S| \cdot (1 - p_{\min}^{|S|}) + \frac{\varepsilon}{2} \cdot |S| \cdot (1 - p_{\min}^{|S|})^2 + \cdots = \frac{\varepsilon}{2} \cdot |S| \cdot p_{\min}^{-|S|} = \delta$.

Example 12 *We provide an example of strategy construction using the method defined in this section. Consider the stochastic game shown in Figure 5.10a, and the conjunctive rPATL query* $\varphi(\vec{v}) = \langle\langle\{\Box\}\rangle\rangle(\mathsf{P}_{\ge v_1}[\mathsf{F}\,\mathsf{x}] \wedge \mathsf{P}_{\ge v_1}[\mathsf{F}\,\mathsf{y}])$. *The Pareto sets for this formula for all states of the game can be computed in three iterations of the functional from Theorem 8 and are shown in Figures 5.10b to 5.10e.*

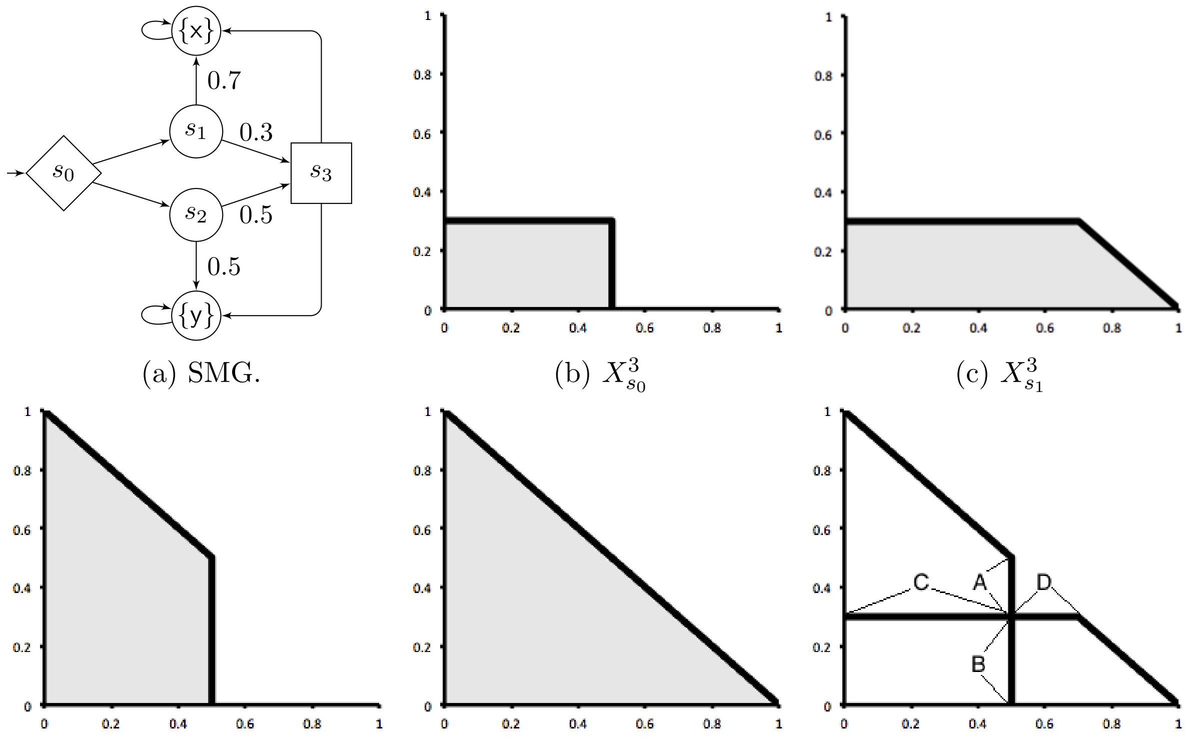
We show how to achieve the point $(0.5, 0.3)$, *e.g., how to use the algorithm above to construct a winning strategy* $\sigma_\Box = \langle \mathcal{M}, \sigma_\Box^u, \sigma_\Box^n, \alpha \rangle$ *for player* $\Box$ *with the conjunctive rPATL objective* $\langle\langle\{\Box\}\rangle\rangle(\mathsf{P}_{\ge 0.5}[\mathsf{F}\,\mathsf{x}] \wedge \mathsf{P}_{\ge 0.3}[\mathsf{F}\,\mathsf{y}])$. *The sets of corner points (also shown in the figures) for the states are as follows:*

$$\begin{aligned}
Cnr(X_{s_0}^3) &= \{(0, 0.3), (0.5, 0.3), (0.5, 0)\}, \\
Cnr(X_{s_1}^3) &= \{(0, 0.3), (0.7, 0.3), (1, 0)\}, \\
Cnr(X_{s_2}^3) &= \{(0, 1), (0.5, 0.5), (0.5, 0)\}, \textit{ and} \\
Cnr(X_{s_2}^3) &= \{(0, 1), (1, 0)\}.
\end{aligned}$$

So the memory elements of the strategy are as follows:

$$\begin{aligned}
\mathcal{M} = \{ \quad & (s_0, (0, 0.3)), (s_0, (0.5, 0.3)), (s_0, (0.5, 0)), \\
& (s_1, (0, 0.3)), (s_1, (0.7, 0.3)), (s_1, (1, 0)), \\
& (s_2, (0, 1)), (s_2, (0.5, 0.5)), (s_2, (0.5, 0)), \\
& (s_3, (0, 1)), (s_3, (1, 0))\}.
\end{aligned}$$

The initial distribution function is defined by $\alpha(s_0) = (s_0, (0.5, 0.3))$, *since this is the*



(a) SMG. (b) $X^3_{s_0}$ (c) $X^3_{s_1}$

(d) $X^3_{s_2}$ (e) $X^3_{s_3}$ (f) Choices when leaving s_0

Figure 5.10: Pareto sets for the conjunctive rPATL formula $\varphi(\vec{v}) = \langle\langle\{\Box\}\rangle\rangle(\mathsf{P}_{\geq v_1}[\mathsf{F}\,\mathsf{x}] \wedge \mathsf{P}_{\geq v_2}[\mathsf{F}\,\mathsf{y}])$. The horizontal axis represents values for v_1 and the vertical axis represents values for v_2.

point that we want to achieve. Also, the definition of the next move function for the only state of player $\Box$, s_3, is straightforward, i.e., we have $\sigma^n_\Box(\,s_3, (s_3, (1, 0))\,) = x$ and $\sigma^n_\Box(s_3, (s_3, (0, 1))) = y$.

We now describe the memory update function. In the state s_0, if player $\Diamond$ picks the action to go to s_1, then we have

$$\sigma^u_\Box(\,(s_0,\, (0.5, 0.3)),\, s_1\,) = [(s_1,\, (0, 0.3)) \mapsto p,\, (s_1,\, (0.7, 0.3)) \mapsto 1{-}p],$$

where the probability $p = \frac{2}{7}$ so that $p \cdot 0 + (1{-}p) \cdot 0.7 = 0.5$ and $p \cdot 0.3 + (1{-}p) \cdot 0.3 = 0.3$. The reason for the choice of p is shown in Figure 5.10f: we can see that the point $(0.5, 0.3)$ is composed of C parts of point $(0.7, 0.3)$ and D parts of $(0, 0.3)$ in $X^3_{s_1}$, and hence we have $p = \frac{D}{C+D} = \frac{0.2}{0.5+0.2} = \frac{2}{7}$. Similarly, if player $\Diamond$ chooses to go to s_2, the memory update function is defined by

$$\sigma^u_\Box(\,(s_0,\, (0.5, 0.3)),\, s_2\,) = [(s_2,\, (0.5, 0))\mapsto p,\, (s_2,\, (0.5, 0.5)) \mapsto 1{-}p],$$

where $p = \frac{2}{5}$, because in the point $(0.5, 0.3)$ is composed of A parts of $(0.5, 0)$ and B parts *of* $(0.5, 0.5)$*, as shown in Figure 5.10f, so we have* $p = \frac{A}{A+B} = \frac{0.2}{0.2+0.3} = \frac{2}{5}$.

Finally, we define the memory update function for the stochastic states. For s_1 *we have*

$$\sigma^u_\Box(\,(s_1,\,(0,0.3)),\,s_3\,) = \sigma^u_\Box(\,(s_1,\,(0.7,0.3)),\,s_3\,) = (s_3,\,(0,1)),$$

and for s_2 *we have*

$$\sigma^u_\Box(\,(s_2,\,(0.5,0)),\,s_3\,) = \sigma^u_\Box(\,(s_2,\,(0.5,0.5)),\,s_3\,) = (s_3,\,(1,0)).$$

Essentially, the strategy always chooses to go to the state y *if player* $\Diamond$ *chooses to go to* s_1*, and to the state* x*, if player* $\Diamond$ *chooses to go to* s_2*. One can easily verify that if player* $\Diamond$ *chooses to go to* s_1*, then the reachability probabilities to* x *and* y *are* 0.7 *and* 0.3*, respectively; and, if he chooses to go to* s_2*, then reachability probabilities are* 0.5 *and* 0.5*, and hence the strategy achieves the desired objective* $\langle\langle\{\Box\}\rangle\rangle(\mathsf{P}_{\geq 0.5}[\mathsf{F}\,\mathsf{x}] \wedge \mathsf{P}_{\geq 0.3}[\mathsf{F}\,\mathsf{y}])$*. Note that we constructed the memory update (and hence the choices of the strategy) in such a way that the expectation of the memory element of the strategy is at least* $(0.5, 0.3)$*, and so the actual value achieved is always at least this vector.*

5.5 Multi-objective rPATL*

To conclude our presentation of the multi-objective rPATL, we extend multi-objective rPATL to incorporate LTL into its path formulae. We also provide model-checking algorithm for the multi-objective rPATL* for stopping games via a reduction to multi-objective rPATL model checking. We begin by introducing the syntax and semantics of the logic.

Definition 10 (Multi-objective rPATL*) *The syntax of the multi-objective rPATL* is given by the following grammar:*

$$\begin{aligned}
\phi &::= \top \mid a \mid \neg\phi \mid \phi \wedge \phi \mid \langle\langle C \rangle\rangle\theta \\
\theta &::= \mathsf{P}_{\bowtie q}[\psi] \mid \mathsf{R}^r_{\bowtie x}[\mathsf{F}\,\phi] \mid \theta \wedge \theta \mid \theta \vee \theta \mid \neg\theta \\
\psi &::= \phi \mid \neg\psi \mid \psi \wedge \psi \mid \mathsf{X}\,\psi \mid \psi\,\mathsf{U}\,\psi
\end{aligned}$$

where $a \in AP$*,* $C \subseteq \Pi$*,* $\bowtie \in \{<, \leq, \geq, >\}$*,* $q \in \mathbb{Q} \cap [0,1]$*,* $x \in \mathbb{Q}_{\geq 0}$*,* r *is a reward structure.*

The semantics for the logic is, for all except path formulae, as in multi-objective rPATL (see Definition 9), and for path formulae ψ semantics is the same as for rPATL* (see Section 4.5). Example properties of multi-objective rPATL* are as follows.

- $\langle\langle\{sensor1, sensor2\}\rangle\rangle(\mathsf{P}_{\geq 1}[\mathsf{G}\,(\mathsf{on1} \vee \mathsf{on2})] \wedge \mathsf{P}_{\leq 0.1}[\mathsf{F}\,(\mathsf{appears} \wedge \neg\mathsf{detected})])$ – "Sensors 1 and 2 have a strategy such that at least one of them always remains on and that the probability that the suspect appears and is not detected is less that 0.1, no matter what actions are taken by the other players (e.g., player *suspect*)."
- $\langle\langle\{algorithm\}\rangle\rangle(\mathsf{R}^{income}_{\geq 100}[\mathsf{F}^c\bot] \wedge \mathsf{P}_{\geq 1}[\mathsf{G}\,\mathsf{X}\,\langle\langle\{algorithm\}\rangle\rangle(\mathsf{P}_{\geq 1}[\mathsf{F}\,\mathsf{exit}] \wedge \mathsf{R}^{cost}_{\leq 95}[\mathsf{F}^c\mathsf{exit}])])$ – "There exists strategy for the *algorithm* which generates expected income of at least 100, but also there always is an option to go to the state where it has a strategy to exit the position at an expected cost of 95, no matter what actions other players take in the game."

Example 13 *Multi-objective rPATL* allows the usage of rPATL* formulae as objectives of the game. So, for example, we can enrich the formulae discussed in Example 9 with resource constraints to obtain, e.g.,* $\langle\langle\{\Box, \triangle, \Diamond\}\rangle\rangle(\mathsf{P}_{>0.4}[(\mathsf{F}\,\mathsf{s}_5) \wedge (\mathsf{F}\,\mathsf{T}_2)] \wedge \mathsf{R}^{cost}_{\leq 99}[\mathsf{F}^c\,\mathsf{T}_2])$ *states that the grand coalition of players has a strategy to make sure that the probability of the path visiting both, state* s_5 *and a state labelled with* T_2 *is greater than 0.4, and the expected cost incurred during the execution before reaching* T_2 *is at most 99 units.*

Consider also the formula $\varphi(\vec{v}) = \langle\langle\{\Box, \triangle\}\rangle\rangle(\mathsf{P}_{\geq v_1}[\mathsf{GF}\,\mathsf{s}_{12}] \vee \mathsf{P}_{\geq v_2}[\mathsf{GF}\,\mathsf{s}_9])$ *and a game starting in* s_0 *of an SMG from Figure 4.1. It can be shown that the formula* $\varphi(\vec{v})$ *is satisfied if and only if* $v_1 + v_2 \leq 1$ *(for* $v_1, v_2 \in [0,1]$*). To see this, recall that, in Example 9, we have shown that the rPATL* formula* $\langle\langle\{\Box, \triangle\}\rangle\rangle\mathsf{P}_{\geq 1}[(\mathsf{GF}\,\mathsf{s}_{12}) \vee (\mathsf{GF}\,\mathsf{s}_9)]$ *is true, because no matter which strategy player* $\Diamond$ *chooses, any path of the game satisfies either* $\mathsf{GF}\,\mathsf{s}_{12}$ *or* $\mathsf{GF}\,\mathsf{s}_9$*. In this case,* $\Diamond$ *can choose randomly between these outcomes, and hence if* $v_1 + v_2 > 1$*, it can always pick a distribution such that the formula is false. For example, consider the case where* $v_1 = 0.5$ *and* $v_2 = 0.6$*. Player* $\Diamond$ *can choose a strategy which, with probability* 0.55 *chooses to take action 'init-1', and thus achieves probability* 0.55 *to satisfy* $\mathsf{GF}\,\mathsf{s}_9$*, and with probability* 0.45 *it always takes action 'idle', making sure that probability for a path to satisfy* $\mathsf{GF}\,\mathsf{s}_{12}$ *is* 0.45*. This makes sure that both objectives,* $\mathsf{P}_{\geq 0.6}[\mathsf{GF}\,\mathsf{s}_9]$ *and* $\mathsf{P}_{\geq 0.5}[\mathsf{GF}\,\mathsf{s}_{12}]$*, are false, and hence the formula is not satisfied in* s_0*. Using the same reasoning one can easily see that if* $v_1 + v_2 \leq 1$ *then, no matter how* $\Diamond$ *divides the probabilities between the objectives, at least one of them is satisfied.*

Next, we consider model checking of rPATL* for *stopping games*. We show how model checking multi-objective rPATL* for such games can be reduced to model checking multi-objective rPATL by providing, in the style of Section 5.2.1, a replacement procedure for $\mathsf{P}_{\bowtie q}[\psi]$ operator by $\mathsf{R}^r_{\bowtie x}[\mathsf{F}^c\bot]$. This transforms multi-objective rPATL* formula into multi-objective rPATL and the algorithms from Section 5.2 can be used for model checking.

“$\mathsf{P}_{\bowtie q}[\psi]$ to $\mathsf{R}^{r}_{\bowtie x}[\mathsf{F}^{c}\bot]$”

Let us consider rPATL* formula $\langle\langle C\rangle\rangle\theta$ where θ contains $\mathsf{P}_{\bowtie q}[\psi]$ operator and a stopping SMG $\mathcal{G} = \langle \Pi, S, (S_{\bigcirc}, (S_i)_{i\in\Pi}), \Delta, AP, \chi\rangle$ with a set of terminal states Term. We start by constructing a *deterministic* Rabin automaton for the LTL formula ψ.

Definition 11 (Rabin automaton) *A* deterministic Rabin automaton *is a tuple $\mathcal{A} = \langle Q, \Sigma, \tau, q_0, ((L_1, R_1), \dots, (L_j, R_j))\rangle$, where Q are states of automaton with initial state $q_0 \in Q$, $\Sigma = S$ is the alphabet, $\tau : Q \times \Sigma \to Q$ is a transition function and $L_l, R_l \subseteq Q$ are Rabin pairs.*

We construct automaton $\mathcal{A}$ such that any path λ satisfies ψ iff λ is accepted by $\mathcal{A}$. Without loss of generality we assume each DRA is *complete* (i.e., every infinite word induces an infinite run on the automaton). We construct a new game $\mathcal{G}'$ as follows.

- $\Pi' = \Pi$;
- $S' = \{(s, q, I) \mid s \in S, q \in Q, I \in \{0, 1\}\}$;
- $(s, q, I) \in S'_i$ if $s \in S_i$ for $i \in \Pi$;
- $\Delta'((s, q, I), (t, p, J)) = x$ whenever
  - $\Delta(s, t) = x$;
  - if $I = 0$ then $\tau(q, t) = p$;
  - if $I = 1$ then $p = q$ and $I = J$;
  - $J = 1$ iff $s \in \mathsf{Term}$;
  - $J = 0$ iff $s \notin \mathsf{Term}$;
- $AP' = AP$;
- $\chi'((s, q, I)) = \chi(s)$.

A new set of terminal states for the game is $\mathsf{Term}' = \{(s, q, I) \in S' \mid I = 1\}$. The reward function r is defined by $r(s, q, I) = 1$ if $s \in \mathsf{Term}$, $I = 0$ and the run of $\mathcal{A}$ on the word s^{ω} starting from q is accepting, and otherwise $r(s, q, I) = 0$. For a state s of $\mathcal{G}$, $\langle\langle C\rangle\rangle\theta$ is satisfied if and only if $\langle\langle C\rangle\rangle\theta'$ is satisfied in $(s, \tau(q_0, s), 0)$ of $\mathcal{G}'$. To see this, it suffices to observe that, due to the constructions above, the reward function r assigns reward 1 to the path in $\mathcal{G}'$ if and only if the path ends in a terminal state and the word that it generates is accepted by DRA $\mathcal{A}$ (and hence, it satisfies ψ). For stopping games, we do not need to consider paths, which do not reach terminal states, since any set consisting of only such paths has probability 0.

5.6 Summary

The contributions of this chapter can be summarised as follows.

Logic. We have extended the logic rPATL to support multiple objectives within the coalition operator. This allows us to specify properties like $\langle\langle\{controller\}\rangle\rangle(\mathsf{P}_{\geq 0.99}[\mathsf{F}\,\mathsf{target}] \wedge \mathsf{R}^{energy}_{\leq 100}[\mathsf{F}^c\,\mathsf{target}])$, which states that *controller* has a strategy to guarantee that probability to reach the target is at least 0.99, while simultaneously, achieving the expected energy consumption of at most 100 units.

Algorithms. We have provided approximation algorithms to model check the logic in stopping games. Similarly to the algorithms for rPATL, which were based on the optimal value computation, algorithms for multi-objective rPATL are based on the computation of the Pareto sets – the values for bounds for which the formula is true. The algorithms that we provide are based on the iterative evaluation of recursive equations, providing successive approximations of the Pareto sets.

Complexity. We have analysed the complexity of the games with multi-objective rPATL objectives and showed that they differ significantly from the single-objective games. First of all, the games are not determined already for two objectives; also optimal strategies may not exist for some points in the Pareto set. The model checking problem is PSPACE-hard in general and undecidable if one restricts coalition players to use deterministic strategies. We have also analysed the memory requirements for the optimal strategies and showed that infinite memory randomised strategy may be required for the coalition to achieve a multi-objective rPATL formula. For the case where the formulae are restricted to disjunctions of expected total reward goals, memoryless strategies suffice, but the problem is NP-complete.

Synthesis. We have provided a method to construct a stochastic-update strategy for the coalition, which achieves the given objective in a state of a stopping SMG with a specified precision. The construction uses the Pareto set approximations computed by the model checking algorithms as its memory elements. This strategy construction has been used in a case study on autonomous urban driving in [42].

Discussion. In addition to providing means to verify system specifications containing several objectives, multi-objective rPATL allows to analyse the competitive nature of the system in more detail allowing, to some extent, to circumvent the drawbacks of zero-sum single-objective games. For example, consider a game with two players A and B having goals $\mathsf{P}_{\geq 0.5}[\mathsf{F}\,\mathsf{G_A}]$ and $\mathsf{P}_{\geq 0.9}[\mathsf{F}\,\mathsf{G_B}]$, respectively. We are interested in synthesising a strategy for agent A, which would achieve the objective in this game, but the rPATL formula $\neg\langle\langle\{A\}\rangle\rangle\mathsf{P}_{\geq 0.5}[\mathsf{F}\,\mathsf{G_A}]$ is true in the game meaning that we cannot find a strategy for A that achieves its objective for any strategy for B. However, since we know that B plays a strategy to achieve its own objective (we assume such strategy exists, i.e., $\langle\langle\{B\}\rangle\rangle \mathsf{P}_{\geq 0.9}[\mathsf{F}\,\mathsf{G_B}]$ is true), and hence quantification over all strategies of B is too pessimistic. In this case, we can use multi-objective rPATL formula $\langle\langle\{A\}\rangle\rangle(\mathsf{P}_{\geq 0.5}[\mathsf{F}\,\mathsf{G_A}] \vee \neg \mathsf{P}_{\geq 0.9}[\mathsf{F}\,\mathsf{G_B}])$ to check whether there exists agent A strategy, such that, if agent B plays a strategy to falsify the objective of A, then its own objective is not achieved; and if B plays a strategy to achieve its own objective, then agent A's objective is also satisfied. Such strategy would guarantee that, as long as agent B takes actions to achieve its objective, the objective of A is satisfied. Note that the strategy for player A that achieves $\langle\langle\{A\}\rangle\rangle(\mathsf{P}_{\geq 0.5}[\mathsf{F}\,\mathsf{G_A}] \vee \neg \mathsf{P}_{\geq 0.9}[\mathsf{F}\,\mathsf{G_B}])$ together with a strategy for player B that achieves $\langle\langle\{B\}\rangle\rangle \mathsf{P}_{\geq 0.9}[\mathsf{F}\,\mathsf{G_B}]$ form a winning secure equilibria strategy profile [32].

To the best of our knowledge, the work published in [39, 40] and presented in this chapter is the first one providing algorithms for multi-objective verification for stochastic games (the previous works of [8] and [19] showed undecidability for more general classes of problems). We have shown that, already for terminal state reachability objectives, the problem becomes more difficult than single-objective games and multi-objective MDPs. Nevertheless, we have provided approximation algorithms that can be used to implement verification of multi-objective rPATL.

There are several open questions raised by the research done in this chapter. Probably the most important one is whether the model checking problem for multi-objective rPATL is decidable in the general case. Also, the question of whether the infinite memory argument extends to the case where the coalition players are allowed to use stochastic-update strategies remains open (we have shown that such strategies can be more compact in certain cases). It would also be interesting to consider approximation algorithms for games without the stopping restriction for both multi-objective rPATL and rPATL* formulae.

Chapter 6

Tool Implementation and Applications

In this chapter we present PRISM-games, a model checker for stochastic multi-player games, which we built to implement rPATL model checking and strategy synthesis algorithms developed in this thesis. We also applied the tool to three case studies of competitive stochastic systems: distributed demand-side management algorithm for microgrid, collective decision making algorithm for sensor networks, and reputation and virtual currency mechanism for user centric-networks.

The tool is built as an extension of the PRISM model checker [77]. The PRISM-games implementation extends PRISM's explicit engine to support stochastic multi-player game models, rPATL property specifications and model checking and synthesis algorithms. In Section 6.1 we discuss the SMG modelling language and Section 6.2 describes the rPATL property specification notation. Then, in Section 6.3, we present the pseudocode of the model checking algorithms that are derived from results in Section 4.2. We also discuss the strategy synthesis functionality of PRISM-games in Section 6.4. In Section 6.5 we present the experimental results showing the scalability of the tool and discuss the trade-offs between time and precision for the value iteration algorithms. Finally, in Section 6.6 we provide an analysis of the three aforementioned case studies using rPATL model checking and strategy synthesis functionality of the tool.

PRISM-games is free and open source (GPL license), and runs on all major operating systems. It is available for download from `http://www.prismmodelchecker.org/games/`. The summary of the tool functionality and usage instructions is presented in Appendix E. Sections 6.1, 6.2, 6.3, 6.4 and 6.5 are an extended version of [38]. The case studies of the microgrid demand-side management and collective decision making algorithms are adapted from [37], and the user-centric network case study has been published in [81].

```
smg

player p1 scheduler, [do] endplayer
player p2 agent1, [go1] endplayer
player p3 agent2, [go2] endplayer

global num_tasks : [-1..2] init -1;

module scheduler
   turn : [1..3] init 1;
   [] num_tasks=-1 → 0.5 : (num_tasks'=1) + 0.5 : (num_tasks'=2);
   [go1] num_tasks>0 ∧ turn=1 → (turn'=2);
   [go2] num_tasks>0 ∧ turn=2 → (turn'=3);
   [do] num_tasks>0 ∧ turn=3 → (turn'=1) ∧ (num_tasks'=num_tasks − 1);
endmodule

module agent1
   team1 : [1..2] init 1;
   [go1] true → (team1'=1);
   [go1] true → (team1'=2);
endmodule

module agent2 = agent1 [go1=go2, team1=team2] endmodule

rewards "total"
   turn=3 ∧ team1≠team2 : 0.3;
   turn=3 ∧ team1=team2 : 1.0;
endrewards
```

Figure 6.1: A PRISM-games SMG model of a team formation game [41].

6.1 Modelling language

In PRISM-games, SMGs are described in a modelling language, which is an extension of the PRISM language (see [93] for details). An example of a PRISM-games model of a variant of team formation game is shown in Figure 6.1. We refer to the components of this example in the description of the modelling language given below.

A model is composed of *modules* (e.g., *scheduler*, *agent*1 and *agent*2), whose states are determined by a set of *variables* (e.g., *turn*, *team*1) and whose behaviour is specified by a set of *guarded commands*, containing an (optional) action label, a guard and a probabilistic update for the module's (or global) variables. For example, the command

$$[]\ \mathit{num\_tasks}{=}\text{-}1 \rightarrow\ 0.5\ :\ (\mathit{num\_tasks}'{=}1)\ +\ 0.5\ :\ (\mathit{num\_tasks}'{=}2);$$

updates the value of *num_tasks* to either 1 or 2 with equal probability, provided the current value is -1.

Simultaneous updates across several modules can be synchronised using action-labelled commands. For such command to be executed, the guard has to be satisfied in every module. For example, the command $[go1]$ `true` $\rightarrow$ $(team1'{=}1)$; of module *agent*1 can only be executed if the guard $num\_tasks{>}0 \ \wedge \ turn{=}2$ is satisfied in module *scheduler* as well. Such composition semantics ensures that each probabilistic transition in the model is associated with either an action label or a single module. A model also defines *players* (e.g., *p*1, *p*2 and *p*3), each of which is assigned a disjoint subset of the model's synchronising action labels and modules (e.g., player *p*2 controls action labelled with [*go*1]). This assigns each probabilistic transition to one player. To ensure a turn-based SMG, all possible probabilistic transitions in a state must belong to the same player; the tool detects and disallows concurrent actions. The rewards are assigned to states using the *rewards* construct. For example, reward "total" assigns reward of 0.3 to states satisfying $turn{=}3 \ \wedge \ team1{\neq}team2$ and 1 to states satisfying $turn{=}3 \ \wedge \ team1{=}team2$, and 0 to all other states.

This description results in a slightly different model than that of SMGs used previously, because the successor in a non-stochastic player's state can be chosen according to a probability distribution. This is a useful modelling convenience, but the two models are equivalent in their expressiveness. The SMG model as used in the thesis can be specified in PRISM-games directly and the PRISM-games' SMG can be converted into the one used here by inserting, for each action, a stochastic state representing the distribution. For a more detailed discussion see Section 6.3.

6.2 Property specification

PRISM-games supports rPATL logic specifications, including the quantitative form of the operators, e.g., $\langle\langle C\rangle\rangle \mathsf{P}_{\max=?}[\psi]$, which returns the maximum probability of ψ that coalition C can guarantee, instead of a true/false value. PRISM-games also supports *precise* value operators representing multi-objective rPATL queries $\langle\langle C\rangle\rangle \mathsf{P}_{=q}[\psi] \equiv \langle\langle C\rangle\rangle(\mathsf{P}_{\geq q}[\psi] \wedge \mathsf{P}_{\leq q}[\psi])$ and $\langle\langle C\rangle\rangle \mathsf{R}^r_{=x}[\mathsf{F}^c\phi] \equiv \langle\langle C\rangle\rangle(\mathsf{R}^r_{\geq x}[\mathsf{F}^c\phi] \wedge \langle\langle C\rangle\rangle \mathsf{R}^r_{\leq x}[\mathsf{F}^c\phi])$ for stopping games.

Example 14 *We several example properties, expressed in PRISM-games syntax, for the team formation game from Figure 4.1. We use B for player* $\Box$*, D for player* $\Diamond$ *and T for player* $\triangle$*.*

- `<<B,T,D>> P>=1 [F ("T1" & "T2")]` – *"players B, T and D have a joint strategy to guarantee that a state labelled with T1 and T2 is reached with probability* 1*."*

- `<<B,D>> R{"cost"}>=10 [Fc "T1"]` – *"players B and D can ensure that the expected total reward before reaching state labelled with T1 is at least 10."*

- `<<B,D>> R{"bonus"}max=? [F0 ("T1" | "T2")]` – *"what is the maximum expected bonus along the paths that reach T1 or T2 that players B and D can guarantee?"*

- `<<T>> R{"cost"}min=? [F "T1"]` – *"what is the minimum cost that player T can ensure before reaching T1?."*

- `<<B,D,T>> P=0.5 [!"T1" U "T2"]` – *"players B, D and T can guarantee the probability to reach T2 before reaching T1 is exactly 0.5."*

6.3 Model checking

This section describes the implementation of the rPATL model checking algorithms in PRISM-games that were derived from fixpoint equations given in Section 4.2.1.

As mentioned earlier, SMG modelling in PRISM-games has been built by extending PRISM's MDP model, which uses different (but equivalent) definition, where, in each state, a player can choose an action, which then leads to a successor state that is chosen according to a probability distribution. Formally, in addition to a game $\mathcal{G} = \langle \Pi, S, (S_{\bigcirc}, (S_i)_{i\in\Pi}), \Delta, AP, \chi \rangle$, we are given a set of actions A, and the transition function Δ is modified so that $\Delta : S \times A \to \mathcal{D}(S)$, and we denote by $A(s)$ the set of actions available in state s. For every stochastic state $s \in S_{\bigcirc}$, there is only one action available, and equations from Section 4.2.1 are modified accordingly, i.e., $\mathrm{opt}_{s\in\Delta(s)}\star$ is replaced by $\mathrm{opt}_{a\in A(s)} \sum_{t\in S} \Delta(s,a,t) \cdot \star$ where $\mathrm{opt} \in \{\max, \min\}$. We present rPATL model checking algorithms in pseudocode. Given a game $\mathcal{G}$ and an rPATL formula ϕ the algorithms presented here compute the set of states *sat* that satisfy the formula.

The general model checking routine is shown in Algorithm 1, which recursively traverses the formula's parse tree and calls the appropriate model checking sub-routines in the process. Standard logic operations are evaluated in the expected way. The routine for checking probabilistic rPATL formula $\langle\langle C\rangle\rangle \mathsf{P}_{\bowtie x}[\psi]$ is shown in Algorithms 2 and 3. Note that lines 2-4 of the algorithm use the determinacy result from Equations (4.1) to convert the comparison operator into "maximisation" by swapping the players in the coalition, e.g., for a state s of an SMG we have $\langle\langle C\rangle\rangle \mathsf{P}_{\leq q}[\psi] \Leftrightarrow \neg\langle\langle \Pi \setminus C\rangle\rangle \mathsf{P}_{>q}[\psi] \Leftrightarrow \mathrm{Pr}_s^{\max,\min}(\psi) \leq q$, where in $\mathrm{Pr}_s^{\max,\min}(\psi) \leq q$ maximisation is performed by players $\Pi \setminus C$.

Model checking for the "Next" operator is a one step process, shown in Algorithm 2, which checks which action gives the biggest probability to reach a target in the next step. For the "Until" and bounded "Until" we have to perform multiple iterations of

Algorithm 1 General model checking routine for rPATL.

```
1: procedure ModelCheck($\phi$, $\mathcal{G} = \langle \Pi, S, (S_{\bigcirc}, (S_i)_{i\in\Pi}), \Delta, AP, \chi, A\rangle$)
2:     $sat := \emptyset$
3:     if $\phi \equiv \top$ then
4:         $sat := S$
5:     else if $\phi \equiv a \in AP$ then
6:         $sat := \{s \in S \,|\, a \in \chi(s)\}$
7:     else if $\phi \equiv \neg\phi_1$ then
8:         $sat := S \setminus$ ModelCheck($\phi_1$, $\mathcal{G}$)
9:     else if $\phi \equiv \phi_1 \wedge \phi_2$ then
10:        $sat :=$ModelCheck($\phi_1$, $\mathcal{G}$) $\cup$ ModelCheck($\phi_2$, $\mathcal{G}$)
11:    else if $\phi \equiv \langle\langle C\rangle\rangle \mathsf{P}_{\bowtie x}[\psi]$ then
12:        $sat :=$ ModelCheckProb($\phi$, $\mathcal{G}$)
13:    else if $phi \equiv \mathsf{R}^{r}_{\bowtie x}[\mathsf{F}^{\star}\phi]$ then
14:        $sat :=$ ModelCheckRew($\phi$, $\mathcal{G}$)
15:    end if
16:    return $sat$
17: end procedure
18: procedure ProbReach($C$, $T$, $\mathcal{G}$)
19:    return $X$ such that $X[s] = \mathrm{Pr}_s^{\max,\min}(\mathsf{F}\,T)$
20: end procedure
21: procedure CumRew($C$, $T$, $\mathcal{G}$)
22:    return $X$ such that $X[s] = \mathbb{E}_s^{\max,\min}[rew(r, c, T)]$
23: end procedure
24: procedure Büchi($C$, $\mathcal{G}$)
25:    return $\{s \in S \mid \mathrm{Pr}_s^{\max,\min}(inf^{\mathsf{t}}(\mathsf{a}_{\mathrm{rew}})) > 0\}$
26: end procedure
```

Algorithm 2 Probability operator model checking routine (Next).

```
1: procedure ModelCheckProb($\langle\langle C\rangle\rangle \mathsf{P}_{\bowtie x}[\psi]$, $\mathcal{G}$)
2:     if $\bowtie \in \{\leq, <\}$ then
3:         $C := \Pi \setminus C$
4:     end if
5:     $sat := \emptyset$
6:     if $\psi \equiv \mathsf{X}\,\phi_1$ then
7:         $T :=$ ModelCheck($\phi_1$, $\mathcal{G}$)
8:         for all $s \in S$ do
9:             if $s \in \bigcup_{i\in C} S_i$ then
10:                $val := \max_{a\in A(s)} \sum_{t\in T} \Delta(s, a, t)$
11:            else
12:                $val := \min_{a\in A(s)} \sum_{t\in T} \Delta(s, a, t)$
13:            end if
14:            if $val \bowtie x$ then
15:                $sat := sat \cup \{s\}$
16:            end if
17:        end for
```

the fixpoint computation as shown in Algorithm 3. For the bounded version we execute exactly k iterations of the loop and for the unbounded version we use an absolute precision stopping criterion, where we execute the iteration until the maximum difference between the values of two consecutive iterations becomes smaller than a given constant ε. The effect on performance of the algorithm for different choices of ε is discussed in Section 6.5.

Model checking procedure for the reward operator is shown in Algorithm 4. Similarly as for the general formulae, here we determine which type of reward the formula uses and call the appropriate sub-routine after performing the swap of coalitions in order to convert

Algorithm 3 Probability operator model checking routine (Until).

```
    else if $\psi \equiv \phi_1 \mathsf{U}\, \phi_2$ or $\psi \equiv \phi_1 \mathsf{U}^{\leq k}\, \phi_2$ then
        $T_2 :=$ ModelCheck($\phi_2$, $\mathcal{G}$)
        $T_1 :=$ ModelCheck($\phi_1$, $\mathcal{G}$) $\setminus T_2$
        $iter := 0$
        $X := 0$
        $X' := 0$
        for all $s \in S$ do
            if $s \in T_2$ then $X[s] := 1$ else $X[s] := 0$ end if
        end for
        while $true$ do
            $X' := X$
            $iter := iter + 1$
            for all $s \in S$ do
                if $s \in T_1$ then
                    if $s \in \bigcup_{i \in C} S_i$ then
                        $X[s] := \max_{a \in A(s)} \sum_{t \in S} \Delta(s,a,t) \cdot X'[t]$
                    else
                        $X[s] := \min_{a \in A(s)} \sum_{t \in S} \Delta(s,a,t) \cdot X'[t]$
                    end if
                end if
                if $X[s] \bowtie x$ then
                    $sat := sat \cup \{s\}$
                end if
            end for
            if $\psi \equiv \phi_1 \mathsf{U}^{\leq k}\, \phi_2$ then
                if $k \leq iter$ then
                    break
                end if
            else if $\varepsilon > \max_{s \in S} |X[s] - X'[s]|$ then
                break
            end if
        end while
    end if
    return $sat$
end procedure
```

Algorithm 4 Model checking routine for the reward operator.

```
 1: procedure ModelCheckRew($\langle\langle C\rangle\rangle \mathsf{R}^r_{\bowtie x}[\mathsf{F}^\star\phi]$, $\mathcal{G}$)
 2:     if $\bowtie \in \{\leq, <\}$ then
 3:         $C := \Pi \setminus C$
 4:     end if
 5:     if $\star = c$ then
 6:         ModelCheckRewFc($\langle\langle C\rangle\rangle \mathsf{R}^r_{\bowtie x}[\mathsf{F}^c\phi_1]$, $\mathcal{G}$)
 7:     else if $\star = \infty$ then
 8:         ModelCheckRewFinf($\langle\langle C\rangle\rangle \mathsf{R}^r_{\bowtie x}[\mathsf{F}^\infty\phi_1]$, $\mathcal{G}$)
 9:     else if $\star = 0$ then
10:         ModelCheckRewF0($\langle\langle C\rangle\rangle \mathsf{R}^r_{\bowtie x}[\mathsf{F}^0\phi_1]$, $\mathcal{G}$)
11:     end if
12: end procedure
```

the problem into a "maximisation" one, as described earlier for the $\langle\langle C\rangle\rangle \mathsf{P}_{\bowtie p}[\psi]$ operator.

Algorithm 5 gives a procedure to compute the set of states satisfying the formula of the form $\langle\langle C\rangle\rangle \mathsf{R}^r_{\bowtie x}[\mathsf{F}^c\phi]$ based on the fixpoint equations given in Section 4.2.3. In order to avoid explicitly computing the states that have infinite expected reward, we do this indirectly by performing convergence check only on states which have not yet been found to satisfy (or not satisfy if $\bowtie \in \{\leq, <\}$) the formula (we need perform this check every $|S|$ iterations to make sure that the progress is reflected properly). We can apply this implementation technique because, for the states, which can accumulate infinite reward value, $X[s]$ becomes greater than x eventually and then we can add it to the set *sat* (or *unsat*) and terminate considering the convergence of its value. For the states, which do not satisfy the bound, convergence check is eventually triggered.

The model checking algorithm of rPATL formulae $\langle\langle C\rangle\rangle \mathsf{R}^r_{\bowtie x}[\mathsf{F}^\infty\phi]$ derived from equations presented in Section 4.2.3 is shown in Algorithm 6. The algorithm begins by computing the set of states for which value is infinity, that definitely satisfy the formula for any bound if $\bowtie \in \{\geq, >\}$, and definitely do not if $\bowtie \in \{\leq, <\}$ for any bound x, and then removes them from the game. For the remaining states, we start by increasing every reward by δ and computing the over-approximation of the values $X[s]$ for all states s. After this has been done we revert back to the original reward structure and compute the actual values via value iteration until the desired convergence level has been reached. Finally, the maximal value is compared against the bound and the states for which the value meets the bound are added to *sat*.

In Algorithm 7 we describe the procedure for model checking the final type of the reward formula, $\langle\langle C\rangle\rangle \mathsf{R}^r_{\bowtie x}[\mathsf{F}^0\phi]$. The algorithm starts by solving a Büchi game (see Section 3.2.4) to identify the set of states for which the expectation is infinite (adds them to *sat* if $\bowtie \in \{\geq, >\}$) and removes them from the game. Then, after computing the bound B,

Algorithm 5 Model checking routine for the F^c reward operator.

```
1: procedure ModelCheckRewFc($\langle\langle C\rangle\rangle \mathsf{R}^r_{\bowtie x}[\mathsf{F}^c \phi_1]$, $\mathcal{G}$)
2:     $T :=$ModelCheck($\phi_1$, $\mathcal{G}$)
3:     $sat := \emptyset$; $unsat := \emptyset$; $X := 0$; $X' := 0$; $iter := 0$
4:     while $true$ do
5:         $X' := X$
6:         for all $s \in S \setminus T$ do
7:             if $s \in \bigcup_{i \in C} S_i$ then
8:                 $X[s] := r(s) + \max_{a \in A(s)} \sum_{t \in S} \Delta(s, a, t) \cdot X'[t]$
9:             else
10:                $X[s] := r(s) + \min_{a \in A(s)} \sum_{t \in s} \Delta(s, a, t) \cdot X'[t]$
11:            end if
12:            if $\bowtie \in \{\geq, >\} \wedge X[s] \bowtie x$ then
13:                $sat := sat \cup \{s\}$
14:            else if $(\bowtie = \leq \wedge X[s] > x) \vee (\bowtie = < \wedge X[s] \geq x)$ then
15:                $unsat := unsat \cup \{s\}$
16:            end if
17:        end for
18:        if $iter > |S|$ then
19:            if $\varepsilon > \max_{s \in S \setminus (sat \cup unsat)} |X[s] - X'[s]|$ then
20:                break
21:            end if
22:            $iter := 0$
23:        end if
24:        $iter := iter + 1$
25:    end while
26:    return $sat$
27: end procedure
```

we compute the values of the reward achieved by the "rich man" strategy, which is played after $\geq B$ of the reward has been accumulated along the path (this is done in Algorithm 8 based on the algorithm described in Section 4.2.3). Then we compute the value using the Equations (4.10) from Section 4.2.3. Note that, for each 'step' of these equations, we need to perform the iteration until the values converge (see lines 16-28); the values converge in one iteration if all rewards are strictly positive in the game, but may need more iterations in the presence of zero rewards.

Finally, Algorithm 9 describes a model checking routine for computing the set of states in which the multi-objective rPATL formula $\langle\langle C\rangle\rangle \mathsf{P}_{=x}[\mathsf{F}\,\phi] \equiv \langle\langle C\rangle\rangle \mathsf{P}_{\geq x}[\mathsf{F}\,\phi] \wedge \langle\langle C\rangle\rangle \mathsf{P}_{\leq x}[\mathsf{F}\,\phi]$ is satisfied.

Algorithm 6 Model checking routine for F^{∞} reward operator.

```
1: procedure ModelCheckRewFinf($\langle\langle C\rangle\rangle\mathsf{R}^{r}_{\bowtie x}[\mathsf{F}^{\infty}\phi_1]$, $\mathcal{G}$)
2:     $T$ :=ModelCheck($\phi_1$, $\mathcal{G}$)
3:     $inf$ :=ModelCheck($\langle\langle C\rangle\rangle\mathsf{P}_{<1}[\top \,\mathsf{U}\, \phi_1]$, $\mathcal{G}$)
4:     $S := S \setminus inf$
5:     if $\bowtie \in \{\geq, >\}$ then
6:         $sat := inf$
7:     else
8:         $sat := \emptyset$
9:     end if
10:    $X := 0$; $X' := 0$
11:    $\delta := 1$
12:    while $true$ do
13:        $X' := X$
14:        for all $s \in S \setminus T$ do
15:            if $s \in \bigcup_{i \in C} S_i$ then
16:                $X[s] := (r(s) + \delta) + \max_{a \in A(s)} \sum_{t \in S} \Delta(s, a, t) \cdot X'[t]$
17:            else
18:                $X[s] := (r(s) + \delta) + \min_{a \in A(s)} \sum_{t \in S} \Delta(s, a, t) \cdot X'[t]$
19:            end if
20:        end for
21:        if $\varepsilon > \max_{s \in S} |X[s] - X'[s]|$ then
22:            if $\delta > 0$ then
23:                $\delta := 0$
24:            else
25:                break
26:            end if
27:        end if
28:    end while
29:    for all $s \in S$ do
30:        if $X[s] \bowtie x$ then
31:            $sat := sat \cup \{s\}$
32:        end if
33:    end for
34:    return $sat$
35: end procedure
```

Algorithm 7 Model checking routine for F^0 reward operator.

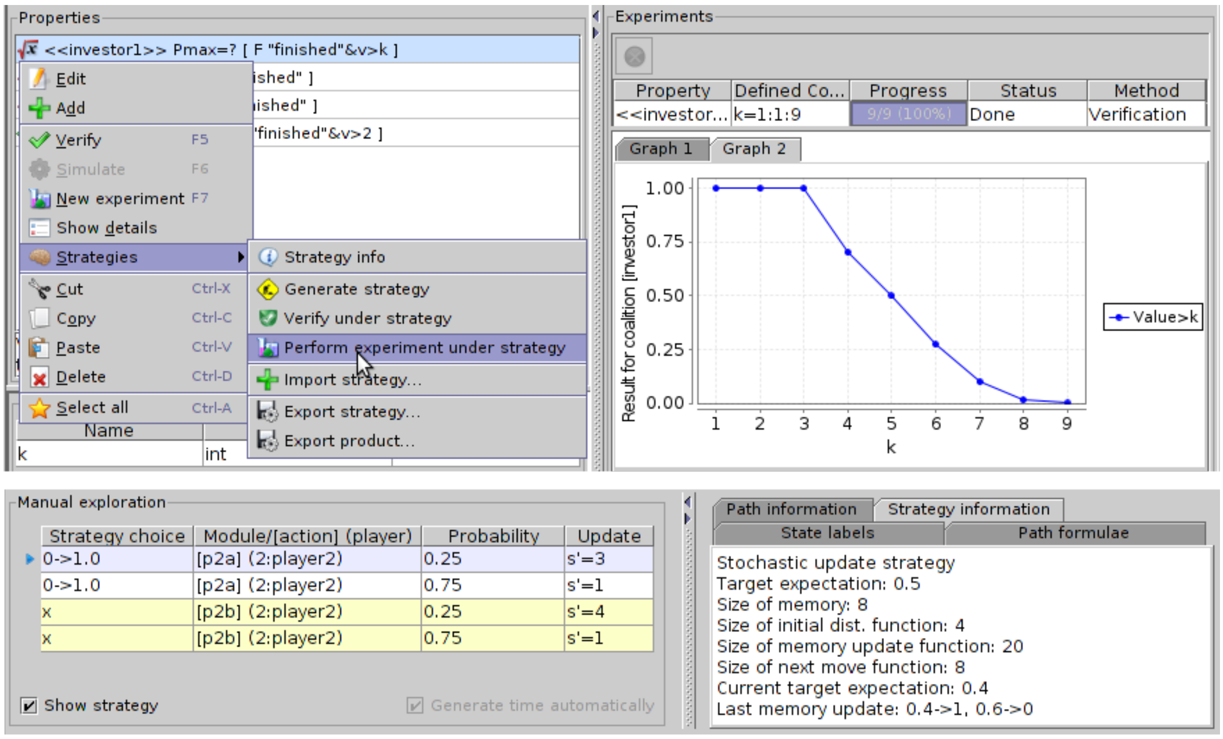
```
procedure ModelCheckRewF0($\langle\langle C\rangle\rangle \mathsf{R}^r_{\bowtie x}[\mathsf{F}^0\phi_1]$, $\mathcal{G}$)
    $T$ :=ModelCheck($\phi_1$, $\mathcal{G}$)
    $inf$ := Büchi($C$, $\mathcal{G}$)
    if $\bowtie\in\{\geq,>\}$ then
        $sat := inf$
    else
        $sat := \emptyset$
    end if
    $S := S \setminus sat$
    $P$ :=ProbReach($C$, $T$, $\mathcal{G}$)
    $C$ :=CumRew($C$, $T$, $\mathcal{G}$)
    $B := \lceil \max_{s\in S} C[s]/(\min_{a,b\in A(s)} \sum_{t\in S}\Delta(s,a,t)\cdot P[t] - \sum_{t\in S}\Delta(s,b,t)\cdot P[t])\rceil$
    $RB$ := ComputeRB($T$, $P$, $\mathcal{G}$)
    for all $i \in \{B,\ldots,B{+}r_{\max}{-}1\}$ do
        for all $s \in S$ do
            $R_i[s] = i + RB[s]$
        end for
    end for
    for $i = B{-}1,\ldots,0$ do
        $R_i := 0$
        while $true$ do
            $R'_i := R_i$
            for all $s \in S$ do
                if $s \in \bigcup_{i\in C} S_i$ then
                    $R_i[s] := i + \max_{a\in A(s)} \sum_{t\in S}\Delta(s,a,t)\cdot R'_{i+r(s)}[t]$
                else
                    $R_i[s] := i + \min_{a\in A(s)} \sum_{t\in S}\Delta(s,a,t)\cdot R'_{i+r(s)}[t]$
                end if
            end for
            if $\varepsilon > \max_{s\in S} |R_i[s] - R'_i[s]|$ then
                break
            end if
        end while
    end for
    for all $s \in S \setminus T$ do
        if $R_0[s] \bowtie x$ then
            $sat := sat \cup \{s\}$
        end if
    end for
    return $sat$
end procedure
```

Algorithm 8 Routine for computing value for R_i for $B \leq i \leq B{+}r_{\max}{-}1$.

```
procedure ComputeRB($T$, $P$, $\mathcal{G}$)
    $X := 0$; $X' := 0$
    while $true$ do
        $X' := X$
        for all $s \in S \setminus T$ do
            if $s \in \bigcup_{i \in C} S_i$ then
                $X[s] := P[s] \cdot r(s) + \max_{a \in A(s)} \sum_{t \in S} \Delta(s,a,t) \cdot X'[t]$
            else
                $X[s] := P[s] \cdot r(s) + \min_{a \in A(s)} \sum_{t \in S} \Delta(s,a,t) \cdot X'[t]$
            end if
        end for
        if $\varepsilon > \max_{s \in S} |X[s] - X'[s]|$ then
            break
        end if
    end while
    return $X$
end procedure
```

Algorithm 9 Model checking routine for exact reachability probability.

```
procedure ModelCheckExact($\langle\langle C \rangle\rangle \mathsf{P}_{=x}[\mathsf{F}\,\phi]$, $\mathcal{G}$)
    $T :=$ModelCheck($\phi$, $\mathcal{G}$)
    $sat := \emptyset$
    $bad := S$
    while $bad \neq \emptyset$ do
        $bad := \emptyset$
        $X_{\min} :=$ ProbReach($\Pi \setminus C$, $T$, $\mathcal{G}$)
        $X_{\max} :=$ ProbReach($C$, $T$, $\mathcal{G}$)
        for all $s \in S$ do
            if $X_{\min} > X_{\max}$ then
                $bad := bad \cup \{s\}$
            end if
        end for
        $S := S \setminus bad$
    end while
    for all $s \in S$ do
        if $X_{\min}[s] \leq x \leq X_{\max}[s]$ then
            $sat := sat \cup \{s\}$
        end if
    end for
    return $sat$
end procedure
```



<<investor1>> Pmax=? [F "finished"&v>k]
ished"]
ished"]
"finished"&v>2]

Name	
k	int

Property	Defined Co...	Progress	Status	Method
<<investor...	k=1:1:9	9/9 (100%)	Done	Verification

Strategy choice	Module/[action] (player)	Probability	Update
0->1.0	[p2a] (2:player2)	0.25	s'=3
0->1.0	[p2a] (2:player2)	0.75	s'=1
x	[p2b] (2:player2)	0.25	s'=4
x	[p2b] (2:player2)	0.75	s'=1

Stochastic update strategy
Target expectation: 0.5
Size of memory: 8
Size of initial dist. function: 4
Size of memory update function: 20
Size of next move function: 8
Current target expectation: 0.4
Last memory update: 0.4->1, 0.6->0

Figure 6.2: PRISM-games screenshots: simulation of a synthesised strategy (bottom) and verification of a property under the strategy (top).

6.4 Strategy synthesis

PRISM-games implements the strategy construction algorithms for rPATL properties described in Section 4.4 and the one for precise probability described in Section 3.4. Strategies can be analysed manually by exploring the choices taken and memory updates in the simulator view. They can also be exported to (and imported from) files for automatic analysis and usage for strategy implementation (see Figure 6.2 for screenshots demonstrating this functionality). PRISM-games also supports application of strategies – the product of a strategy and the original game can be built, resulting in a new model on which other properties can be verified. For example, in a game with three players we can generate a strategy for player 1 achieving $\langle\langle\{1\}\rangle\rangle \mathsf{P}_{\geq 0.5}[\mathsf{F}\,\mathsf{goal1}]$ in the initial state. Implementing this strategy would then result in a two-player game, in which only players 2 and 3 are able to chose their actions and on which further properties may be verified. For example, in rPATL formula $\langle\langle\{2,3\}\rangle\rangle \mathsf{P}_{\geq 0.99}[\mathsf{F}\,\mathsf{goal2}]$, player 1 now does not minimise the probability of reaching a $\mathsf{goal2}$ state; instead its strategy is fixed to one which achieves the first rPATL formula. This functionality is useful when evaluating the generated strategy for other properties. For example, in the case study of user-centric networks that is presented in Section 6.6.3, we have used strategy implementation to compare the distributions of requests submitted to service providers by client strategies achieving optimal service cost in different pricing schemes.

6.5 Experimental results

In this section we present experimental results testing the performance of PRISM-games. The results shown here have been obtained during the analysis of the following case studies:

- MDSM: microgrid demand-side management (see Section 6.6.1);
- CDMSN: collective decision making for sensor networks (see Section 6.6.2);
- UCN: reputation and virtual currency protocol for user-centric networks (see Section 6.6.3);
- TEAM-FORM: team formation protocol (see Section 3.5).

Case study [parameters]		SMG statistics			Model checking		
		Players	States	Transitions	Property type	Constr. (s)	Check (s)
MDSM $[N]$	5	5	743,904	2,145,120	$\langle\langle C\rangle\rangle \mathsf{R}^{r}_{\max=?}[\mathsf{F}^{\infty}\phi]$	14.5	61.9
	6	6	2,384,369	7,260,756		55.0	221.7
	7	7	6,241,312	19,678,246		210.7	1,054.8
CDMSN $[N]$	3	3	1,240	6,240	$\langle\langle C\rangle\rangle \mathsf{P}_{\bowtie q}[\,\mathsf{F}^{\leq k}\,\phi]$	0.2	0.2
	4	4	11,645	73,948		0.8	0.8
	5	5	100,032	760,430		3.2	6.4
UCN $[K]$	5	4	5,737	9,121	$\langle\langle C\rangle\rangle \mathsf{R}^{r}_{\max=?}[\mathsf{F}^{c}\phi]$	0.163	0.2
	10	4	78,516	129,798		2.4	4.4
	20	4	718,499	1,227,391		14.9	98.4
TEAM-FORM $[N]$	3	3	12,475	15,228	$\langle\langle C\rangle\rangle \mathsf{P}_{\max=?}[\mathsf{F}\,\phi]$	0.8	0.2
	4	4	96,665	116,464		1.6	0.9
	5	5	907,993	1,084,752		13.6	11.2

Table 6.1: Performance statistics for PRISM-games.

Table 6.1 shows model statistics (number of players, states and transitions) for three SMGs taken from each case study. It also gives the execution time for model checking a sample property on each one, consisting of the time for model construction (building an SMG from the modelling language description) and for executing the model checking algorithms, described earlier in the chapter. Experiments were run on a 2.80GHz PC with 32GB RAM. The details of the PRISM-games models can be found in Appendix F.

As we can see from the table, the largest model that we analysed using PRISM-games has about 6 million states and 20 million transitions. Due to the use of the explicit representation, storing such game requires substantial amount of memory, about 20Gb for the MDSM model with 7 households. The actual model construction and model checking times are less of a bottleneck.

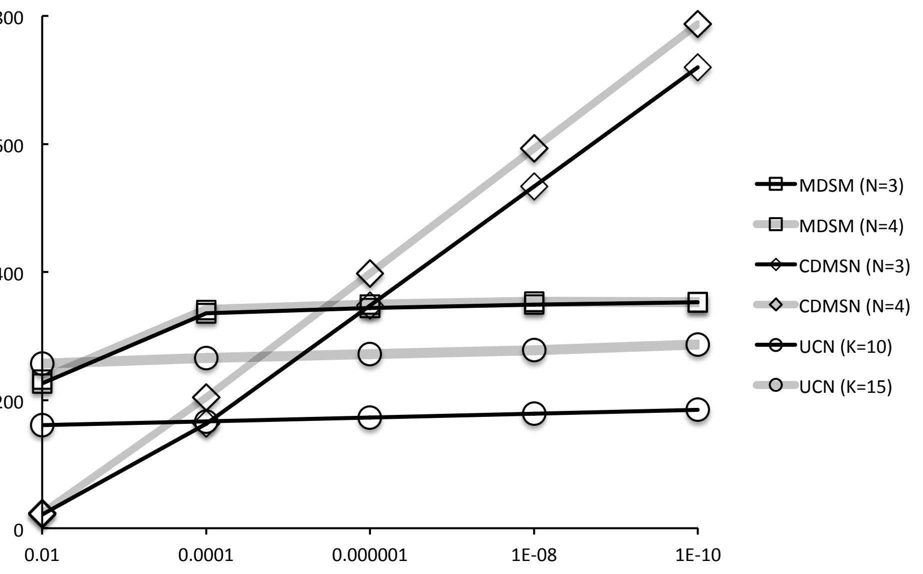


Figure 6.3: Performance of numerical computation algorithms (number of iterations) for varying convergence thresholds ε using absolute difference convergence check.

We also discuss how the convergence test and its threshold affect the convergence of model checking algorithms. As mentioned in Section 6.3, our algorithms are mostly based on the evaluation of numerical fixpoints, termination of which is decided using a simple convergence test. More precisely, if X_s^k denotes the value computed for a state s in iteration k and ε represents a pre-specified *convergence threshold*, the computation terminates when the maximum *absolute* difference between values for successive iterations falls below ε, i.e., when:

$$\max_{s\in S} |X_s^k - X_s^{k-1}| < \varepsilon$$

Figure 6.3 illustrates, for several of the models from Table 6.1, how the total number of iterations of numerical computation required varies for different values of the convergence threshold ε. For the CDMSN example, we check a property of the form $\langle\langle C\rangle\rangle\mathsf{P}_{\bowtie q}[\mathsf{F}\,\phi]$ since the bounded property used in Table 6.1 always requires exactly k iterations to converge. We do not include the results for the team formation example since the corresponding SMG is a tree fixed depth and hence computation always terminates after the same number of steps. From the plots in Figure 6.3, we can see that varying $\varepsilon = 10^{-n}$ for increasing values of n results in, for the case studies studied here, at most a linear increase in the number of iterations required by the method to converge, but the shape of the curve is model dependent. Also note that, for all models, changing the model size does not affect the shape of the curve, just the slope and the magnitude. For the MDSM and UCN protocols, the number of iterations required to satisfy the convergence check stabilises after reaching $\varepsilon = 10^{-4}$, whereas for CDMSN it keeps increasing with precision.

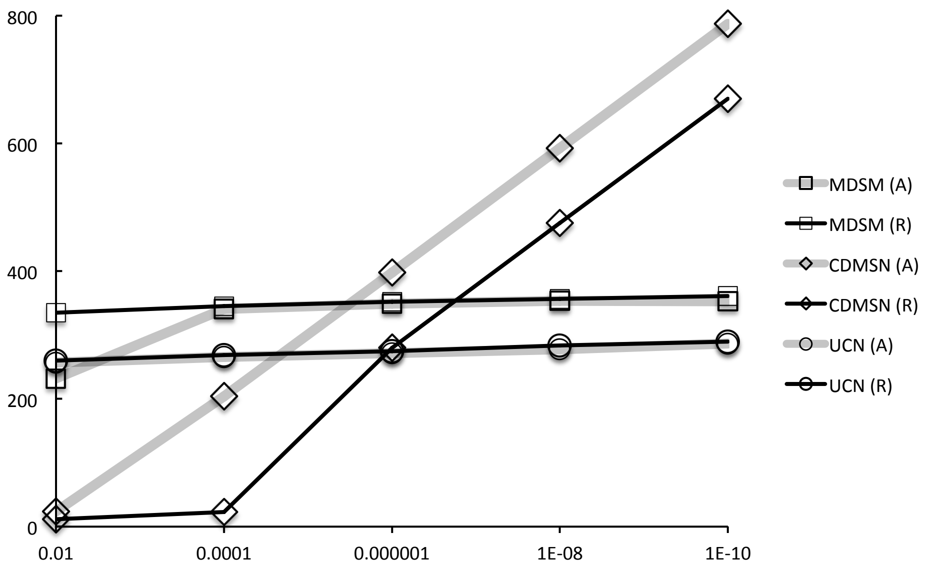


Figure 6.4: Comparison of absolute and relative termination criteria for value iteration algorithms for different convergence thresholds ε. (A) stands for absolute and (R) stands for relative convergence checks.

There are other variants of convergence checking used in practice. One such check is relative difference check. For a given convergence threshold ε the value iteration is terminated if we have that

$$\max_{s \in S} \frac{|X_s^k - X_s^{k-1}|}{|X_s^k|} < \varepsilon.$$

Figure 6.4 shows the comparison of the performance between the absolute and relative convergence checks. The models used were MDSM ($N = 4$), CDMSN ($N = 4$) and UCN ($K = 15$). For the UCN case study there is almost no difference in the number of iterations performed, whereas for CDMSN model, there is a significant improvement in convergence time, which is particularly apparent for lower values of ε. The converse is true for the MDSM model, where the absolute termination criterion requires less iterations to converge than the relative one.

Discussion. In this section we have presented the performance statistics for the PRISM-games model checker. We have also discussed convergence checks for two difference termination criteria for value iteration. In the case studies that we analysed, varying the order of magnitude of the convergence threshold ε required linear increase in the number of iteration required by the algorithm to converge, suggesting that higher precision can be obtained relatively cheaply. These results regarding the termination have to be interpreted with care, because neither of the convergence checks used here guarantees the correctness of the result of the algorithm.

6.6 Case studies

In this section we present the analysis of the three case studies of algorithms for competitive stochastic systems that we have analysed using PRISM-games. We start by presenting the analysis of the microgrid demand-side management protocol of [68] for which we use rPATL model checking functionality of PRISM-games to identify a potential flaw in the algorithm, then we present the performance and robustness analysis of the algorithm for collective decision making in sensor networks developed by [99], and finally, we use a combination of rPATL model checking and strategy synthesis functionality to analyse of the reputation and virtual currency mechanisms for user-centric networks that were introduced in [15].

The case studies presented in Sections 6.6.1 and 6.6.2 have been published in [36, 37]. The case study from Section 6.6.3 has been published in [81]. The analyses presented here are the extended versions of the above publications.

6.6.1 Microgrid demand-side management

Microgrid is an increasingly popular (distributed) model for future energy markets where neighbourhoods use electricity generation from local sources (e.g., wind/solar power) to satisfy local demand. Microgrid environment is highly decentralised and users have a significant control over the infrastructure, and hence the success of such system is heavily dependent on the active demand management and cooperation between its users (*demand-side management*). One of the key requirements for the infrastructure is the ability to incentivise cooperation and discourage abuse and selfish behaviour. Such systems are usually analysed using simulation studies, but these approaches can fail to uncover important features or weaknesses of the models. In this case study, we use rPATL model checking to analyse the microgrid demand-side management (MDSM) infrastructure of [68] and identify an important incentive-related weakness.

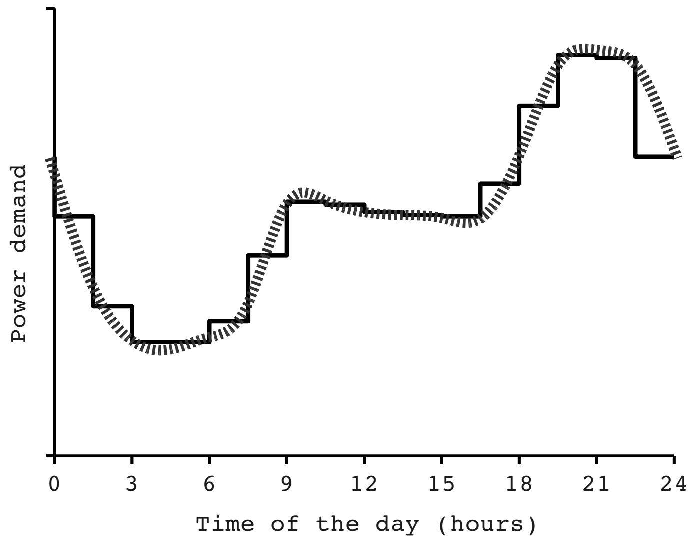


Figure 6.5: MDSM energy demand curve from [68] and its piecewise approximation.

The algorithm. The system in [68] consists of N *households* connected to a single *distribution manager* (DM). At every time-step, the DM randomly contacts a household that, if needed, can submit a load for execution. Each load has an energy demand that is required for its execution. The probability of the generating a load at each time-step is determined by a daily demand curve from [68] (see Figure 6.5). The duration of a load is random, between 1 and D time-steps, where D is a model parameter. The cost of executing a load for a single step is the number of tasks currently running. Hence, the total cost for the whole system increases quadratically with households executing more loads in a single step.

Each household follows a very simple algorithm, the essence of which is that, when it is scheduled to act and has a load to execute, if the cost is below an agreed limit c_{lim}, it executes the load, and otherwise it only does so with a pre-agreed probability P_{start}. In [68], the *value* for each household in a time-step is measured by $V{=}\frac{\text{loads executing}}{\text{cost of execution}}$ and it is shown (through simulations) that, *provided every household conforms to this algorithm*, the peak demand and the total cost of energy to the system are reduced significantly while still providing a good (expected) value V for each household.

Modelling and analysis. We model the system as an SMG where each household is represented by a player (see Appendix F.2 for the details of PRISM-games models). We vary the number of households (N) from 2 to 7. The size of the underlying SMG model is exponential in N (see Table 6.2 for the experimental results) and so we are unable to analyse systems with more than 7 households. Other parameters that we use are $D{=}4$

Number of households	States	Transitions	Construction time (s)
2	5,302	8,328	0.5
3	33,528	82,560	1.2
4	178,272	473,088	3.1
5	743,904	2,145,120	16.6
6	2,384,369	7,260,756	61.2
7	6,241,312	19,678,246	234.8

Table 6.2: SMG model sizes and construction times.

and c_{lim}=1.5. We analyse a period of 3 days, each consisting of 16 time-steps. The daily demand is generated using a piecewise approximation of the daily demand curve, shown in Figure 6.5.

First, as a benchmark, assume that all households follow the algorithm of [68], i.e., the strategies of all players are fixed and the model is a DTMC. Define reward structure r_i for the value for household i at each step so that for all $s \in S$ $r_i(s) = \frac{\text{household load}}{\text{cost of execution}}$ (where 'household load' can be 0 or 1 and the cost of execution is the total number of jobs running in the system). For the coalition C of players the reward is defined in a similar way, $r_C = \frac{\text{loads executed by coalition}}{\text{cost of execution}}$ (where the cost of execution is the total number of jobs running in the system multiplied by the number of loads executed by the coalition) be the total reward for a set of households C.

To compute the expected value per household, we use the rPATL query:

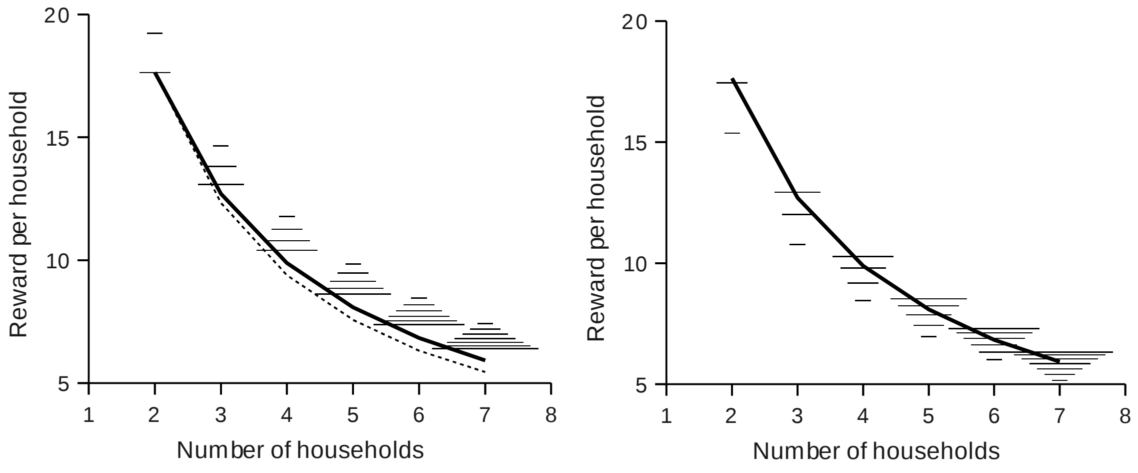
$$\frac{1}{|C|} \cdot \langle\langle C \rangle\rangle \mathsf{R}^{r_C}_{\max=?}[\,\mathsf{F}^0\,\mathsf{time{=}max\_time}\,],$$

where $\mathsf{time{=}max\_time}$ is an atomic proposition labelling the terminal states of the protocol. Initially, fix C to be the set Π of all N players (households) and use DTMC model checking to determine the optimal value of P_{start} (up to precision 0.05) achievable by a memoryless strategy for each player. The values for games of different sizes are shown in Table 6.3. The model checking results are shown by the bold lines in Figure 6.6. We also plot (as a dotted line) the values obtained if no demand-side management is applied (i.e., $P_{\text{start}} = 1$).

SMG size (N)	2	3	4	5	6	7
P_{start} value	1.0	0.8	0.7	0.65	0.6	0.6

Table 6.3: Optimal values for P_{start} parameter.

Next, we consider the situation where the set of households C is allowed to deviate from the pre-agreed strategy, by choosing to ignore the limit c_{lim} if they wish. We check the same rPATL query as above, but now varying C to be coalitions of different sizes, $C \in \{\{1\}, \{1, 2\}, \ldots, \Pi\}$. The resulting values are plotted in Figure 6.6a, shown as horizontal



(a) Original version. (b) Version with punishment.

Figure 6.6: Expected value per household for MDSM. The bold line shows all households following the algorithm of [68]; the dotted line shows the case without DSM. Horizontal dashes show deviations by collaborations of increasing size (shortest dash: individual deviation; longest dash: deviation of all households).

dashes of width proportional to the size of C: the shortest dash represents the deviation of a single household and the longest is a collaboration of all households in the coalition. The former shows the maximum value that can be achieved by following the optimal collaborative strategy, and in presents a benchmark for the performance of the original algorithm. The key result is that deviations by individuals or small coalitions guarantee a better expected value for the households than *any* larger collaboration. This is a highly undesired weakness for an MDSM system, because it essentially shows that users can ignore the proposed demand-side management method.

Changing the incentives. We propose a simple punishment mechanism that addresses the problem: we allow the distribution manager to cancel *one* job per step if the cost exceeds c_{lim}. The intuition is that, if a household is constantly abusing the system, its job could be cancelled. Results for the same set of rPATL queries on a revised model that incorporates the punishment mechanism are shown in Figure 6.6b, where all households outside the coalition follow the original algorithm. We see that the modification of the algorithm inverts the incentives. The best option now is full collaboration and small coalitions who deviate *cannot* guarantee better expected values any more.

Discussion. In this case study we demonstrated how game-based modelling can be applied to the analysis of a decentralised protocol for MDSM. Using rPATL model checking we identified a problem with the incentive design of the pricing/value scheme, which discourages cooperation between different households. We also proposed a simple 'policing' mechanism, which inverts the incentives and encourages cooperation. The size of the SMG increases exponentially with the number of households, and even though we could only analyse models with up to seven households, we have managed to identify clear patterns of behaviour of the system and explore how they change when adding new features to the model.

6.6.2 Collective decision making for sensor networks

Sensor networks are composed of a set of low-power, autonomous devices, which often must act collaboratively in order to achieve a particular goal. Formal analysis of such systems can help to establish performance boundaries (e.g., find the best value achievable by the sensor network assuming full collaboration) and analyse robustness (e.g., look at the performance of the system in the presence of failure or unexpected behaviour of some nodes). In this case study, we illustrate the use of rPATL model checking to analyse a distributed consensus algorithm for sensor networks [99].

The algorithm. There are N sensors deployed in the environment having a set of targets $K = \{k_1, k_2, \dots\}$. Each of the targets has a *quality* parameter $Q_{k_j} \in [0,1]$. The goal is for the sensors to agree on a target with the maximum quality Q_{k_j}. This is quite an abstract model and the target can be used for anything from locating an intruder in a security system to RoboBees[1] trying to select the best nesting site.

Each sensor i stores a *preferred* target $p_i \in K$, its quality Q_{p_i} and an integer $l_i \in \{1, \dots, L\}$ to represent *confidence* in the preference. A sensor has three actions: *sleep*, *explore* and *communicate*. As proposed by [99], each sensor repeatedly *sleeps* for a random time t and then either *explores* (with probability P_{exp}) or *communicates*. For the *explore* action, sensor i picks a target $k \in K$ uniformly at random and, with probability $P_k = Q_k^\eta/(Q_k^\eta + Q_{p_i}^\eta)$, switches its preference (p_i) to k and resets confidence to 1. The intuition behind the formula is that the bigger the difference between qualities of targets, the more likely it is that the sensor chooses the better one, with parameter η amplifying this effect. To *communicate*, sensor compares its preference with that of another sensor chosen at uniformly at random j. If they agree (i.e., both prefer the same target), confidence levels of both sensors are increased. If not, with probability

$$P_{\text{s}} = \frac{Q_{p_j}^\lambda l_j^\gamma}{Q_{p_j}^\lambda l_j^\gamma + Q_{p_i}^\lambda l_i^\gamma},$$

[1]For an in-depth discussion of RoboBees see `http://robobees.seas.harvard.edu`.

sensor i switches its preference to the target preferred by sensor j. The confidence level of i is reset to 1 and the confidence of sensor j is increased. With probability $1-P_s$, the roles of sensors i and j are swapped, i.e., sensor j switches preference to p_i, resets confidence to 1 and the confidence of sensor i is increased. This mechanism effectively conducts a tournament between two sensors with conflicting targets and the one, which has higher confidence and/or quality of its preferred target, wins with higher probability. The key property is that after the tournament both sensors prefer the same target. Parameters γ and λ represent the influence of confidence in and a quality of the target, respectively.

The purpose of the system is to locate and agree upon the target having the best quality by striking a balance between exploration of the environment (i.e., sensing) and communication with other sensors. Intuitively, a 'good' strategy for sensors should be to actively communicate when they believe they have found the best quality target and to explore otherwise. An example of an 'bad' strategy could be the one, which actively communicates to advertise a low quality target, thus not only polluting the system with false information, but also draining the resources of the sensor network.

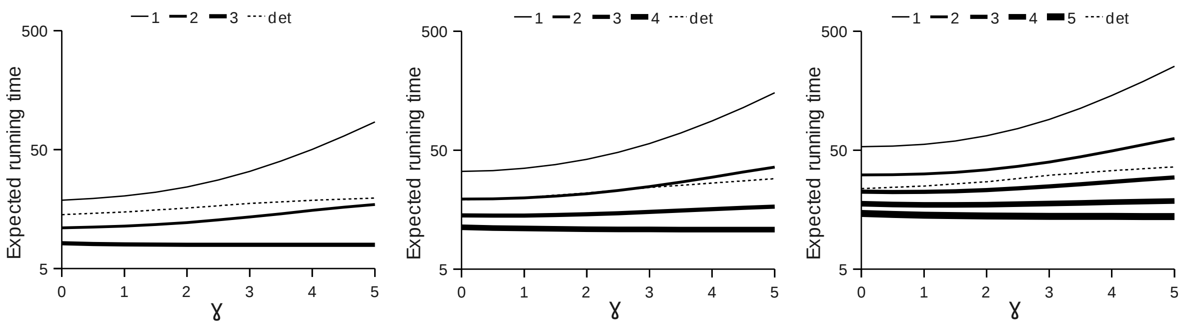
Modelling and analysis. We model each sensor in the system as a player in an SMG (see Appendix F.3 for the details of PRISM-games models). We consider models with number of players $N{=}3, 4, 5$, three targets $K{=}\{k_1, k_2, k_3\}$ with qualities $Q_{k_1}{=}1$, $Q_{k_2}{=}0.5$, $Q_{k_3}{=}0.25$ and two confidence levels $l_i \in \{1, 2\}$. Table 6.4 shows SMG model sizes for systems of different size. As in [99], we assume a random scheduling in turns and fix the

Number of sensors	States	Transitions	Construction time (s)
2	111	466	0.05
3	1,240	6,240	0.2
4	11,645	73,948	0.8
5	100,032	760,430	3.2

Table 6.4: SMG model sizes and construction times.

parameters $\eta{=}1$ and $\lambda{=}1$. In [99], two key properties of the algorithm are studied using simulation: *speed of convergence* and *robustness*. We use rPATL model checking to evaluate both of these properties and explore the values that can be obtained by sensor when they are allowed to execute any action when active. We also assume that only a subset C of the sensors are under our control, and use rPATL queries to optimise performance under the worst-case behaviour assumption about the other sensors (modelling hostility or faults).

First, we study the *speed of convergence* and the influence of parameter γ upon it. In [99], it is shown that increasing γ improves the speed of convergence to *a decision* and



(a) $N = 3$ (b) $N = 4$ (c) $N = 5$

Figure 6.7: Expected running time until the selection of the best quality target for different models and increasing sizes of coalition C. Dotted lines show optimal performance that can be achieved using the original algorithm from [99].

stability of it. Figure 6.7 shows the expected running time to reach the *best decision* (i.e. select k_1) for various values of γ and sizes of the coalition C. We use the reward structure: $r(s) = 1$ for all $s \in S$ and rPATL query:

$$\langle\langle C \rangle\rangle \mathsf{R}^{r}_{\min=?}[\, \mathsf{F}^{\infty} \textstyle\bigwedge_{i=1}^{|\Pi|} \mathsf{p_i{=}k_1} \,] \,.$$

where $C \in \{\{1\}, \{1,2\}, \ldots, \Pi\}$. Figure 6.7 also shows the performance of the original algorithm (line 'det'). We make several important observations. First, in the scenario we analysed, we obtain the opposite effect of γ on the convergence than the one shown in [99], where authors studied the convergence to *a* decision. This result is expected, as increasing γ while keeping λ constant increases the relative importance of the preference against the quality of the site, and thus sensors converge to a (possibly sub-optimal) decision. Second, observe that, if we lose control of a few sensors (e.g., because a fault occurs), we can still guarantee a good convergence time, indicating the fault tolerance potential of the system.

Secondly, we consider *robustness*: the ability of the coalition C to recover from a 'bad' state to a 'good' state in n steps; this property can be specified by rPATL formula $\langle\langle C \rangle\rangle \mathsf{P}_{\max=?}[\, \mathsf{F}^{\leq n}\, \phi_{good}]$, where ϕ_{good} represents 'good' states. The definition of a 'bad' state that we use is the state in which all sensors have a preference for the lowest quality target, i.e., $\bigwedge_{i=1}^{|\Pi|} p_i = k_3$. The definition of a 'good' state that we adopt here is that "there exists a strategy for coalition C to make all sensors, *with probability* > 0.9*, select* k_1 *within 10 steps*". So robustness in rPATL is:

$$\langle\langle C \rangle\rangle \mathsf{P}_{\max=?}[\, \mathsf{F}^{\leq n}\, \langle\langle C \rangle\rangle \mathsf{P}_{>0.9}[\, \mathsf{F}^{\leq 10} \textstyle\bigwedge_{i=1}^{|\Pi|} \mathsf{p_i{=}k_1}]] \,.$$

Figure 6.8 shows a range of values over time span $n = 1, \ldots, 100$, the *worst-case* (mini-

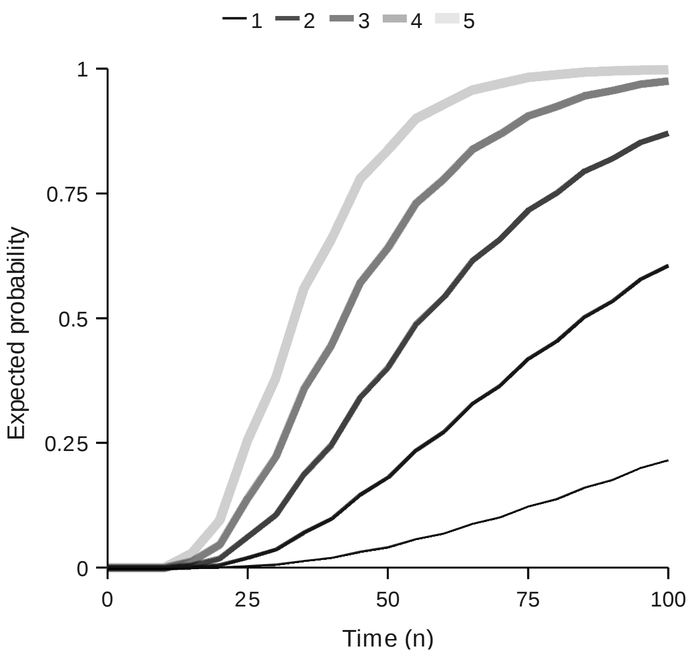


Figure 6.8: Probability to select k_1 within 10 steps is greater than 0.9, with $\gamma = 2$.

mum) value for the rPATL query from all possible 'bad states'. The results are intuitive: the larger the coalition, the better its capability to recover. Also, as in Figure 6.7, there is little use of controlling only one sensor, but the control of at least two provides a significant improvement. It can also be seen that, for the failure of one sensor does not have a significant impact on performance, but the failure of more than half has.

Discussion. In this case study we have demonstrated how rPATL can used to analyse performance and robustness of the sensor network protocol. Note that the results detailed here are in a complete-information setting, which implicitly assumes that the members of the coalition (i.e., non-faulty sensors) have knowledge of which sensors are faulty and can adjust their behaviour accordingly, and therefore the values provided by our analysis are upper bounds on the performance that can be achieved.

6.6.3 Reputation protocol for user-centric networks

In our final case study we present an analysis of the protocol for user-centric networks. The analysis presented here differs from the previous two case studies in that, in addition to model checking, we use strategy synthesis to analyse the protocol.

User-centric networks are designed to encourage users to act cooperatively by sharing resources and/or services between themselves, for example, in order to provide connectivity in a mobile ad-hoc network. The effectiveness of such networks is heavily dependent on their cooperation mechanisms, i.e., a scheme, which defines the cooperation rules and provides incentives for unselfish behaviour. We analyse a cooperation mechanism for user-centric networks [15], which combines a reputation-based incentive, used to establish a measure of trust between users, and a virtual currency mechanism that uses the trust measures to provide prices for services.

Previously, this cooperation model has been analysed formally using probabilistic model checking by [1, 2] using PCTL verification on DTMCs and MDPs. Here we take a different approach and study the cooperation mechanism using strategy-based analysis using stochastic games. The system is modelled as an SMG, in which service providers and requesters are modelled as players. We model objectives of players using rPATL and perform strategy synthesis using PRISM-games to discover important insights into the model.

The cooperation mechanism. The basic ideas behind the cooperation mechanism of [15] can be summarised as follows. We assume a general model of *providers* offering *services* to *requesters*. Cooperation between users is managed through a combination of reputation and virtual currency mechanisms.

Reputation is represented as a (discrete) *trust measure*, denoted trust_{ij}. This measure represents the extent to which user i trusts user j, based on previous interactions between them and the recommendations provided by the other users in the network. A *trust level* T_{ij} is computed as a weighted sum of trust measure and the recommendations provided by other users in the following way:

$$T_{ij} = \alpha \cdot \mathit{trust}_{ij} + (1-\alpha) \cdot \mathit{recs}_{ij},$$

where recs_{ij} is recommendation – an *indirect* trust measure, taken as the average value of trust_{kj} for other users (we call trust_{ij} a *direct* measure of trust). Provider i decides to accept the request from user j if T_{ij} is above or equal to the *service trust level*, denoted st_i (a fixed parameter). The parameter $\alpha \in [0, 1]$ controls the relative influence that the direct and indirect measures of trust have on this decision.

A virtual currency system, which determines the prices of services, is defined as follows. Minimum and maximum costs C_{min}, C_{max} and threshold T' are parameters of the model and the cost is defined as

$$C(\mathit{trust}_{ij}) = \begin{cases} C_{min} + \frac{C_{max}-C_{min}}{T'} \cdot (T' - \mathit{trust}_{ij}) & \text{if } \mathit{trust}_{ij} < T' \\ C_{min} & \text{if } \mathit{trust}_{ij} \geq T' \end{cases}$$

Service provision happens as follows. A requester j chooses a provider i and submits a request. If $T_{ij} \geq st_i$, then the request is accepted. In this case, the two users then negotiate the service cost, using the function of $trust_{ij}$ given above. The negotiation may fail with probability c_i (this may correspond, for example, to the user cancelling the accepted request). This probability represents the cooperative attitude [1] of the provider i. If negotiation succeeds, the service is delivered and the requester has to decide whether or not to pay the provider. If payment is made, the provider increases the trust measure of the requester by one unit. If not, the measure is decreased by td_i units (also a parameter of the model).

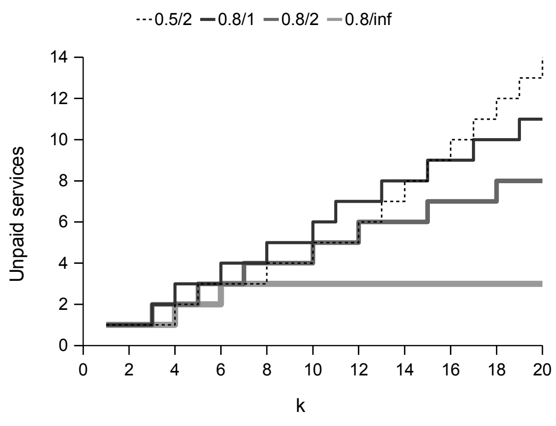
Modelling. We developed the SMG model of the cooperation mechanism of [15], taking the PRISM model of [1] as a starting point (see Appendix F.4 for the details of PRISM-games models). We adopt the same basic network configuration as used in the original analysis of the protocol [1], which comprises 3 providers and 1 requester, each represented as a player in the SMG. The parameters of the cooperation mechanism are also taken from [1] and are as follows. The trust measure $trust_{kj}$ is an integer in the range 0 to 10 and is initially 5. We use α=0.8, unless stated otherwise, and the service trust threshold st_i is set to 5 for all providers. We use a negotiation failure probability of c_i=0.05 for all providers i, and the parameters used to compute prices are fixed at C_{min}=2, C_{max}=10 and threshold T'=8. The statistics for the models constructed in PRISM-games are shown in Table 6.5. The parameter K is the upper bound on the services that can be delivered in the network.

K	States	Transitions	Construction time (s)
4	2,517	3,925	0.1
8	33,808	55,296	0.6
12	150,088	250,625	2.6
16	377,626	639,673	7.4
20	718,499	1,227,391	14.9

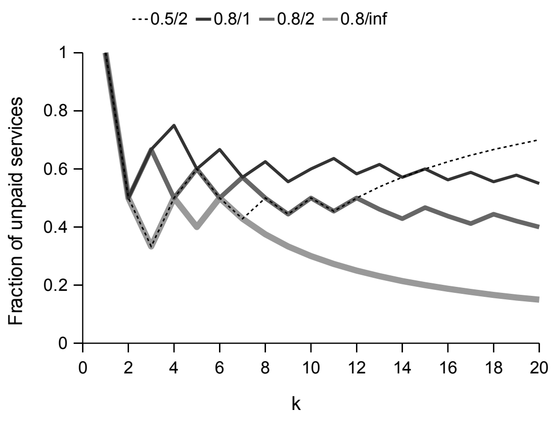
Table 6.5: SMG model sizes and construction times for different values of K (the upper bound on the services delivered in the network).

Unpaid requests. First, we consider the extent to which the requester can obtain services without paying for them. We analyse the maximum (expected) number of unpaid services that the requester can obtain if its goal is to get k services in total. This is expressed in rPATL as:

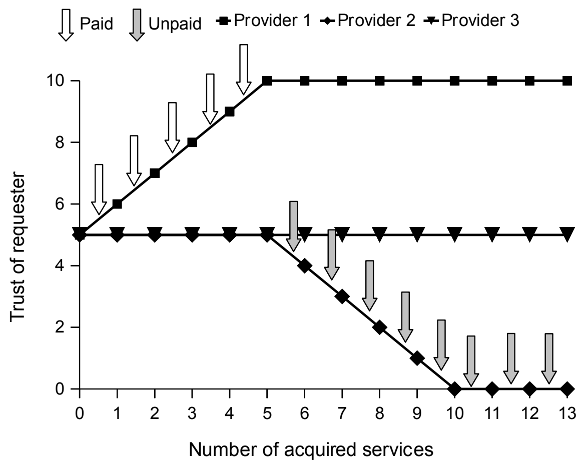
$$\langle\langle\{requester\}\rangle\rangle \mathsf{R}^{unpaid}_{\max=?}[\mathsf{F}^{c}\mathsf{services}{=}\mathsf{k}]$$



(a) *Number* of unpaid services.



(b) *Fraction* of unpaid services.



(c) Strategy for 0.5/2 and $k = 13$.

Figure 6.9: Maximum unpaid services the requester can achieve in obtaining k services.

where *unpaid* denotes a *reward structure* assigning 1 to every unpaid satisfied request. The results for various combinations of model parameters α and td_i are shown in Figure 6.9 (we use 0.5/2 to indicate that $\alpha = 0.5$ and trust is decreased by $td_i = 2$ units upon an unpaid service; $td_i = inf$ means that trust is reset to 0 upon an unpaid service).

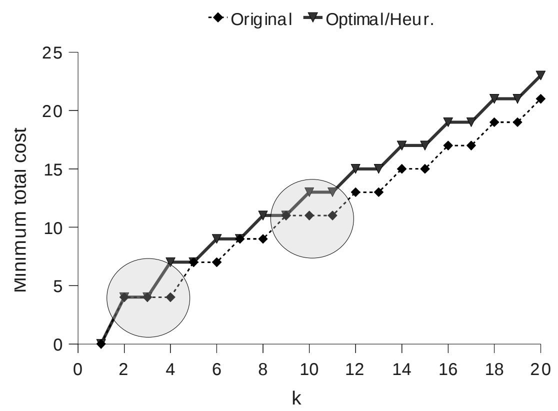
Figures 6.9a and 6.9b show the number and fraction, respectively, of services that are unpaid, for a range of k. From Figure 6.9b, in particular, we see that, for parameters 0.5/2 and $0.8/inf$, the behaviour is fundamentally different from the other two - the portion of requests converges to 1 and 0, respectively. For $0.8/inf$, this behaviour is expected, because the trust measure is decreased to 0 upon non-payment; however, the behaviour of 0.5/2 represents an attack on the trust model allowing the requester to receive an unlimited number of unpaid services for a fixed cost. We synthesise an attacker (requester) strategy for our model with 3 providers, for the case of acquiring $k = 13$ services: for a cost of 5 services, the requester can get an unlimited number of unpaid services. We depict the strategy in Figure 6.9c. Arrows represent 'request-and-pay' (white arrow) and 'request-and-not-pay' (grey arrow) actions of the optimal requester strategy, depending on the number of services acquired so far.

This attack is possible if $st_i \leq (1-\alpha) \cdot T_{\max}$ for some provider i, where $T_{\max}$ is the maximum trust level among all providers. We note that it is only viable if the network is sufficiently small since the fixed cost increases with the number of providers sharing the trust information: to achieve the required indirect trust measure $recs_{ij} \geq \frac{st_i}{1-\alpha}$, the requester must pay for a number of services proportional to the number of providers. However, in order to work, this requires that all providers share their initial direct trust measure even though they have not encountered the requester.

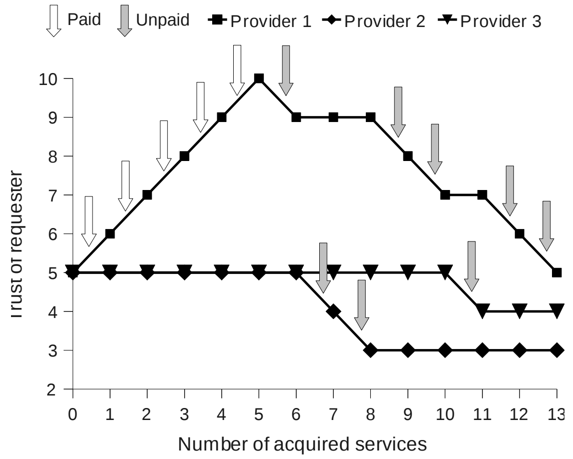
Cost of obtaining services. We now turn our attention to the virtual currency system, and study the minimum price at which the requester can buy k services. For this, we use rPATL formula:

$$\langle\langle\{requester\}\rangle\rangle \mathsf{R}^{cost}_{\min=?}[\mathsf{F}^{\infty}\mathsf{services{=}k}].$$

Intuitively, the requester has a strategy to get one unpaid service for each paid service by executing the following sequence: pay, not pay, pay, not pay, etc. However, a plot of the above property (see highlighted sections of line 'Original' in Figure 6.10a), shows deviations from this pattern, where the requester can get 4 services for the price of 2 and, similarly, 11 services for the price of 9.



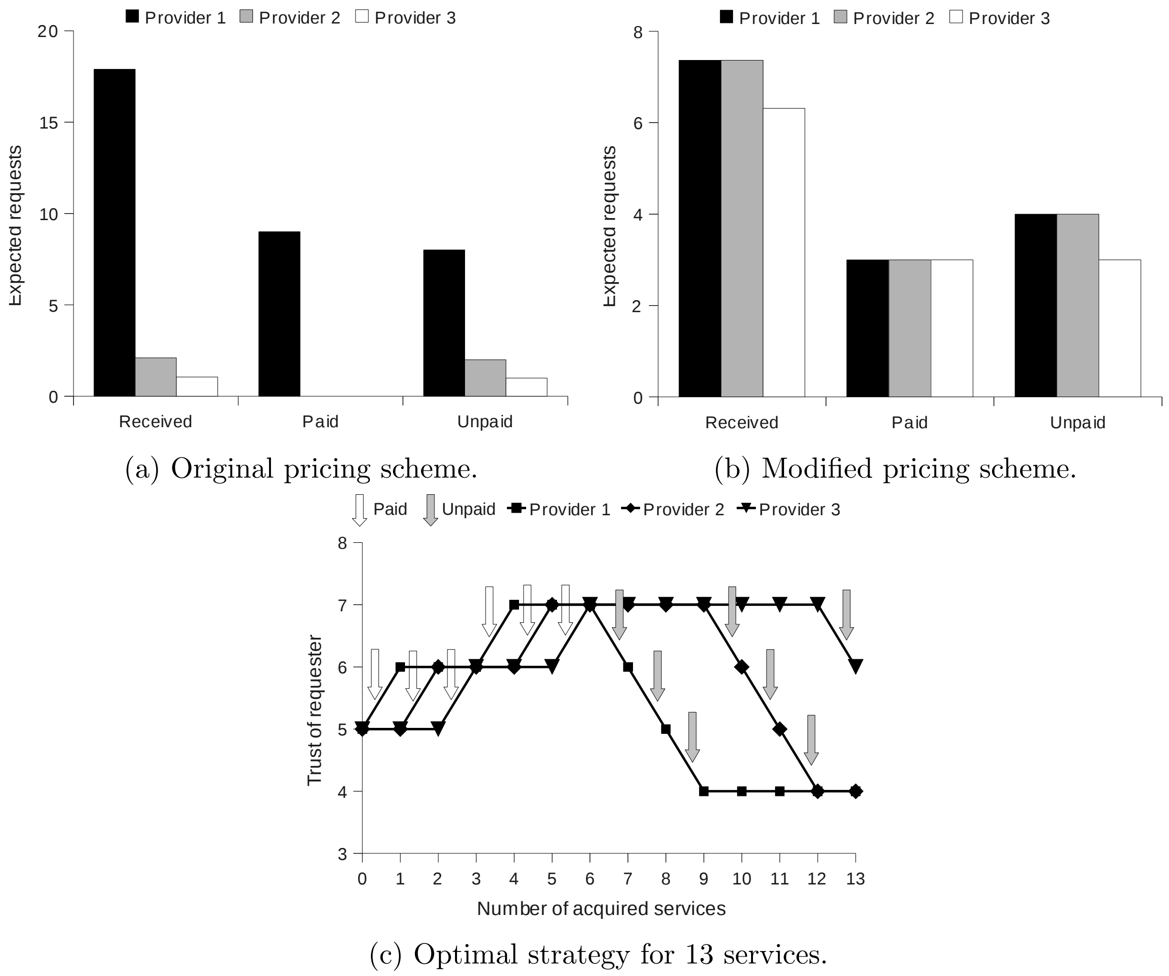
(a) Minimum cost to obtain k services.



(b) Example strategy for $k = 13$.

Figure 6.10: Cost of k services for requester and a strategy example.

We synthesise a strategy achieving this and depict it in Figure 6.10b. We can see that all paid requests are directed to one provider and the others only receive unpaid requests. In fact, by exploiting the reputation system, the requester is even able to obtain 2 unpaid



(a) Original pricing scheme. (b) Modified pricing scheme.

(c) Optimal strategy for 13 services.

Figure 6.11: Distribution of requests among providers.

requests from provider 2.

Next, we devise a fix by changing the model to allow providers to manage the way they share trust information between themselves: they can choose whether to share trust information after interaction with the requester. We synthesise the optimal trust information sharing strategy for cooperating providers, whose behaviour is shown as 'Optimal/Heur.' in Figure 6.10a and can be seen to avoid the above shortfall. Manual examination of the synthesised strategy reveals a suitable heuristic whereby providers share trust information only when its direct trust of the requester is smaller than that of the others. We implement this heuristic in the model and find that it yields the same model checking results as the optimal strategy.

Provider selection incentives. Another interesting feature revealed by the analysis of the strategy provided earlier is that the proposed virtual currency system provides an incentive for the requester to only ever pay for services from one provider (see Figure 6.11a).

This is in fact optimal behaviour because, in the computation of the service cost, only the direct trust measure is used. This may or may not be a desired feature for the mechanism. We can show that a simple change that incorporates the maximum difference between trust into the pricing model (i.e., cost is now computed as $original\_cost + \max_k |trust_{ij} - trust_{kj}|$, where $original\_cost$ is the cost assigned by the pricing scheme) incentivises the requester to disperse its requests between service providers.

Figure 6.11b shows the distribution of requests between providers and Figure 6.11c depicts the actions of the optimal strategy in the new pricing scheme. This strategy contrasts with the strategy for the original mechanism from Figure 6.10b because paid requests are now distributed uniformly across all the service providers. This analysis of strategies has been performed using the 'strategy implementation' feature of PRISM-games, which allows the user to synthesise an optimal player strategy for some rPATL formula, and then evaluate a second rPATL property on the modified SMG in which one coalition's strategy is fixed using the previously synthesised one. In this instance, we used the following rPATL formulae:

$$\langle\langle\{requester\}\rangle\rangle \mathsf{R}^{cost}_{\min=?}[\mathsf{F}^{\infty}\mathsf{services{=}k}] \quad \text{and} \quad \langle\langle\emptyset\rangle\rangle \mathsf{R}^{r}_{\min=?}[\mathsf{F}^{c}\mathsf{services{=}k}]$$

where the first formula was used to synthesise the strategy and the second formula is the one used to analyse it (r represents reward structures for Received, Paid, and Unpaid).

Discussion. In this case study we showed how strategy synthesis can be used in conjunction with model checking to analyse a protocol for user-centric networks. We studied several aspects of the protocol showing how, using rPATL model checking and synthesis on SMGs, we can enrich the results obtained using PCTL on DTMCs and MDPs, which did not allow to study competitive scenarios. Even though the network that we used is relatively small, we were able to captures some of the fundamental aspects of the protocol. For instance, observe that the decision whether to provide a service to a requester does not depend on the trust level of other requesters in the network, so incorporating more requesters does not offer any more information about the dynamics of trust and provided services. On the other hand, using three service providers already allows us to identify malicious strategies for the requester that can be generalised to an arbitrary number of providers.

6.7 Summary

The contributions of his chapter are summarised as follows.

Tool. We have presented PRISM-games, a model checker for stochastic multi-player games implementing rPATL model checking and strategy synthesis techniques developed in this thesis. The tool extends the PRISM model checker with support for stochastic games and rPATL model checking, as well as the strategy synthesis and analysis functionality. The tool is released as open source (under GPL license) and is publicly available to download from `http://www.prismmodelchecker.org/games/`. The summary of the tool functionality and usage instructions is presented in Appendix E.

Algorithms. In this section we have also presented the pseudocode for the rPATL model checking algorithms that were constructed using the results presented in Section 4.2.

Case studies. We have presented three case studies of competitive stochastic systems illustrating the applicability of our methods and tools. We have provided stochastic game models and used rPATL model checking and strategy synthesis functionality to explore the quantitative features of the microgrid demand-side management, collective decision making algorithms and a reputation-based virtual currency mechanism. For these algorithms we have identified several problems/unwanted features and suggested possible improvements.

Discussion. PRISM-games has allowed us to study several protocols for competitive stochastic systems and discover interesting properties and undesired behaviours. However, the scalability that we can achieve is limited in terms of the number of agents when compared to simulation-based approaches for the same protocols. For example, team formation that we studied has been analysed in [62] for systems consisting of 100 agents; the number of agents used in the collective decision algorithm's analysis was 1000 [99] and the microgrid demand-side management algorithm has been studied with agent populations reaching 100,000 [68]. On the other hand, using simulation only allowed authors to study fixed strategies of agents without considering potential optimal behaviour and/or hostility. We have also shown that performing analysis based on strategy synthesis can help to better understand the system and, for example, to generate potential attacks on the protocol as we have done for the user-centric network protocol.

The number of states is usually exponential in the number of players. However, we found that explicit implementation is able to handle reasonable model sizes. Currently, memory requirement is the bottleneck for the model checking, whereas computation time and numerical precision are lesser problems. Symbolic implementation is a natural next step to overcome the memory problem.

Chapter 7

Conclusions

7.1 Summary and evaluation

The aim of this work was to develop a framework for the *automatic formal analysis* of algorithms for competitive stochastic systems. We took the approach where such systems are modelled as stochastic multi-player games and the properties are provided as logic formulae, which are then automatically verified against the model. The main strength of such an analysis is the ability to automatically analyse strategic scenarios involving nondeterministic behaviour. The main weakness is the state-space explosion problem: the underlying SMG size is often exponential in the number of agents in the system. For example, in the case studies presented in this thesis, we were able to analyse systems composed of up to 7 agents. In models of such size it is difficult to capture emergent properties that require large scale. In this thesis, we have provided a set of results ranging from theoretical foundations to verification tools, which provide a framework for the formal analysis of competitive stochastic systems, and are summarised below.

In Chapter 4 we introduced the logic rPATL that is able to express branching-time and reward-based properties for SMGs. We developed model checking algorithms that are based on the solution of two-player stochastic games, for which we provided value iteration algorithms. We also developed strategy synthesis algorithms that use results of the value iteration algorithms directly to efficiently construct strategies with little additional overhead. We analysed the complexity of the model checking problem, showing that model checking of the logic is in NP∩coNP (excluding the $\mathsf{R}^{r}_{\bowtie x}[\mathsf{F}^{0}\phi]$ operator) and in NEXP∩coNEXP (for the full logic). The NP∩coNP complexity bound stems from the complexity of the reachability problem in two-player stochastic games, which has been a long-standing open problem [48]. The NEXP∩coNEXP complexity upper bound is obtained for the full logic, because at most exponential memory is required for the players to satisfy the logic formulae, which contain the $\mathsf{R}^{r}_{\bowtie x}[\mathsf{F}^{0}\phi]$ operator. In addition, we have introduced several extensions of rPATL, including reward-bounded properties, operators to find optimal coalitions that are required to satisfy the formula, and the logic rPATL*, which allows the use of LTL to specify properties of paths.

In Chapter 5 we extended the logic rPATL to support boolean combinations of objectives to allow richer specifications of properties. We found that, already for a conjunction or disjunction of two reachability objectives, such games are not determined and optimal strategies may not exist. Also, infinite memory may be required by the players to achieve a formula composed of the terminal state reachability objectives. This is in contrast with both multi-objective MDPs and single-objective two-player stochastic games, which are determined and optimal memoryless strategies exist. We also analysed the complexity of model checking the logic and showed that it is PSPACE-hard in general, undecidable if players are not allowed to use randomisation, and NP-complete if the formulae are restricted to disjunctions of total expected reward or terminal state reachability objectives. Nevertheless, we provided approximation algorithms that can be used for the implementation of verification and strategy synthesis for multi-objective rPATL.

In Chapter 6 we described PRISM-games, a tool implementing the rPATL model checking and synthesis algorithms. We applied the tool to several case studies of competitive stochastic systems (also presented in the chapter) to assess the strengths and weaknesses of the framework. We were able to find potential drawbacks of the protocols using rPATL model checking and synthesis, e.g., we found the incentive design problem of the pricing scheme in the microgrid demand-side management protocol and identified and constructed a strategy for attack on the reputation mechanism for user-centric networks. Although PRISM-games was able to handle models having several million states, the scalability turned out to be the biggest weakness. In the case studies that we addressed, the underlying SMG size scaled exponentially with the number of players/agents in the system, and hence we were able to verify systems consisting of up to 7 agents.

7.2 Future work

The work done in this thesis has raised many interesting research questions. In this section we discuss several of them.

Multi-objective stochastic games. There are several research directions with respect to the study of stochastic games with multiple objectives. From the theoretical point of view, some of the important questions that remain open are the following: providing an upper bound for complexity of deciding whether a multi-objective rPATL (or conjunctive rPATL) formula is true; providing guarantees on the accuracy of Pareto set approximation for non-stopping games; and deciding whether there exists a finite memory winning strategy to satisfy a given formula.

Techniques for model checking multi-objective MDPs have been successfully applied to the compositional reasoning about probabilistic models using assume-guarantee rules [79]. It would be interesting to investigate, whether multi-objective games could be used to provide a similar assume-guarantee-based framework for the compositional reasoning about stochastic games. This could be a useful approach to tackle the state space explosion problem, particularly applicable to competitive stochastic systems, because such systems are often composed of many autonomous components having small (local) state spaces, and the state-space explosion arises from their composition.

From the practical perspective, it would be important to investigate the implementation techniques that would allow to efficiently compute the Pareto sets for the multi-objective rPATL formulae. Another way to address this problem could be to consider alternative methods to verifying the formulae, e.g., instead of computing the Pareto sets for the formula, one may find a method to check the satisfiability of the formula without doing so (as it is done for MDPs). One may also consider providing a model checking algorithm for specific formulae, for example, as it is done for the $\mathsf{R}^r_{\leq x}[\mathsf{F}^\infty\phi]$ operator in rPATL, where we require the target set $Sat(\phi)$ to be reached with probability one, in addition to meeting the reward bound. This could be extended to properties other than reachability, e.g., requiring a strategy to satisfy a CTL formula, in addition to minimising the reward. Another direction is to extend the reward-bounded properties, presented for rPATL in Section 4.6, to contain bounds on multiple reward functions, for which the probability of paths satisfying a CTL or LTL formula could be maximised.

Connection with non-zero-sum games. Using rPATL, competitive scenarios are analysed to establish worst-case performance guarantees, analyse malicious behaviours and, to some extent, perform incentive design. However, the setting of zero-sum games is quite restrictive and the analysis is pessimistic, i.e., the strategy of a player in zero-sum game could be very different from a game where other agents try to maximise their own (not necessarily opposing) objectives. For example, in Section 5.6, we showed that if certain multi-objective rPATL formulae are satisfied in the game, then this implies existence of winning secure equilibria. This example shows that, in some cases, multi-objective rPATL can express the existence of equilibria, and the strategy synthesis algorithm could be used to construct the equilibria strategies for players. However, the exact relationship between games with multi-objective rPATL goals and non-zero sum solution concepts like Nash or secure equilibria is yet to be explored and provides a compelling direction for future work.

Specification language. The syntax of the logic rPATL and its multi-objective extension builds on ATL, PCTL and their extensions. Arguably, this does not provide the easiest way for people to write and/or read and understand the specifications, especially when they are meant to imply some other properties of games or contain several levels of nesting. For instance, in Section 6.6.2 we used multi-objective rPATL formula

$$\neg\langle\langle\{A\}\rangle\rangle(\neg\mathsf{P}_{\geq 0.9}[\mathsf{F}\,\mathsf{G}_\mathsf{B}]) \wedge \langle\langle\{A\}\rangle\rangle(\mathsf{P}_{\geq 0.5}[\mathsf{F}\,\mathsf{G}_\mathsf{A}] \vee \neg\mathsf{P}_{\geq 0.9}[\mathsf{F}\,\mathsf{G}_\mathsf{B}])$$

to specify that, under some conditions, it implies that there exists a winning secure equilibria strategy in the game for player A, but this is not obvious from the specification. Similarly, in Section 5.6, we used rPATL formula

$$\langle\langle C\rangle\rangle\mathsf{P}_{\max=?}[\,\mathsf{F}^{\leq n}\,\langle\langle C\rangle\rangle\mathsf{P}_{>0.9}[\,\mathsf{F}^{\leq 10}\,\bigwedge_{i=1}^{|\Pi|}\mathsf{p_i}{=}\mathsf{k_1}]]$$

to express the robustness of the system, but it is not clear what it means without the additional explanation. This is a common problem for temporal logics, and one of the ways that could be provided to address this issue could be creating frameworks that address specific types of problems and contain property specification templates, e.g., like Specification Pattern System for LTL [100], where one can construct natural language statements, which are then translated in corresponding LTL formula, for example, statement $AtLeastOne_A$ (where A is a set of n atomic propositions) is transformed into LTL formula $(\neg a_1 \wedge \cdots \wedge \neg a_n)\,\mathsf{U}\,(a_1 \vee \cdots \vee a_n)$; or ProProST [65], which is a tool for generating PCTL formulae from natural language specifications.

7.3 Conclusion

In this thesis we set out to provide a formal framework, based on model checking for stochastic games, for the analysis of systems, which incorporate competitive and stochastic behaviour. We introduced the temporal logic rPATL for specifying properties for such systems, which incorporates several expected total reward objectives that turn out to be very important for the analysis of case studies. For this logic we provided efficient model checking algorithms, based on the solution of two-player stochastic games using value iteration. We also provided strategy synthesis algorithms that use the results of the model checking algorithms to efficiently obtain the strategies for players achieving the objective.

We introduced an extension of the logic, which allows specifying that players of the game have a strategy to achieve several objectives at the same time. A similar problem has been addressed in a non-competitive setting of MDPs, where going from single to multiple objectives does not increase the complexity of model checking (both problems are in P). The only change is that memoryless randomised strategies may be needed instead of memoryless deterministic ones required for single objectives [57]. However, the competitive stochastic setting of games, when going from single to multiple objectives, the leap in complexity is much bigger. Even though, for single-objective games, memoryless deterministic strategies are also sufficient, for the setting with multiple objectives both randomisation and (infinite) memory may be required to win the game, and the model checking problem is PSPACE-hard, whereas for the single-objective problem it is in NP∩coNP [48]. Also, several other fundamental properties such as determinacy and existence of optimal strategies are not preserved. Interestingly, non-stochastic multi-objective games are both determined and optimal strategies exist [27, 35], and hence we show that combining stochastic and competitive behaviours gives rise to new problems, which were not present when considering them in isolation.

The implementation of the PRISM-games tool has allowed us to apply the methods developed in this thesis to competitive stochastic systems. We analysed several case studies and managed to identify important properties of systems that would otherwise be difficult to capture using other methods. For example, for the microgrid demand-side management protocol we identified an incentive related weakness using reward-based rPATL properties; also, using strategy synthesis for rPATL formulae for the reputation and virtual currency mechanism, we constructed malicious attacks that can be attempted by users to obtain unlimited number of unpaid services in the network. This suggests that the techniques developed in this thesis can be successfully applied to the analysis of competitive stochastic systems. Also, this work has raised several intriguing research questions that remain open and provide interesting directions for future research. □

Appendix A

Comparison of strategy models

In this appendix we prove Theorem 5 from Section 3.4, which states that stochastic-update strategies can be exponentially more succinct than the deterministic-update strategies. The proof is a special case of a more general result for stopping games showing the sufficient and necessary conditions for player $\Box$ to achieve a precise expectation of any linearly bounded function [39].

To prove this result, we use the game from Figure 3.7 starting in state s_1 and consider an objective, where we want player $\Box$ to reach a state labelled with t with probability exactly 0.5, i.e., we want to obtain a player $\Box$ strategy $\sigma_\Box \in \Sigma_\Box$ such that for all strategies of player $\Diamond$ we have $\mathrm{Pr}_{s_1}(\mathsf{F}\,\mathsf{t}) = 0.5$. Note that this effectively encodes two reachability objectives that a strategy has to satisfy at the same time, that is, to reach s_t with probability at least 0.5 and to reach s_f with probability at least 0.5. We start by proving that exponential memory is required for player $\Box$ strategy if deterministic-update representation is used. Proposition 1 shows that there are exponentially many different distributions that player $\Box$ has to use in its sole control state s_d in order to achieve the goal. Then, in Theorem 17, we prove that we can use stochastic-update to encode all of these distributions using a linear number of memory elements. In addition, the proof also establishes sufficient and necessary conditions to achieve any precise value for terminal state reachability problem in stopping games.

Proposition 1 *In SMG $\mathcal{G}$ from Figure 3.7, player $\Box$ requires exponential memory for any deterministic-update strategy in order to make sure that for all player $\Diamond$ strategies we have* $\mathrm{Pr}_{s_1}(\mathsf{F}\,\mathsf{t}) = 0.5$.

Proof. Observe that under any strategy of player $\Diamond$, the probability of runs that end in state s_{t_i} or s_{f_i} is exactly $\sum_{i=1}^{n} x_i \cdot \beta(i-1)$, where $\beta(k) = \prod_{j=1}^{k}(1-x_j)$. We now construct a deterministic update strategy $\sigma_\Box = \langle \mathcal{M}, \sigma_\Box^u, \sigma_\Box^n, \alpha \rangle$, which ensures that the probability to reach a state labelled with t is exactly 0.5. Intuitively, the strategy remembers the exact history, and upon arriving in s_d it identifies which states a_i, for $1 \leq i \leq n$ were visited on a prefix of a history (and hence how much of the probability mass was directed to states s_{t_i}), and sets the probability of going to s_f so that it "compensates" for these paths to target states to get the overall probability to reach target equal to 0.5. Formally,

- $\mathcal{M} = \{\mathrm{Prefix}(\lambda, k) \mid \lambda \in \Omega_{\mathcal{G},s_1}, k \in \{1, \ldots, 2n\}\}$
- $\sigma^u_\Box(m, s)$ equals $[m{\cdot}s \mapsto 1]$ if $m{\cdot}s \in \mathcal{M}$, and $[m \mapsto 1]$ otherwise,
- $\sigma^n_\Box(s_d, m) = [s_t \mapsto p, s_f \mapsto 1 - p]$, s.t. $p \cdot \beta(n) + \sum_{a_i \in m} x_i \cdot \beta(i - 1) = 0.5$
- $\alpha(s) = [s \mapsto 1]$

Note that the p above surely exists, because $\beta(n) \geq \beta(\infty) > \frac{1}{2}$. We argue that any strategy needs at least 2^n memory elements to achieve probability exactly 0.5. Otherwise, there are two different histories $s_1t_1s_2t_2 \ldots s_nt_ns_d$ and $s_1t'_1s_2t'_2 \ldots s_nt'_ns_d$, where $t_i, t'_i \in \{a_i, b_i\}$, after which $\sigma_\Box$ assigns the same distribution $[s_t \mapsto y, s_f \mapsto 1 - y]$. Let k be the smallest number such that $t_k \neq t'_k$, and without loss of generality suppose $t_k = a_k$. Let $\sigma_\Diamond \in \Sigma_\Diamond$ be a deterministic strategy that chooses to go to t_i in s_i, and let $\sigma'_\Diamond \in \Sigma_\Diamond$ be a deterministic strategy that chooses to go to t'_i in s_i. Then the probability to reach a target state under $\sigma_\Box$ and $\sigma_\Diamond$ is at least $\sum_{i<k,t_i=a_i} x_i{\cdot}\beta(i - 1) + x_k{\cdot}\beta(k - 1) + y{\cdot}\beta(n)$, and under $\sigma_\Box$ and $\sigma'_\Diamond$ it is at most $\sum_{i<k,t_i=a_i} x_i{\cdot}\beta(i - 1) + \sum_{k<i\leq n} x_i{\cdot}\beta(i - 1) + y{\cdot}\beta(n)$. Because $x_k{\cdot}\beta(k - 1) > (\sum_{k<i\leq n} x_i){\cdot}\beta(k - 1) > \sum_{k<i\leq n} x_i{\cdot}\beta(i - 1)$, we obtain a contradiction. $\Box$

We now prove a general result for stopping games showing that, if there exists a strategy for player $\Box$ which makes sure that for any strategy of player $\Diamond$ the target set of states T is reached with probability exactly x, then there exists a compact stochastic-update strategy with at most $2 \cdot \mathcal{M}$ memory elements. We first identify the states in which the formula is false for any x. In what follows we denote by $\mathrm{Pr}_s(\mathsf{F}\,T)$ the probability of the paths that reach a state in T, i.e., $\mathrm{Pr}_s(\mathsf{F}\,T) \overset{\text{def}}{=} \mathrm{Pr}_s(\{\lambda \in \Omega_\mathcal{G} \mid \exists i \in \mathbb{N}\,.\,\lambda_i \in T\})$.

Lemma 6 (Losing states) *Given and SMG $\mathcal{G}$, a state s, and a target set T, if*

$$\inf_{\sigma_\Box\in\Sigma_\Box} \sup_{\sigma_\Diamond\in\Sigma_\Diamond} \mathrm{Pr}_s(\mathsf{F}\,T) > \sup_{\sigma_\Box\in\Sigma_\Box} \inf_{\sigma_\Diamond\in\Sigma_\Diamond} \mathrm{Pr}_s(\mathsf{F}\,T)\,,$$

then for any $x \in [0, 1]$ player $\Box$ does not have a strategy such that for all strategies of player $\Diamond$ we have $\mathrm{Pr}_s(\mathsf{F}\,T) = x$.

Proof. For the sake of contradiction assume that there exists a winning strategy for player $\Box$, which achieves precise value for some x from s, and fix such a strategy $\sigma_\Box$. Observe that now there are two *distinct* probability values that player $\Diamond$ can ensure to reach a state in T, namely, $\inf_{\sigma_\Box \in \Sigma_\Box} \sup_{\sigma_\Diamond \in \Sigma_\Diamond} \Pr_s(\mathsf{F}\, T)$ and $\sup_{\sigma_\Box \in \Sigma_\Box} \inf_{\sigma_\Diamond \in \Sigma_\Diamond} \Pr_s(\mathsf{F}\, T)$, and hence at least one of them has to be different from x. $\Box$

Theorem 17 (Precise probability) *Given a stopping stochastic two-player game* $\mathcal{G} = \langle \{\Box, \Diamond\}, S, (S_\bigcirc, S_\Box, S_\Diamond), \Delta, AP, \chi\rangle$ *without states satisfying the condition of Lemma 6, a state* $s \in S$ *and a target set* $T \subseteq S$*, for all* x *such that*

$$\inf_{\sigma_\Box \in \Sigma_\Box} \sup_{\sigma_\Diamond \in \Sigma_\Diamond} \Pr_s(\mathsf{F}\, T) \leq x \leq \sup_{\sigma_\Box \in \Sigma_\Box} \inf_{\sigma_\Diamond \in \Sigma_\Diamond} \Pr_s(\mathsf{F}\, T)$$

player $\Box$ *has a winning stochastic-update strategy with* $2\cdot|S|$ *memory elements to guarantee the probability to reach* T *exactly* x*.*

Proof. We prove the theorem by constructing a winning strategy of the required size. Let f^T be a random variable that to every path assigns 1 if it reaches a state in T and 0 otherwise. Let $\sigma_\Box^-$ and $\sigma_\Box^+$ be memoryless deterministic strategies achieving, for every $s \in S$, the minimum and maximum expected value for f^T, respectively (such values exist by Theorem 4). We denote these values by $\mathrm{val}^-(s)$ and $\mathrm{val}^+(s)$. It is easy to see that they correspond to the maximal/minimal reachability probabilities. The winning strategy $\sigma_\Box = \langle \mathcal{M}, \sigma_\Box^u, \sigma_\Box^n, \alpha\rangle$ that achieves the reachability probability exactly x from state s is defined as follows:

- $\mathcal{M} = \{(s, \mathrm{val}^-(s)), (s, \mathrm{val}^+(s)) \mid s \in S\}$,
- $$\sigma_\Box^u((s,y),t) = \begin{cases} (t,y) & \text{if } s \in S_\Box, \\ [(t, \mathrm{val}^-(t)) \mapsto \beta(y,t), & \\ \ (t, \mathrm{val}^+(t)) \mapsto 1 - \beta(y,t)] & \text{if } s \in S_\Diamond, \\ (t, \mathrm{val}^-(t)) & \text{if } s \in S_\bigcirc \text{ and } y = \mathrm{val}^-(s), \\ (t, \mathrm{val}^+(t)) & \text{if } s \in S_\bigcirc \text{ and } y = \mathrm{val}^+(s), \end{cases}$$
- $$\sigma_\Box^n(s,(s,y)) = \begin{cases} \sigma_\Box^-(s) & \text{if } y = \mathrm{val}^-(s), \\ \sigma_\Box^+(s) & \text{otherwise} \end{cases}$$
- $\alpha(s) = [\langle s, \mathrm{val}^-(s)\rangle \mapsto \beta(x,s), \langle s, \mathrm{val}^+(s)\rangle \mapsto 1 - \beta(x,s)]$,

for all $s, t \in S$, and $(s,y) \in \mathcal{M}$, where $\beta(y,s) = c$ such that $0 \leq c \leq 1$ and $y = c \cdot \mathrm{val}^-(s) + (1-c)\cdot \mathrm{val}^+(s)$. Intuitively, the strategy updates its memory elements in such a way that the expectation of the second element of the memory tuple is equal to x. Now we can prove that such a way of updating the memory, together with the stopping game condition and the fact that for all states we have $\mathrm{val}^-(s) \leq \mathrm{val}^+(s)$ by Lemma 6, ensures that the expectation of this memory element is indeed equal to the expectation of the function f^T representing the reachability probability.

First, note that, for stopping games, there is $\delta > 0$ such that the probability for the game to end in a terminal state in $|S|$ steps is at least δ. The correctness of the construction of the strategy $\sigma_\Box$ now follows from the following steps.

1. Let $\sigma_\Diamond$ be an arbitrary strategy for player $\Diamond$, and let w be a path starting in the initial state s_0 in $\mathcal{G}_C$ and ending in a state s and memory element of $\sigma_\Box$ is (s, y) for some terminal state s. To show $f^T(\mathrm{Cyl}(w)) = y$ we use induction that for any finite path w ending in s with $\sigma_\Box$ memory being (s, y) we have

$$\inf_{\sigma_\Box \in \Sigma_\Box} \sup_{\sigma_\Diamond \in \Sigma_\Diamond} \mathbb{E}_{s_0}[f^T \mid \mathrm{Cyl}(w)] \leq y \leq \sup_{\sigma_\Box \in \Sigma_\Box} \inf_{\sigma_\Diamond \in \Sigma_\Diamond} \mathbb{E}_{s_0}[f^T \mid \mathrm{Cyl}(w)]\,.$$

   The claim follows because for paths that reach the terminal states the minimum and maximum values coincide.

2. In this step we prove that, after every step, the expected value of the second element of the memory of $\sigma_\Box$ stays equal to x. Let X_i be the random variable that assigns y to a path w of $\mathcal{G}_C$ where $w_i = s$ and define $\mathrm{mem}(w, i) = (s, y)$ to be the value of the memory element of $\sigma_\Box$ in the i^{th} step of that path. We prove that $\mathbb{E}_{s_0}(X_i) = x$ for all $i \in \mathbb{N}$ by induction on i.

   Because there are no states identified by Lemma 6, we have that for any $\mathrm{mem}(w, i) = (s, y)$ it holds that $\mathrm{val}^-(s) \leq y \leq \mathrm{val}^+(s)$.

   - **Base case.** For $i = 0$ we have that

$$\mathbb{E}_{s_0}[X_0] = \mathrm{val}^-(s_0) \cdot \beta(x, s_0) + \mathrm{val}^+(s_0) \cdot (1 - \beta(x, s_0)).$$

     If $\mathrm{val}^-(s_0) = x = \mathrm{val}^+(s_0)$, then $\beta(x, s_0) = 1$ and hence $\mathbb{E}_{s_0}[X_0] = \mathrm{val}^-(s_0) = x$. Otherwise,

$$\begin{aligned}\mathbb{E}_{s_0}[X_0] \quad = \quad & \mathrm{val}^-(s_0) \cdot \frac{\mathrm{val}^+(s_0) - x}{\mathrm{val}^+(s_0) - \mathrm{val}^-(s_0)} \\ & + \mathrm{val}^+(s_0) \cdot \left(1 - \frac{\mathrm{val}^+(s_0) - x}{\mathrm{val}^+(s_0) - \mathrm{val}^-(s_0)}\right) = x\end{aligned}$$

   - **Inductive case.** Assume that $\mathbb{E}_{s_0}[X_i] = x$. Let U_i be the set of all finite paths of length i in $\mathcal{G}_C$. By the induction hypothesis,

$$\mathbb{E}_{s_0}[X_i] = \sum_{\lambda \cdot s \in U_i} \Pr_{s_0}(\mathrm{Cyl}(\lambda \cdot s)) \cdot y = x,$$

where y is defined by $\mathrm{mem}(\lambda \cdot s, i) = (s, y)$. To show $\mathbb{E}_{s_0}[X_{i+1}] = x$, we consider each path $\lambda' = \lambda \cdot s \in U_i$ individually, and show that, after an action has been taken, the expected value of the second component of the memory remains y. There are three cases.

(a) Case $s \in S_\Box$. From the definition of $\sigma_\Box$, it follows that $\sigma^u_\Box((s, y), t) = (t, y)$ for all $t \in S$.

(b) Case $s \in S_\Diamond$. From the definition of the memory update function $\sigma^u_\Box$, we have that for every choice $t \in \Delta(s)$ of player $\Diamond$

$$\begin{aligned} &\mathrm{val}^-(t) \cdot \beta(y, t) + \mathrm{val}^+(t) \cdot (1 - \beta(y, t)) \\ = \quad &\mathrm{val}^-(t) \cdot \frac{\mathrm{val}^+(t) - y}{\mathrm{val}^+(t) - \mathrm{val}^-(t)} \\ &+ \mathrm{val}^+(t) \cdot \left(1 - \frac{\mathrm{val}^+(t) - y}{\mathrm{val}^+(t) - \mathrm{val}^-(t)}\right) = y \end{aligned}$$

if $\mathrm{val}^-(t) \neq \mathrm{val}^+(t)$, and $\mathrm{val}^-(t) = y$, otherwise.

(c) Case $s \in S_\bigcirc$. We know that either $y = \mathrm{val}^-(s)$ or $y = \mathrm{val}^+(s)$. By definition $\mathrm{val}^-(s) = \sum_{t \in S} \Delta(s, t) \cdot \mathrm{val}^-(t)$ and $\mathrm{val}^+(w) = \sum_{t \in S} \Delta(s, t) \cdot \mathrm{val}^+(t)$. From the construction of $\sigma_\Box$, we see that if $y = \mathrm{val}^-(s)$ then y is set to $\mathrm{val}^-(t)$ for all successors t and to $\mathrm{val}^+(t)$ if $y = \mathrm{val}^+(s)$, and therefore $\mathbb{E}_{s_0}[X_{i+1} \mid \mathrm{Cyl}(\lambda')] = y$.

3. Let U_k be the set of all finite paths in $\mathcal{G}_C$ that reach a terminal state in at most k steps and have non-zero probability measure under $\sigma_\Box$ and $\sigma_\Diamond$. And let $p(k) = (1 - \delta)^{\lfloor \frac{k}{|S|} \rfloor}$ be an upper bound on the probability not to reach a terminal state in k steps, where δ is the lower bound on the probability to terminate in $|S|$ steps. For any $k \in \mathbb{N}$, we have:

$$\begin{aligned} &\sum_{w \in U_k} \Pr_{s_0}(\mathrm{Cyl}(w)) \cdot \mathbb{E}_{s_0}[f^T \mid \mathrm{Cyl}(w)] - p(k) \\ &\qquad \le \mathbb{E}_{s_0}[f^T] \le \\ &\qquad \sum_{w \in U_k} \Pr_{s_0}(\mathrm{Cyl}(w)) \cdot \mathbb{E}_{s_0}[f^T \mid \mathrm{Cyl}(w)] + p(k) \end{aligned}$$

and as k goes to infinity, $p(k)$ goes to 0, thus

$$\mathbb{E}_{s_0}[f^T] = \lim_{k\to\infty} \sum_{w\in U_k} \Pr_{s_0}(\mathrm{Cyl}(w)) \cdot \mathbb{E}_{s_0}[f^T \mid \mathrm{Cyl}(w)]\,.$$

Also,

$$\sum_{w\in U_k} \Pr_{s_0}(\mathrm{Cyl}(w)) \cdot \mathrm{mem}(w,k) - p(k)$$

$$\leq \mathbb{E}_{s_0}[X_k] \leq$$

$$\sum_{w\in U_k} \Pr_{s_0}(\mathrm{Cyl}(w)) \cdot \mathrm{mem}(w,k) + p(k)$$

and again, as k goes to infinity $p(k)$ goes to 0, thus

$$\lim_{k\to\infty} \mathbb{E}_{s_0}[X_k] = \lim_{k\to\infty} \sum_{w\in U_k} \Pr_{s_0}(\mathrm{Cyl}(w)) \cdot \mathrm{mem}(w,k)\,.$$

We have shown in step 2) that $\mathrm{mem}(w,k) = \mathbb{E}_{s_0}[f^T \mid \mathrm{Cyl}(w)]$ for any $w \in U_k$, which together with the above result gives $x = \lim_{k\to\infty} \mathbb{E}[X_k] = \mathbb{E}_{s_0}[f^T]$ as required.

$\square$

By picking the values x_i small enough we ensure that there are no losing states (as per Lemma 6) in the game from Figure 3.7 and thus by Theorem 17 there exists a stochastic-update player $\Box$ strategy that contains exactly $2 \cdot |S|$ memory elements.

Appendix B

Proof of Theorem 8

The proof consists of two parts. First we prove in that for all k the sets X_s^k contain exactly the points achievable by some strategy in k steps, then we show that we can find k such that for all $s \in S$ all the points in the Pareto set of X_s^k are within ε from the Pareto points for $\varphi(\vec{r}, \vec{v})$, and vice versa.

We consider the coalition game $\mathcal{G}_C$ and define a the set of vectors than can be achieved by player $\Box$ strategy $\sigma_\Box$ in k steps as

$$R_{s,k}^{\sigma_\Box} \overset{\text{def}}{=} \{\vec{y} \in \mathbb{R}^n \,|\, \forall \sigma_\Diamond \in \Sigma_\Diamond \,.\, \mathbb{E}_s[rew^{\leq k}(\vec{r})] \geq \vec{y}\},$$

where $rew^{\leq k}(\vec{r})(\lambda) \overset{\text{def}}{=} \sum_{j=0}^{k} \vec{r}(\lambda_j)$, and we let $R_{s,k} \overset{\text{def}}{=} \bigcup_{\sigma_\Box \in \Sigma_\Box} R_{s,k}^{\sigma_\Box}$. For all $s \in S$, let X_s^k be the k-th iteration of the functional given by the equations from Theorem 8, starting with $X_s^0 = \{\vec{x} \in \mathbb{R}^n \mid \vec{x} \leq \vec{r}(s)\}$.

We start by proving that, for all $k \geq 0$, it is the case that $R_{s,k} = X_s^k$ by induction on k. The induction hypothesis is $\forall s \in S \,.\, \bigcup_{\sigma_\Box \in \Sigma_\Box} R_{s,k-1}^{\sigma_\Box} = X_s^{k-1}$,and we want to show that $\forall s \in S \,.\, \bigcup_{\sigma_\Box \in \Sigma_\Box} R_{s,k}^{\sigma_\Box} = X_s^k$.

- **Base case.** Let $k = 0$. We have that for all $s \in S$ and all strategies $\sigma_\Box \in \Sigma_\Box$,

$$\begin{aligned} R_{s,0}^{\sigma_\Box} &= \{\vec{x} \in \mathbb{R}^n \,|\, \forall \sigma_\Diamond \in \Sigma_\Diamond \,.\, \mathbb{E}_s[rew^{\leq 0}(\vec{r})] \geq \vec{x}\} \\ &= \{\vec{x} \in \mathbb{R}^n \,|\, \vec{x} \leq \vec{r}(s)\} \\ &= X_s^0. \end{aligned}$$

  Hence, $\forall s \in S \,.\, \bigcup_{\sigma_\Box \in \Sigma_\Box} R_{s,0}^{\sigma_\Box} = X_s^0$.

- **Induction step.** Suppose the claim holds for $k-1$, i.e. for all $s \in S$ we have that $\bigcup_{\sigma_\Box \in \Sigma_\Box} R_{s,k-1}^{\sigma_\Box} = X_s^{k-1}$. We suppose without loss of generality that s has exactly two successors s_1 and s_2. Furthermore, for $\ell \in \{1, 2\}$ we define $\sigma_\Box^\ell$ to be the strategy

$\sigma_\Box$ conditioned on picking the edge (s, s_ℓ), i.e., $\sigma_\Box^\ell(s_\ell \cdot \lambda) \overset{\text{def}}{=} \sigma_\Box(s \cdot s_\ell \cdot \lambda)$. We now distinguish several cases for $s \in S$.

– $s \in S_\Box$. For any $\sigma_\Box \in \Sigma_\Box$ we have that player $\Box$ picks s_1 with some probability $p \in [0,1]$ and s_2 with probability $1-p$. Hence, in s, player $\Box$ can achieve all points that can be achieved by some convex combination of some points in the successors of s. This can be stated formally as

$$R_{s,k}^{\sigma_\Box} = \mathsf{dwc}(\bigcup_{p\in[0,1]} (p \times R_{s_1,k-1}^{\sigma_\Box^1} + (1-p) \times R_{s_2,k-1}^{\sigma_\Box^2}) + \vec{r}(s)). \tag{B.1}$$

Further, for any convex sets $X_\ell \subseteq \mathbb{R}^n$ for $\ell \in \{1,2\}$, by the definition of the convex hull,

$$\bigcup_{p\in[0,1]} (p \times X_1 + (1-p) \times X_2) = \mathsf{conv}(\bigcup_\ell X_\ell). \tag{B.2}$$

Now, from the induction hypothesis and the definition of X_s^k, we get that

$$\begin{aligned}
&\bigcup_{\sigma_\Box\in\Sigma_\Box} R_{s,k}^{\sigma_\Box}\\
&\overset{(B.1)}{=} \bigcup_{\sigma_\Box\in\Sigma_\Box} \mathsf{dwc}(\bigcup_{p\in[0,1]} (p \times R_{s_1,k-1}^{\sigma_\Box^1} + (1-p) \times R_{s_2,k-1}^{\sigma_\Box^2}) + \vec{r}(s))\\
&= \mathsf{dwc}(\bigcup_{p\in[0,1]} \bigcup_{\sigma_\Box\in\Sigma_\Box} (p \times R_{s_1,k-1}^{\sigma_\Box^1} + (1-p) \times R_{s_2,k-1}^{\sigma_\Box^2}) + \vec{r}(s))\\
&= \mathsf{dwc}(\bigcup_{p\in[0,1]} (p \times (\bigcup_{\sigma_\Box\in\Sigma_\Box} R_{s_1,k-1}^{\sigma_\Box^1}) + (1-p) \times (\bigcup_{\sigma_\Box\in\Sigma_\Box} R_{s_2,k-1}^{\sigma_\Box^2}) + \vec{r}(s)))\\
&\overset{(B.2)}{=} \mathsf{dwc}(\mathsf{conv}(\bigcup_{\ell\in\{1,2\}} \bigcup_{\sigma_\Box\in\Sigma_\Box} R_{s_\ell,k-1}^{\sigma_\Box^\ell}) + \vec{r}(s))\\
&\overset{IH}{=} \mathsf{dwc}(\mathsf{conv}(\bigcup_{\ell\in\{1,2\}} X_s^{k-1}) + \vec{r}(s))\\
&\overset{def}{=} X_s^k.
\end{aligned}$$

– $s \in S_\Diamond$. For any $\sigma_\Box \in \Sigma_\Box$ we have that player $\Diamond$ picks s_1 with some probability $p \in [0,1]$ and s_2 with probability $1-p$. Hence, in s player $\Box$ can only achieve points that can be achieved by any convex combination of some points in the successors of s. This can be stated formally as

$$R_{s,k}^{\sigma_\Box} = \mathsf{dwc}(\bigcap_{p\in[0,1]} (p \times R_{s_1,k-1}^{\sigma_\Box^1} + (1-p) \times R_{s_2,k-1}^{\sigma_\Box^2}) + \vec{r}(s)). \tag{B.3}$$

Further, for any sets $X_\ell \subseteq \mathbb{R}^n$ for $\ell \in \{1,2\}$,

$$\bigcap_{p\in[0,1]} (p \times X_1 + (1-p) \times X_2) = \bigcap_{\ell} X_\ell, \tag{B.4}$$

which can be justified as follows:

- For any $\vec{x} \in \bigcap_{p\in[0,1]}(p \times X_1 + (1-p) \times X_2)$, let p be either 1 or 0, we obtain that $\vec{x} \in X_1$ and $\vec{x} \in X_2$ respectively.
- For $\vec{x} \in X_1 \cap X_2$, we have that for all $p \in [0,1]$, $p\vec{x} \in p \times X_1$ and $(1-p)\vec{x} \in (1-p) \times X_2$, so $\vec{x} = p\vec{x} + (1-p)\vec{x} \in (p \times X_1 + (1-p) \times X_2)$.

We now show that

$$\bigcup_{\sigma_\Box\in\Sigma_\Box} \bigcap_{\ell} R^{\sigma^\ell_\Box}_{s_\ell,k-1} = \bigcap_{\ell} \bigcup_{\sigma_\Box\in\Sigma_\Box} R^{\sigma_\Box}_{s_\ell,k-1}. \tag{B.5}$$

- $\subseteq$. Take $\vec{x} \in \bigcap_{\ell\in\{1,2\}} R^{\sigma^\ell_\Box}_{s_\ell,k-1}$ for some $\sigma_\Box \in \Sigma_\Box$. Then for any $\ell \in \{1,2\}$, $\vec{x} \in R^{\sigma^\ell_\Box}_{s_\ell,k-1} \subseteq \bigcup_{\sigma_\Box\in\Sigma_\Box} R^{\sigma_\Box}_{s_\ell,k-1}$. Hence, $\vec{x} \in \bigcap_\ell \bigcup_{\sigma_\Box\in\Sigma_\Box} R^{\sigma_\Box}_{s_\ell,k-1}$.
- $\supseteq$. Take $\vec{x} \in \bigcap_\ell \bigcup_{\sigma_\Box\in\Sigma_\Box} R^{\sigma_\Box}_{s_\ell,k-1}$. Therefore, for each $\ell \in \{1,2\}$ have a strategy $\sigma^\ell_\Box$ such that $\vec{x} \in R^{\sigma^\ell_\Box}_{s_\ell,k-1}$. We construct a strategy $\sigma_\Box$ from $\sigma^1_\Box$ and $\sigma^2_\Box$ as follows: $\sigma_\Box(s \cdot s_\ell \cdot \lambda) \overset{\text{def}}{=} \sigma^\ell_\Box(s_\ell \cdot \lambda)$ for all ℓ. Then $\sigma^\ell_\Box = \sigma^\ell_\Box$, and hence $R^{\sigma^\ell_\Box}_{s_\ell,k-1} = R^{\sigma^\ell_\Box}_{s_\ell,k-1}$. Therefore, we have that $\sigma_\Box$ satisfies $\vec{x} \in \bigcap_{\ell\in\{1,2\}} R^{\sigma^\ell_\Box}_{s_\ell,k-1}$ and hence $\vec{x} \in \bigcup_{\sigma_\Box\in\Sigma_\Box} \bigcap_\ell R^{\sigma^\ell_\Box}_{s_\ell,k-1}$.

Now, from the induction hypothesis and the definition of X^k_s, we get that

$$\begin{aligned}
\bigcup_{\sigma_\Box\in\Sigma_\Box} R^{\sigma_\Box}_{s,k} &\overset{\text{(B.3),(B.4)}}{=} \bigcup_{\sigma_\Box\in\Sigma_\Box} \mathsf{dwc}(\bigcap_\ell R^{\sigma^\ell_\Box}_{s_\ell,k-1} + \vec{r}(s)) \\
&= \mathsf{dwc}(\bigcup_{\sigma_\Box\in\Sigma_\Box} \bigcap_\ell R^{\sigma_\Box}_{s_\ell,k-1} + \vec{r}(s)) \\
&\overset{\text{(B.5)}}{=} \mathsf{dwc}(\bigcap_\ell \bigcup_{\sigma_\Box\in\Sigma_\Box} R^{\sigma_\Box}_{s_\ell,k-1} + \vec{r}(s)) \\
&\overset{IH}{=} \mathsf{dwc}(\bigcap_\ell (X^{k-1}_s + \vec{r}(s))) \\
&\overset{def}{=} X^k_s.
\end{aligned}$$

– $s \in S_\bigcirc$. We have that s_ℓ is picked with probability $\Delta(s, s_\ell)$. Hence, in s player $\Box$ can achieve all points that can be achieved by the convex combination with coefficients $\Delta(s, s_\ell)$ of some points in the successors of s. This can be stated formally as

$$R^{\sigma_\Box}_{s,k} = \mathsf{dwc}(\Delta(s,s_1) \times R^{\sigma^1_\Box}_{s_1,k-1} + \Delta(s,s_2) \times R^{\sigma^2_\Box}_{s_2,k-1} + \vec{r}(s)). \tag{B.6}$$

Now, from the induction hypothesis and the definition of X^k_s, we get that

$$\begin{aligned}
&\bigcup_{\sigma_\Box \in \Sigma_\Box} R^{\sigma_\Box}_{s,k} \\
&\quad \overset{(B.6)}{=} \bigcup_{\sigma_\Box \in \Sigma_\Box} \mathsf{dwc}((\Delta(s,s_1) \times R^{\sigma^1_\Box}_{s_1,k-1} + \Delta(s,s_2) \times R^{\sigma^2_\Box}_{s_2,k-1}) + \vec{r}(s)) \\
&\quad = \mathsf{dwc}(\bigcup_{\sigma_\Box \in \Sigma_\Box} (\Delta(s,s_1) \times R^{\sigma^1_\Box}_{s_1,k-1} + \Delta(s,s_2) \times R^{\sigma^2_\Box}_{s_2,k-1}) + \vec{r}(s)) \\
&\quad = \mathsf{dwc}(\Delta(s,s_1) \times \bigcup_{\sigma_\Box \in \Sigma_\Box} R^{\sigma^1_\Box}_{s_1,k-1} + \Delta(s,s_2) \times \bigcup_{\sigma_\Box \in \Sigma_\Box} R^{\sigma^2_\Box}_{s_2,k-1} + \vec{r}(s)) \\
&\quad \overset{IH}{=} \mathsf{dwc}(\Delta(s,s_1) \times X^{k-1}_{s_1} + \Delta(s,s_2) \times X^{k-1}_{s_2} + \vec{r}(s)) \\
&\quad \overset{def}{=} X^k_s.
\end{aligned}$$

To complete the proof we show that given an n-dimensional reward function $\vec{r}$, and $\varepsilon > 0$, after $k = |S| + \lceil |S| \cdot \frac{\ln(\varepsilon \cdot (n \cdot M)^{-1})}{\ln(1-\delta)} \rceil$ iterations of the functional F for any state $s \in S$, the Pareto set of X^k_s is an ε-approximation of the Pareto set for $\varphi(\vec{r}, \vec{v})$ of achievable vectors.

From the previous part of the proof we know that $X^k_s = R_{s,k}$ for all k, i.e. the Pareto set of points achievable by player $\Box$ in k steps is computed by k iterations of F; and from the stopping game assumption we know that, after $|S|$ steps, the game has terminated with probability at least $\delta = p^{|S|}_{\min}$, where $p_{\min}$ is the minimum positive probability in $\mathcal{G}_C$. Hence, the maximum change to any dimension to any vector in X^k_s after k steps of the iteration is less than $M \cdot (1-\delta)^{\lfloor \frac{k}{|S|} \rfloor}$, which is also the maximum change that any strategy can make over a strategy that is optimal for k steps.

Hence, for ε-optimality after k steps, we need to pick a k such that $\varepsilon > n \cdot M \cdot (1-\delta)^{\lfloor \frac{k}{|S|} \rfloor}$. The factor n is because ε-optimality requires that the strategy achieves a point that is

ε-close in each of the n dimensions individually. We get that

$$\begin{aligned}
\varepsilon > n{\cdot}M \cdot (1-\delta)^{\lfloor \frac{k}{|S|} \rfloor} &\Leftrightarrow \ln(\varepsilon) > \ln(n{\cdot}M) + \left\lfloor \frac{k}{|S|} \right\rfloor \cdot \ln(1-\delta) \\
&\Leftrightarrow \frac{\ln(\varepsilon \cdot (n{\cdot}M)^{-1})}{\ln(1-\delta)} < \left\lfloor \frac{k}{|S|} \right\rfloor \\
&\Leftarrow \frac{\ln(\varepsilon \cdot (n{\cdot}M)^{-1})}{\ln(1-\delta)} < \frac{k}{|S|} - 1 \\
&\Leftarrow |S| + |S| \cdot \frac{\ln(\varepsilon \cdot (n{\cdot}M)^{-1})}{\ln(1-\delta)} < k
\end{aligned}$$

Set $k = |S| + \lceil |S| \cdot \frac{\ln(\varepsilon \cdot (n \cdot M)^{-1})}{\ln(1-\delta)} \rceil$. Note that the Pareto set for φ at state s is defined by the Pareto set of $R_s = \{\vec{y} \in \mathbb{R}^n \mid \exists \sigma_\Box \in \Sigma_\Box \,.\, \forall \sigma_\Diamond \in \Sigma_\Diamond \,.\, \mathbb{E}_s[rew(\vec{r}, \emptyset)] \geq \vec{y}\}$,, which is the set whose approximation we aim to compute. We have the following:

- For any point $\vec{x} \in R_s$ there is (by definition) a strategy π which achieves $\vec{x}$, i.e., for all $\sigma_\Diamond$ we have $\mathbb{E}^{\sigma_\Box,\sigma_\Diamond}_{\mathcal{G}_C,s}[rew(\vec{r}, \emptyset)] \geq \vec{x}$. Above we argued that we can find a player $\Box$ strategy $\sigma_\Box$ that after k steps achieves a point that differs from $\vec{x}$ by at most ε in each dimension. Hence, there is a player $\Box$ strategy $\sigma_\Box$ such that for all player $\Diamond$ strategies $\sigma_\Diamond$ we have that $\mathbb{E}^{\sigma_\Box,\sigma_\Diamond}_{\mathcal{G}_C,s}[rew^{\leq k}(\vec{r})] \geq \vec{x} - \varepsilon$, which means that $\vec{x} - \varepsilon \in R_{s,k} = X^k_s$.

- For any point $\vec{x} \in X^k_s = R_{s,k}$, let $\sigma_\Box$ be the strategy that ensures $\vec{x}$ is achieved in k steps, i.e. for all $\sigma_\Diamond$ we have $\mathbb{E}^{\sigma_\Box,\sigma_\Diamond}_{\mathcal{G}_C,s}[rew^{\leq k}(\vec{r})] \geq \vec{x}$. Again, by the above argument the point $\vec{x}$ achieved by $\sigma_\Box$ in k steps may only change by at most ε in each dimension by any other strategy. Hence we have $\mathbb{E}_s[rew(\vec{r}, \emptyset)] \geq \vec{x} - \varepsilon$ for all $\sigma_\Diamond$, and so $\vec{x} - \varepsilon \in R_s$.

Appendix C

Proof of Theorem 12

We prove that infinite memory is required for player $\Box$ to win the game from Figure 5.6. Let $h(k) = s_0(s_1s_2s_3s_0)^k$. Every strategy $\sigma_\Box$ for player $\Box$ determines (and is uniquely given by) an infinite sequence of vectors

$$\begin{aligned}
\vec{p}^k &= (\sigma_\Box(h(k)s_4)(1), \sigma_\Box(h(k)s_4)(s_5)\cdot\frac{1}{4}, \sigma_\Box(h(k)s_4)(s_5)\cdot\frac{3}{4}),\\
\vec{q}^k &= (\sigma_\Box(h(k)s_1s_2s_8)(s_9)\cdot\frac{1}{2}, \sigma_\Box(h(k)s_1s_2s_8)(2), \sigma_\Box(h(k)s_1s_2s_8)(s_9)\cdot\frac{1}{2}),\\
\vec{w}^k &= (\frac{1}{3}, \frac{2}{3}\cdot\sigma_\Box(h(k)s_1s_6s_7)(2), \frac{2}{3}\cdot\sigma_\Box(h(k)s_1s_6s_7)(3)),\\
\vec{z}^k &= (\frac{2}{3}\cdot\sigma_\Box(h(k)s_1s_2s_3s_{10}s_{11})(1), \frac{1}{3}, \frac{2}{3}\cdot\sigma_\Box(h(k)s_1s_2s_3s_{10}s_{11})(1)).
\end{aligned}$$

The intuitive interpretation of vector $\vec{p}^k$ (resp. $\vec{q}^k$, $\vec{w}^k$, $\vec{z}^k$) is that it represents the probability of reaching states $(1,2,3)$ from s_4 (resp. s_8, s_6, s_{10}) in the k-th step A (resp. B).

We define a winning strategy $\sigma_\Box$ by means of the vectors

$$\begin{aligned}
\vec{p}^k &= (1-\tfrac{1}{3\cdot 2^{k-1}}, \tfrac{1}{3\cdot 2^{k+1}}, \tfrac{1}{2^{k+1}}), & \vec{q}^k &= (\tfrac{1}{3\cdot 2^{k+1}}, 1-\tfrac{1}{3\cdot 2^{k}}, \tfrac{1}{3\cdot 2^{k+1}}),\\
\vec{w}^k &= (\tfrac{1}{3}, +\tfrac{1}{6}+\tfrac{1}{2}-\tfrac{1}{3\cdot 2^{n+2}}), 1-w_1^k-w_2^k), & \vec{z}^k &= (\tfrac{2}{3}-\tfrac{1}{3\cdot 2^{n+1}}, \tfrac{1}{3}, 1-z_1^k-z_2^k).
\end{aligned}$$

as follows. First, suppose player $\Diamond$ picks a strategy $\sigma_\Diamond$ which does not take the "*check*" transition before the $(n+1)$st visit to s_0. Then the probability of the runs that reach $\mathsf{1}$ while visiting s_0 at most $n+1$ times is independent of $\sigma_\Diamond$ and equal to $V(1,n) = \frac{1}{3} - \frac{1}{3\cdot 2^{2n+1}}$, as can be shown by the straightforward induction on n.

$$\begin{aligned}
V(1,n) &= V(1,n-1) + \frac{1}{2^{2n}} \cdot (q_1^n + \frac{1}{2} \cdot p_1^{n+1}) \\
&= V(1,n-1) + \frac{1}{2^{2n}} \cdot (\frac{1}{3 \cdot 2^{n+1}} + \frac{1}{2}(1 - \frac{1}{3 \cdot 2^n})) \\
&= V(1,n-1) + \frac{1}{2^{2n}} \cdot (\frac{1}{3 \cdot 2^{n+1}} + \frac{1}{2} - \frac{1}{3 \cdot 2^{n+1}}) \\
&= V(1,n-1) + \frac{1}{2^{2n+1}} \\
&= \frac{1}{3} - \frac{1}{3} \cdot \frac{1}{2^{2n-1}} + \frac{1}{2^{2n+1}} = \frac{1}{3} - \frac{1}{3 \cdot 2^{2n+1}}.
\end{aligned}$$

Further, supposing player $\Diamond$ picks a strategy $\sigma_\Diamond$ which does not take the "*check*" transition before the (n+1)st visit to s_2, the probability of the runs that reach $\mathsf{1}$ while visiting s_2 at most $n+1$ times is also independent of $\sigma_\Diamond$ and equal to $V(2,n) = \frac{1}{3} - \frac{1}{3 \cdot 2^{2n+2}}$. This is again shown by an induction on n.

$$\begin{aligned}
V(2,n) &= V(2,n-1) + \frac{1}{2^{2n+1}} \cdot (p_2^{n+1} + \frac{1}{2} q_2^{n+1}) \\
&= V(2,n-1) + \frac{1}{2^{2n+1}} \cdot (\frac{1}{3 \cdot 2^{n+2}} + \frac{1}{2}(1 - \frac{1}{3 \cdot 2^{n+1}})) \\
&= V(2,n-1) + \frac{1}{2^{2n+1}} \cdot (\frac{1}{3 \cdot 2^{n+2}} + \frac{1}{2} - \frac{1}{3 \cdot 2^{n+2}})) \\
&= V(2,n-1) + \frac{1}{2^{2n+2}} \\
&= \frac{1}{3} - \frac{1}{3 \cdot 2^{2n}} + \frac{1}{2^{2n+2}} = \frac{1}{3} - \frac{1}{3 \cdot 2^{2n+2}}.
\end{aligned}$$

Now we are ready to show that $\sigma_\Box$ is winning by showing that the probabilities of reaching states labelled with $\mathsf{1}$ and $\mathsf{2}$ are both $\frac{1}{3}$ under any $\sigma_\Diamond$. By [87] it suffices to consider deterministic strategies $\sigma_\Diamond$. First, consider a strategy $\sigma_\Diamond$, which never takes any transition labelled "*check*". We have $\mathrm{Pr}_{s_0}(\mathsf{F}\,\mathsf{1}) = \lim_{n\to\infty} V(1,n) = \frac{1}{3}$ and $\mathrm{Pr}_{s_0}(\mathsf{F}\,\mathsf{2}) = \lim_{n\to\infty} V(2,n) = \frac{1}{3}$; and for a strategy $\sigma_\Diamond$ which picks "*check*" on the (n+1)st visit to s_1 we have

$$\begin{aligned}
&\mathrm{Pr}_{s_0}(\mathsf{F}\,\mathsf{1}) = V(1,n) + w_1^n = \frac{1}{3} - \frac{1}{3 \cdot 2^{2n+1}} + \frac{1}{2^{2n+1}} \cdot \frac{1}{3} = \frac{1}{3}, \\
&\mathrm{Pr}_{s_0}(\mathsf{F}\,\mathsf{2}) = V(2,n) - \frac{1}{2^{2n+2}} \cdot (1 - \frac{1}{3 \cdot 2^{n+1}}) + \frac{1}{2^{2n+1}} \cdot w_2^n \\
&\qquad = \frac{1}{3} - \frac{1}{3 \cdot 2^{2n+2}} - \frac{1}{2^{2n+2}} \cdot (1 - \frac{1}{3 \cdot 2^{n+1}}) + \frac{1}{2^{2n+1}} \cdot w_2^n = \frac{1}{3}.
\end{aligned}$$

Finally, for a strategy $\sigma_\Diamond$ which picks "*check*" on the $(n{+}1)$st visit to s_3 we have

$$\begin{aligned}
\Pr_{s_0}(\mathsf{F}\,1) &= V(1,n) + \frac{1}{2^{2n+2}} \cdot \frac{1}{3 \cdot 2^{n+1}} + \frac{1}{2^{2n+2}} \cdot z_1^n \\
&= \frac{1}{3} - \frac{1}{3 \cdot 2^{2n+1}} + \frac{1}{2^{2n+2}} \cdot \frac{1}{3 \cdot 2^{n+1}} + \frac{1}{2^{2n+2}} \cdot z_1^n = \frac{1}{3}, \\
\Pr_{s_0}(\mathsf{F}\,2) &= V(2,n) + z_2^n = \frac{1}{3} - \frac{1}{3 \cdot 2^{2n+2}} + \frac{1}{2^{2n+2}} \cdot \frac{1}{3} = \frac{1}{3}.
\end{aligned}$$

We have shown that $\sigma_\Diamond$ ensures that each of the states labelled with $\mathsf{1}$, $\mathsf{2}$ and $\mathsf{3}$ is reached with probability exactly $\frac{1}{3}$. Now we show that there is no finite-memory strategy ensuring this. Let $\bar{\sigma}_\Box$ be a finite memory strategy, determined by vectors $\bar{\vec{p}}^k$, $\bar{\vec{q}}^k$, $\bar{\vec{w}}^i$ and $\bar{\vec{z}}^i$. Since $\bar{\sigma}_\Box$ is finite memory, there must be a k such that $\bar{\vec{p}}^k \neq \vec{p}$ or $\bar{\vec{q}}^k \neq \vec{q}$. Let k be the lowest such number. There are two possibilities:

- $\bar{\vec{p}}^k \neq \vec{p}^k$. Then necessarily $\bar{p}_1^k \neq p_1^k$. Also note that $w_1^k = \bar{w}_1^k = \frac{1}{3}$. We define the counter-strategy $\sigma_\Diamond$ to take "*check*" on the $(k{+}1)$st visit to s_1 and get (because k is minimal) that $\Pr_{s_0}(\mathsf{F}\,1) = \frac{1}{3} + \frac{1}{2^{2k \cdot +1}}(\bar{p}_1^k - p_1^k) \neq \frac{1}{3}$.

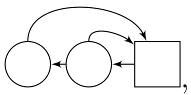
- $\bar{\vec{q}}^k \neq \vec{q}^k$. Then necessarily $\bar{q}_2^k \neq q_2^k$ and $z_1^k = \bar{z}_1^k = \frac{1}{3}$ and so we can define the counter-strategy σ to take "*check*" on the $(k{+}1)$th visit to s_3. We get $\Pr_{s_0}(\mathsf{F}\,2) = \frac{1}{3} + \frac{1}{2^{2k \cdot +1}}(\bar{q}_2^k - q_2^k) \neq \frac{1}{3}$.

This completes the proof.

Appendix D

Proof of Theorem 15

We prove the result in three steps. In the first we show how to construct a game for a given two-counter machine using the gadgets from Figure 5.8, then, in the second, we prove that the have structure to force player □ to pick the probability distributions in the required states according to counter updates, and finally, in the third step, we establish the claim that player □ has a winning strategy in $\mathcal{G}(\mathcal{M})$ if and only if $\mathcal{M}$ does *not* terminate.

1. Let $\mathcal{M}$ be a two-counter machine. We define the game $\mathcal{G}(\mathcal{M})$ incrementally. The game has two players, □ and ◊. For each type of instruction, we have a corresponding gadget, i.e., Init, Terminate, Increment, and Decrement, which are shown in Figure 5.8. In this figure, player □ states with double border are of the form , which allows player □ to simulate any probability distribution with deterministic strategies.

   Note that, for Increment and Decrement gadgets, $j \in \{1, 2\}$ refers to the counter (c_1 or c_2) that on which operation is applied in the instruction and i is the other counter. The game $\mathcal{G}(\mathcal{M})$ is then constructed by "gluing" the instructions together. Namely,

   - for the initial instruction "$l_0 : c_1 := c_2 := 0$ and goto l_k;", we use the Init gadget and link $(init)_{out}$ to $(q_k, op)_{in}$, if l_k is not terminal location where op is the operation type, and we link $(init)_{out}$ to $(term)_{in}$ if l_k is a terminal location;
   - for the increment instruction "$l_i : c_j := c_j{+}1$ and goto l_k;", we use the Increment gadget and link $(q_i, inc_j)_{out}$, if l_k is not terminal location where op is the operation type, and we link $(q_i, inc_j)_{out}$ to $(term)_{in}$ if l_k is a terminal location;
   - for the decrement instruction if "$c_j = 0$ then goto l_k else $c_j = c_j - 1$ and goto $l_{k'}$;", we use the Decrement gadget and link $(q_i, dec_j)^{=0}_{out}$ to $(q_k, op_k)_{in}$ (resp.

$(q_i, dec_j)^{>0}_{out}$ to $(q_{k'}, op_{k'})_{in})$, if l_k (resp. $l_{k'}$) is not a terminal location where op_k (resp. $op_{k'}$) is the operation type, and we link $(q_i, dec_j)^{=0}_{out}$ (resp. $(q_i, dec_j)^{>0}_{out}$) to $(term)_{in}$ if l_k (resp. $l_{k'}$) is a terminal location.

Note that the $(init)_{in}$ is also the initial state of the whole game and we use $s_{init} = (init)_{in}$ as alias for it. We label the states as "targets" as follows. States a^t_j, a_j are labelled with atomic proposition $\mathsf{T}_{\mathsf{a_j}}$, states b^t_j, b_j are labelled with atomic proposition $\mathsf{T}_{\mathsf{b_j}}$ for $j \in \{1, 2\}$, states c, c^t are labelled with T_{c}, and states $a^t_1, a^t_2, b^t_1, b^t_2, c^t$ are all labelled with T_{t}. We consider the following multi-objective rPATL formula

$$\varphi = \langle\langle\{\Box\}\rangle\rangle\, (\mathsf{P}_{\geq\frac{1}{6}}[\mathsf{F}\,\mathsf{T}_{\mathsf{a_1}}] \wedge \mathsf{P}_{\geq\frac{1}{6}}[\mathsf{F}\,\mathsf{T}_{\mathsf{b_1}}] \wedge \mathsf{P}_{\geq\frac{1}{6}}[\mathsf{F}\,\mathsf{T}_{\mathsf{a_2}}] \wedge \mathsf{P}_{\geq\frac{1}{6}}[\mathsf{F}\,\mathsf{T}_{\mathsf{b_2}}] \wedge \mathsf{P}_{\geq\frac{1}{3}}[\mathsf{F}\,\mathsf{T}_{\mathsf{c}}] \wedge \mathsf{P}_{\geq 1}[\mathsf{F}\,\mathsf{T}_{\mathsf{t}}])\,.$$

From now on we denote the player $\Box$ strategy, which achieves φ in the state s_{init} of the game $\mathcal{G}(\mathcal{M})$ by $\sigma^*_\Box$.

2. First observe that, since states labelled with $\mathsf{T}_{\mathsf{a_1}}$, $\mathsf{T}_{\mathsf{b_1}}$, $\mathsf{T}_{\mathsf{a_2}}$, $\mathsf{T}_{\mathsf{b_2}}$, and T_{c} form a partition of the terminal states of $\mathcal{G}(\mathcal{M})$, and hence for any pair of strategies $\sigma_\Box$ and $\sigma_\Diamond$, $\mathrm{Pr}_{s_{init}}(\mathsf{F}\,\mathsf{T}_{\mathsf{a_1}}) + \mathrm{Pr}_{s_{init}}(\mathsf{F}\,\mathsf{T}_{\mathsf{a_2}}) + \mathrm{Pr}_{s_{init}}(\mathsf{F}\,\mathsf{T}_{\mathsf{b_1}}) + \mathrm{Pr}_{s_{init}}(\mathsf{F}\,\mathsf{T}_{\mathsf{b_2}}) + \mathrm{Pr}_{s_{init}}(\mathsf{F}\,\mathsf{T}_{\mathsf{c}}) = 1$; therefore it follows that for any winning strategy $\sigma_\Box$ that achieves φ, it must be the case that for any player $\Diamond$ strategy $\sigma_\Diamond$, $\mathrm{Pr}_{s_{init}}(\mathsf{F}\,\mathsf{T}_{\mathsf{a_1}}) = \mathrm{Pr}_{s_{init}}(\mathsf{F}\,\mathsf{T}_{\mathsf{a_2}}) = \mathrm{Pr}_{s_{init}}(\mathsf{F}\,\mathsf{T}_{\mathsf{b_1}}) = \mathrm{Pr}_{s_{init}}(\mathsf{F}\,\mathsf{T}_{\mathsf{b_2}}) = \frac{1}{6}$ and $\mathrm{Pr}_{s_{init}}(\mathsf{F}\,\mathsf{T}_{\mathsf{c}}) = \frac{1}{3}$.

   We show that, in $\mathcal{G}(\mathcal{M})$, the strategy achieving φ, $\sigma^*_\Box$, must guarantee that, under any player $\Diamond$ strategy $\sigma_\Diamond$, the following properties hold:

   (a) For each state $(q_k, inc_j)_{in}$, $(q_k, dec_j)_{in}$, the reachability probability to $\mathsf{T}_{\mathsf{b_1}}$ and the reachability probability to $\mathsf{T}_{\mathsf{b_2}}$ both must be exactly $\frac{1}{6}$.

   (b) For each state $(q_k, inc_j)_2$ and $(q_k, dec_j)^{>0}_2$, the reachability probability to $\mathsf{T}_{\mathsf{a_1}}$ and the reachability probability to $\mathsf{T}_{\mathsf{a_2}}$ both must be exactly $\frac{1}{6}$.

   To see (a), we examine each gadget, in particular, the "out" states $(q_k, \star)_{out}$, where $\star \in \{dec_j, inc_j\}$. Consider any two different player $\Diamond$ strategies $\sigma^1_\Diamond$ and $\sigma^2_\Diamond$, which select, at $(q_k, \star)_{out}$, the horizontal and the vertical edge respectively. As $\sigma^*_\Box$ has to guarantee that for any player $\Diamond$ strategy the probability to reach $\mathsf{T}_{\mathsf{b_1}}$ is the same for $\sigma^1_\Diamond$ and $\sigma^2_\Diamond$ (namely $\frac{1}{6}$), at $(q_k, \star)_{out}$, the strategy pairs $\sigma^*_\Box, \sigma^1_\Diamond$ and $\sigma^*_\Box, \sigma^2_\Diamond$ must give the same probability to reach $\mathsf{T}_{\mathsf{b_1}}$ as well. From the gadget, the probability to reach $\mathsf{T}_{\mathsf{b_1}}$ following $\sigma^2_\Diamond$ is $\frac{1}{2} \cdot \frac{1}{3} = \frac{1}{6}$, hence the claim. The same holds for $\mathsf{T}_{\mathsf{b_2}}$.

   To see (b), we examine the states $(q_k, inc_j)_1$ and $(q_k, dec_j)^{>0}_1$. By the same argument as (a), the probability to reach a state labelled with $\mathsf{T}_{\mathsf{a_1}}$ must be $\frac{1}{2} \cdot \frac{1}{3} = \frac{1}{6}$. The same holds for $\mathsf{T}_{\mathsf{a}_2}$.

3. In this step we show that each gadget has a property that forces any winning strategy $\sigma^*_\Box$ of player $\Box$ to pick probability distributions corresponding to the operations on counters. In the next step we show that the assumptions stated here indeed hold when the game progresses from one gadget to another.

   Init. In player $\Box$ state s_0, $\sigma^*_\Box$ must select the edge (s_0, s_1) with probability $x = \frac{2}{3} = \frac{2}{3 \cdot 2^0}$. To see this, consider the strategy $\sigma_\Diamond$ for player $\Diamond$, which selects s_4 at state $(init)_{out}$. As the probability of reaching $\mathsf{T}_{\mathsf{a}_1}$ under $\sigma^*_\Box$ and $\sigma_\Diamond$ is $\frac{1}{6}$, and hence it must be the case that
   $$\frac{1}{2} \cdot \frac{1}{2} \cdot (1 - x) + \frac{1}{2} \cdot \frac{1}{2} \cdot \frac{1}{3} = \frac{1}{6},$$
   yielding that $x = \frac{2}{3}$, as desired. By a similar argument for $\mathsf{T}_{\mathsf{a}_2}$, at state s_2, for $\sigma_\Box$ the probability of selecting the edge (s_2, s_3) must be $\frac{2}{3} = \frac{2}{3 \cdot 2^0}$.

   This corresponds to setting both counter values to 0.

   Increment. The key property of the gadget is that when the probability of selecting edge (s_5, s_6) for $\sigma^*_\Box$ is $\frac{2}{3 \cdot 2^{c_j}}$, then the probability to select the edge (s_9, s_{10}) must be $\frac{2}{3 \cdot 2^{c_j+1}}$. To see this, suppose the probability to pick the edge (s_9, s_{10}) is x, and consider a player $\Diamond$ strategy $\sigma_\Diamond$, which selects the vertical edge at $(q_k, inc_j)_{out}$. By (a), the reachability probability to $\mathsf{T}_{\mathsf{b}_\mathsf{j}}$ must be $\frac{1}{6}$. This entails that
   $$\frac{1}{2} \cdot \frac{1}{2} \cdot \frac{2}{3 \cdot 2^{c_j}} \cdot \frac{1}{4} + \frac{1}{2} \cdot \frac{1}{2} \cdot \frac{1}{2} \cdot (1 - x) + \frac{1}{2} \cdot \frac{1}{2} \cdot \frac{1}{2} \cdot \frac{1}{3} = \frac{1}{6},$$
   which implies that $x = \frac{2}{3 \cdot 2^{c_j+1}}$, as desired corresponding to the increase of the counter value.

   Similarly, if the probability of selecting edge (s_7, s_8) for $\sigma^*_\Box$ is $\frac{2}{3 \cdot 2^{c_i}}$, then the probability of selecting edge (s_{11}, s_{12}) must be $\frac{2}{3 \cdot 2^{c_i}}$ as well. To see this, we repeat the same argument as the previous case and consider the reachability probability to $\mathsf{T}_{\mathsf{b}_\mathsf{i}}$, which yields, by (a), that
   $$\frac{1}{2} \cdot \frac{1}{2} \cdot \frac{2}{3 \cdot 2^{c_i}} \cdot \frac{1}{2} + \frac{1}{2} \cdot \frac{1}{2} \cdot \frac{1}{2} \cdot (1 - x) + \frac{1}{2} \cdot \frac{1}{2} \cdot \frac{1}{2} \cdot \frac{1}{3} = \frac{1}{6},$$
   where x is the probability of selecting edge (s_{11}, s_{12}) for $\sigma^*_\Box$. This implies that $x = \frac{2}{3 \cdot 2^{c_i}}$, corresponding to counter value remaining unchanged.

   Decrement. Note that, since player $\Box$ strategies are restricted to deterministic, it has to pick either edge labelled by ">0" or "$=0$". A basic observation is that when entering the state $(q_k, dec_j)_{in}$, suppose that $\sigma^*_\Box$ selects the edge labelled by "> 0", and that the probability of selecting the edge (s_{13}, s_{14}) is $\frac{2}{2^{c_j}}$, then the probability of selecting edge (s_{17}, s_{18}) must be $\frac{2}{3 \cdot 2^{c_j - 1}}$. To see this, suppose the probability of the edge (s_{17}, s_{18}) is x, and consider a player $\Diamond$ strategy $\sigma_\Diamond$ which selects the vertical edge at $(q_k, dec_j)_{out}$. It follows that the reachability probability to $\mathsf{T}_{\mathsf{b_j}}$ must satisfy

$$\frac{1}{2} \cdot \frac{1}{2} \cdot \frac{1}{3} \cdot \frac{2}{2^{c_j}} + \frac{1}{2} \cdot \frac{1}{2} \cdot \frac{1}{2} \cdot (1 - x) + \frac{1}{2} \cdot \frac{1}{2} \cdot \frac{1}{2} \cdot \frac{1}{3} \cdot = \frac{1}{6},$$

which implies that $x = \frac{2}{3 \cdot 2^{c_j - 1}}$, which corresponds to the counter decrement.

Similarly, suppose that $\sigma^*_\Box$ selects the edge labelled by "> 0," and that the probability of selecting edge (s_{15}, s_{16}) is $\frac{2}{3 \cdot 2^{c_i}}$, then the probability of selecting edge (s_{19}, s_{20}) must be $\frac{2}{3 \cdot 2^{c_i}}$ as well. To see this, we repeat the same argument as the previous case and consider the reachability probability to $\mathsf{T}_{\mathsf{b_i}}$, which yields

$$\frac{1}{2} \cdot \frac{1}{2} \cdot \frac{2}{3 \cdot 2^{c_i}} \cdot \frac{1}{2} + \frac{1}{2} \cdot \frac{1}{2} \cdot \frac{1}{2} \cdot \frac{1}{2} \cdot (1 - x) + \frac{1}{2} \cdot \frac{1}{2} \cdot \frac{1}{2} \cdot \frac{1}{3} = \frac{1}{6}.$$

This implies that $x = \frac{2}{3 \cdot 2^{c_i}}$ corresponding to the counter value remaining unchanged.

4. As the next step, we shall verify that when two instructions are "glued" together, the counter values do not change, i.e., the assumptions on probabilities player by $\sigma^*_\Box$ that we used in the previous step hold.

   Init+Increment. We show that the probabilities of selecting edges (s_5, s_6) and (s_7, s_8) for $\sigma^*_\Box$ must be $x = \frac{2}{3 \cdot 2^0}$. To see this, consider the player $\Diamond$ strategy $\sigma_\Diamond$, which selects the vertical edge at state $(q_k, inc_j)_1$. Since from $(init)_{in}$ the reachability to $\mathsf{T}_{\mathsf{a_j}}$ must be $\frac{1}{6}$, we have that

$$\frac{1}{2} \cdot \frac{1}{2} \cdot \frac{1}{3} + \frac{1}{2} \cdot \frac{1}{2} \cdot \frac{1}{2} \cdot (1 - x) + \frac{1}{2} \cdot \frac{1}{2} \cdot \frac{1}{2} \cdot \frac{1}{3} = \frac{1}{6},$$

   which implies that $x = \frac{2}{3 \cdot 2^0}$, as desired.

   Init+Decrement. We show that at state $(q_k, dec_j)_{in}$, $\sigma^*_\Box$ must choose the edge labelled by "$= 0$." To see this, suppose the opposite, i.e., $\sigma^*_\Box$ chooses the edge labelled by "> 0." The reachability probability to $\mathsf{T}_{\mathsf{b_j}}$ is

$$\frac{1}{2} \cdot \frac{1}{2} \cdot \frac{1}{3} + \frac{1}{2} \cdot \frac{1}{2} \cdot \frac{1}{2} \cdot (\frac{2}{3} + \frac{1}{3} \cdot x) > \frac{1}{6},$$

   which contradicts (b).

   Increment+Increment. The first instruction is $l_h : c_{j'} := c_{j'} + 1$, goto l_k, and the second instruction is $l_k : c_j := c_j + 1$. We show that the probability of selecting edge

(s_5, s_6) for $\sigma^*_\Box$ must be $\frac{2}{3 \cdot 2^{c_j}}$, and the probability for edge (s_7, s_8) must be $x = \frac{2}{3 \cdot 2^{c_i}}$. By (b), from $(q_h, inc_{j'})_2$ the reachability probability to $\mathsf{T}_{\mathsf{a_j}}$ must be $\frac{1}{6}$, which means

$$\frac{1}{2} \cdot \frac{1}{2} \cdot \frac{2}{3 \cdot 2^{c_j}} \cdot \frac{1}{2} + \frac{1}{2} \cdot \frac{1}{2} \cdot \frac{1}{2} \cdot (1 - x) + \frac{1}{2} \cdot \frac{1}{2} \cdot \frac{1}{2} \cdot \frac{1}{3} = \frac{1}{6},$$

yielding that $x = \frac{2}{3 \cdot 2^{c_j}}$, as desired.

Increment+Decrement. The first instruction is $l_h : c_{j'} := c_{j'} + 1$, goto l_k, and the second instruction is $l_k : if\ c_j = 0 \cdots$. If $c_j > 0$ when executing instruction l_k, we show that $\sigma^*_\Box$ must choose the edge labelled by "> 0." Assume that this is not the case, and we immediately have that the probability to reach $\mathsf{T}_{\mathsf{a_j}}$ is

$$\frac{1}{2} \cdot \frac{1}{2} \cdot \frac{2}{3 \cdot 2^{c_j + 1}} \cdot \frac{1}{2} + \frac{1}{2} \cdot \frac{1}{6} < \frac{1}{6},$$

which contradicts (b). Hence, the edge labelled by ">0" has to be taken. Then by (b), from $(q_h, inc_{j'})_2$ the probability to reach $\mathsf{T}_{\mathsf{a_j}}$ must be $\frac{1}{6}$, which gives

$$\frac{1}{2} \cdot \frac{1}{2} \cdot \frac{2}{3 \cdot 2^{c_j}} \cdot \frac{1}{2} + \frac{1}{2} \cdot \frac{1}{2} \cdot \frac{1}{2} \cdot \frac{2}{3} + \frac{1}{2} \cdot \frac{1}{2} \cdot \frac{1}{2} \cdot \frac{1}{3} \cdot (1 - x) + \frac{1}{2} \cdot \frac{1}{2} \cdot \frac{1}{2} \cdot \frac{1}{3} = \frac{1}{6},$$

yielding that the probability of $\sigma^*_\Box$ to select the edge (s_{13}, s_{14}) is $x = \frac{2}{2^{c_j}}$.

For the counter i, again by (b), from $(q_h, inc_{j'})_2$ the probability to reach $\mathsf{T}_{\mathsf{a_j}}$ must be $\frac{1}{6}$, which gives

$$\frac{1}{2} \cdot \frac{1}{2} \cdot \frac{2}{3 \cdot 2^{c_i}} \cdot \frac{1}{2} + \frac{1}{2} \cdot \frac{1}{2} \cdot \frac{1}{2} \cdot \frac{2}{3} + \frac{1}{2} \cdot \frac{1}{2} \cdot \frac{1}{2} \cdot (1 - x) + \frac{1}{2} \cdot \frac{1}{2} \cdot \frac{1}{2} \cdot \frac{1}{3} = \frac{1}{6},$$

yielding that $x = \frac{2}{3 \cdot 2^{c_i}}$, as desired.

If $c_j = 0$ when executing instruction l_k, we have that $\sigma^*_\Box$ must choose the edge labelled by $=0$, by exactly the same argument as for the Init+Decrement case.

Decrement+Decrement. Here we verify two cases: the first case is that $(q_h, dec_{j'})^{=0}_{out}$ is linked with $(q_k, dec_j)_{in}$, which is the same as for the Init+Decrement case; the second case is that $(q_h, dec_{j'})^{>0}_{out}$ is linked with $(q, dec_j)_{in}$, which is the same as for the Increment+Decrement case.

Decrement+Increment. Here we verify two cases: the first case is that $(q_h, dec_{j'})^{=0}_{out}$ is linked with $(q_k, inc_j)_{in}$, which is the same as for the Init+Increment case; the second case is that $(q_h, dec_{j'})^{>0}_{out}$ is linked with $(q_h, inc_{j'})_{in}$, which is the same as for the Increment+Increment case.

5. We are now in a position to show the main claim which establishes the correctness

of the construction, namely, that player $\Box$ has a winning strategy in $\mathcal{G}(\mathcal{M})$ if and only if $\mathcal{M}$ does *not* terminate. We show two directions:

"$\Leftarrow$". Suppose that $\mathcal{M}$ does not terminate, then consider a player $\Box$ strategy $\sigma^*_\Box$ for $\mathcal{G}(\mathcal{M})$. We can safely to pick $\sigma^*_\Box$ such that it follows the counter update (because we have shown in the previous steps that any winning strategy has to do so), i.e., $\sigma^*_\Box$ must perform the following:

- For the Init gadget, at state s_0, the probability of selecting edge (s_0, s_1) is $\frac{2}{3\cdot 2^0}$, and at state s_2, the probability of selecting edge (s_2, s_3) is $\frac{2}{3\cdot 2^0}$.
- For each Increment gadget with index k, if the counter values are c_1 and c_2 respectively, then
  - (s_5, s_6) is chosen with probability $\frac{2}{3\cdot 2^{c_j}}$;
  - (s_7, s_8) is chosen with probability $\frac{2}{3\cdot 2^{c_i}}$;
  - (s_9, s_{10}) is chosen with probability $\frac{2}{3\cdot 2^{c_j+1}}$; and
  - (s_{11}, s_{12}) is chosen with probability $\frac{2}{3\cdot 2^{c_i}}$.
- For each Decrement gadget with index k, suppose the counter values are c_1 and c_2 respectively. Then, if $c_j = 0$, then at state $(q_k, dec_j)_{in}$, $\sigma^*_\Box$ selects the edge labelled with "$= 0$," and if $c_j > 0$, then at state $(q_k, dec_j)_{in}$, $\sigma^*_\Box$ selects the edge labelled with "> 0," and
  - (s_{13}, s_{14}) is chosen with probability $\frac{2}{2^{c_j}}$;
  - (s_{15}, s_{16}) is chosen with probability $\frac{2}{3\cdot 2^{c_i}}$;
  - (s_{17}, s_{18}) is chosen with probability $\frac{2}{3\cdot 2^{c_j-1}}$; and
  - (s_{19}, s_{20}) is chosen with probability $\frac{2}{3\cdot 2^{c_i}}$.

It is not difficult to verify that $\sigma^*_\Box$ achieves the first five objectives (by extending the argument from the proof of infinite memory requirement in Theorem 12). Furthermore, as $\mathcal{M}$ does not terminate, under any $\sigma_\Diamond$, $\mathsf{T_t}$ is reached with probability 1. This is because the only way to reach terminal states a_1, b_1, a_2, b_2 or c, which are not labelled with $\mathsf{T_t}$, is by reaching the Termination gadget with positive probability.

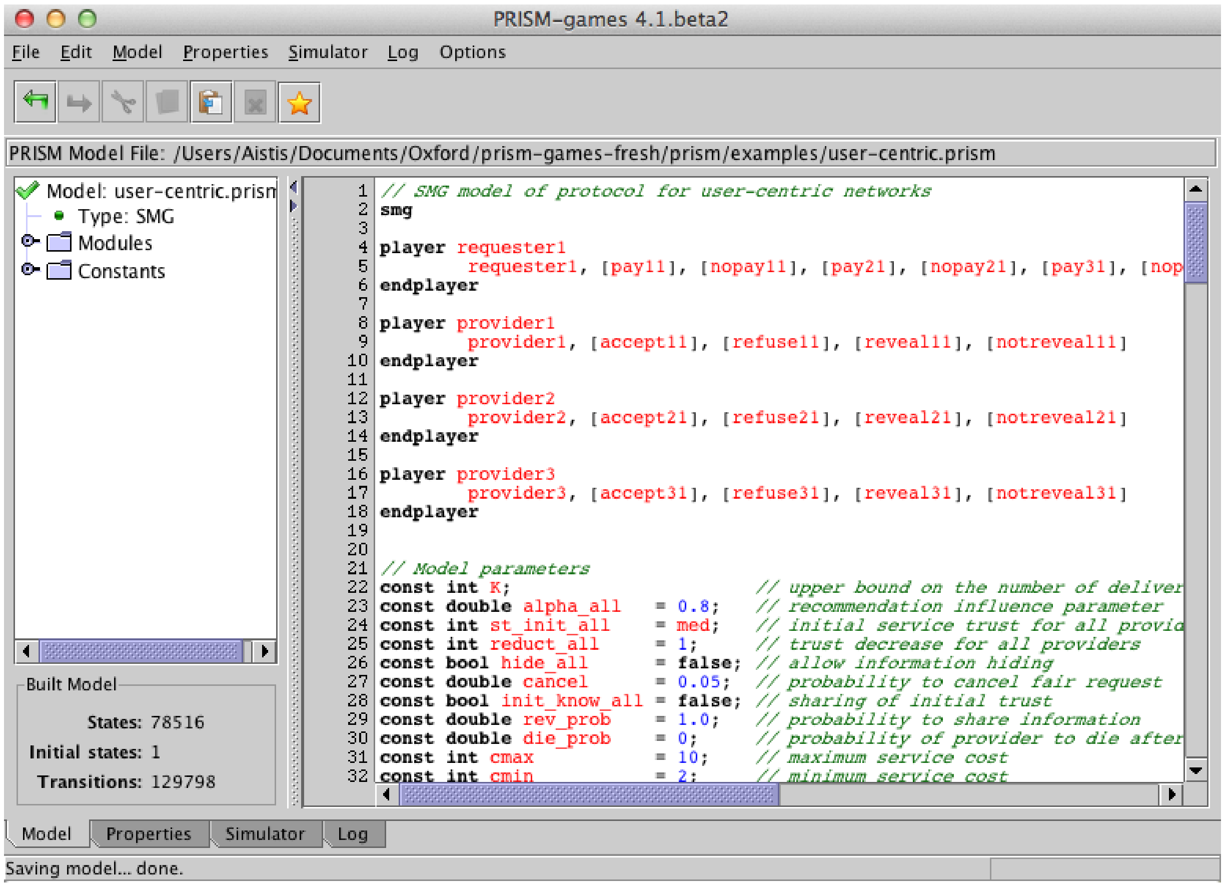
"$\Rightarrow$". For the other direction, suppose that there is a winning player $\Box$ strategy $\sigma^*_\Box$. Then in order to satisfy the first five objectives, $\sigma^*_\Box$ must follow the counter update, as described above. However, in order to satisfy the last objective, i.e. reaching $\mathsf{T_t}$ with probability one, $\sigma^*_\Box$ must ensure that the probability to reach terminals a_1, b_1, a_1, a_2 and c is zero. This is only possible if the Terminal gadget is never reached, implying that $\mathcal{M}$ does not terminate.

This completes the proof.

Appendix E

PRISM-games tool

In this appendix we give a brief overview of the usage of PRISM-games tool described in Chapter 6. For more details about the tool and download instructions please visit `http://www.prismmodelchecker.org/games/`. Here we focus on the features that are distinct from the existing functionality of PRISM [77].

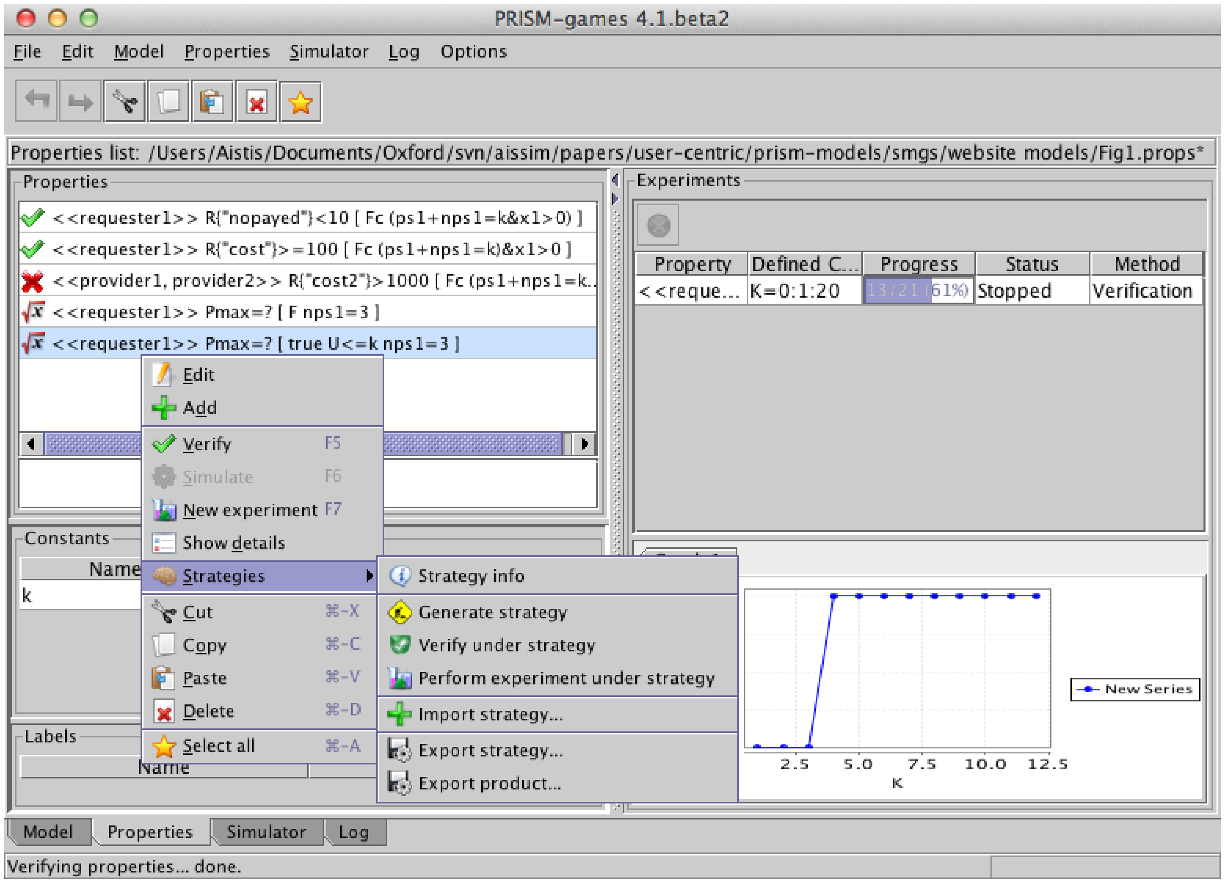

PRISM-games 4.1.beta2
File Edit Model Properties Simulator Log Options
PRISM Model File: /Users/Aistis/Documents/Oxford/prism-games-fresh/prism/examples/user-centric.prism
Model: user-centric.prism
Type: SMG
Modules
Constants
Built Model
States: 78516
Initial states: 1
Transitions: 129798
Model Properties Simulator Log
Saving model... done.

```
// SMG model of protocol for user-centric networks
smg

player requester1
        requester1, [pay11], [nopay11], [pay21], [nopay21], [pay31], [nop
endplayer

player provider1
        provider1, [accept11], [refuse11], [reveal11], [notreveal11]
endplayer

player provider2
        provider2, [accept21], [refuse21], [reveal21], [notreveal21]
endplayer

player provider3
        provider3, [accept31], [refuse31], [reveal31], [notreveal31]
endplayer


// Model parameters
const int K;                                // upper bound on the number of deliver
const double alpha_all    = 0.8;   // recommendation influence parameter
const int st_init_all     = med;   // initial service trust for all provid
const int reduct_all      = 1;     // trust decrease for all providers
const bool hide_all       = false; // allow information hiding
const double cancel       = 0.05;  // probability to cancel fair request
const bool init_know_all  = false; // sharing of initial trust
const double rev_prob     = 1.0;   // probability to share information
const double die_prob     = 0;     // probability of provider to die after
const int cmax            = 10;    // maximum service cost
const int cmin            = 2;     // minimum service cost
```

Figure E.1: PRISM-games model window.

The main modelling window is shown in Figure E.1, where SMGs have to be specified using the modelling language described in Section 6.1. The screenshot shows the specification of the protocol for user-centric networks analysed in Section 6.6.3.

The property analysis window is shown in Figure E.2. We can see several rPATL properties listed in the syntax provided in Section 6.2. The screenshot also shows the strategy menu, which provides access to the strategy synthesis functionality of PRISM-games. We describe the available functionality below.



Properties list: /Users/Aistis/Documents/Oxford/svn/aissim/papers/user-centric/prism-models/smgs/website models/Fig1.props*

✓ <<requester1>> R{"nopayed"}<10 [Fc (ps1+nps1=k&x1>0)]
✓ <<requester1>> R{"cost"}>=100 [Fc (ps1+nps1=k)&x1>0]
✗ <<provider1, provider2>> R{"cost2"}>1000 [Fc (ps1+nps1=k..
<<requester1>> Pmax=? [F nps1=3]
<<requester1>> Pmax=? [true U<=k nps1=3]

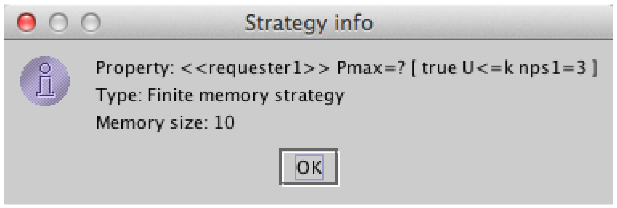
Property	Defined C...	Progress	Status	Method
<<reque...	K=0:1:20	13/21 (61%)	Stopped	Verification

Name
k

Verifying properties... done.

Figure E.2: PRISM-games properties window.

- *Strategy info* - displays the information about the strategy currently held in memory of the model checker (e.g., see Figure E.3).



Property: <<requester1>> Pmax=? [true U<=k nps1=3]
Type: Finite memory strategy
Memory size: 10

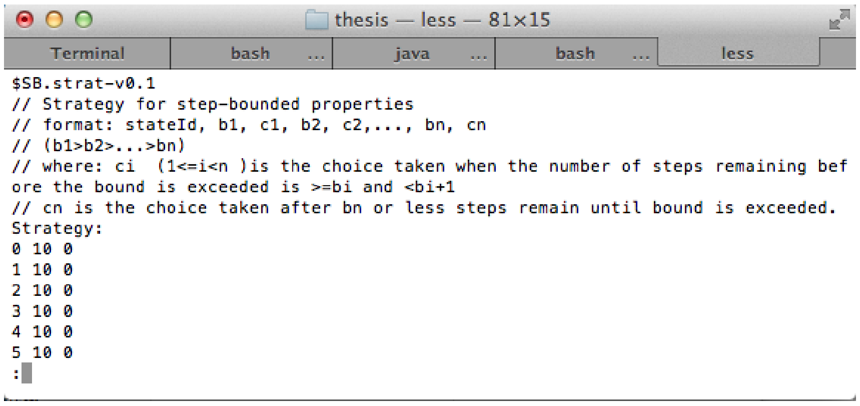
Figure E.3: Strategy information window.

- *Generate strategy* - verifies the property and generates the strategy for the selected property and the model. The optimal strategy is generated for all players in the game: for the players in coalition it contains optimal actions to satisfy the property, and actions for the players outside the coalition are the ones trying to falsify the property. After generation, these strategies can be explored in the simulator window discussed later.

- *Verify under strategy* - performs strategy 'implementation' discussed in Section 6.4. In order to use the feature, the strategy needs to be generated using the *Generate strategy* option beforehand. When invoked, the action fixes the choices of the coalition players to the ones provided by the strategy (i.e., the product of the SMG and the strategy is built), and the selected property is then verified on the product game.

- *Perform experiment under strategy* - similarly to the *Verify under strategy* option, the product of the strategy, that has been generated, and the game is built, but, in this case, we can verify the property for a range of parameter values, e.g., we can verify the property `<<requester1>> Pmax=?  [true U<=k nps1=3]` for $k \in \{1, \ldots, 12\}$, and plot the resulting values (see graph in Figure E.2).

- *Import strategy/Export strategy* - imports/exports strategy from/to a given file. Accepted file formats for different strategy types are given at the beginning of the export file, e.g., see Figure E.4 for a sample file header describing strategy format for the step-bounded until formula.

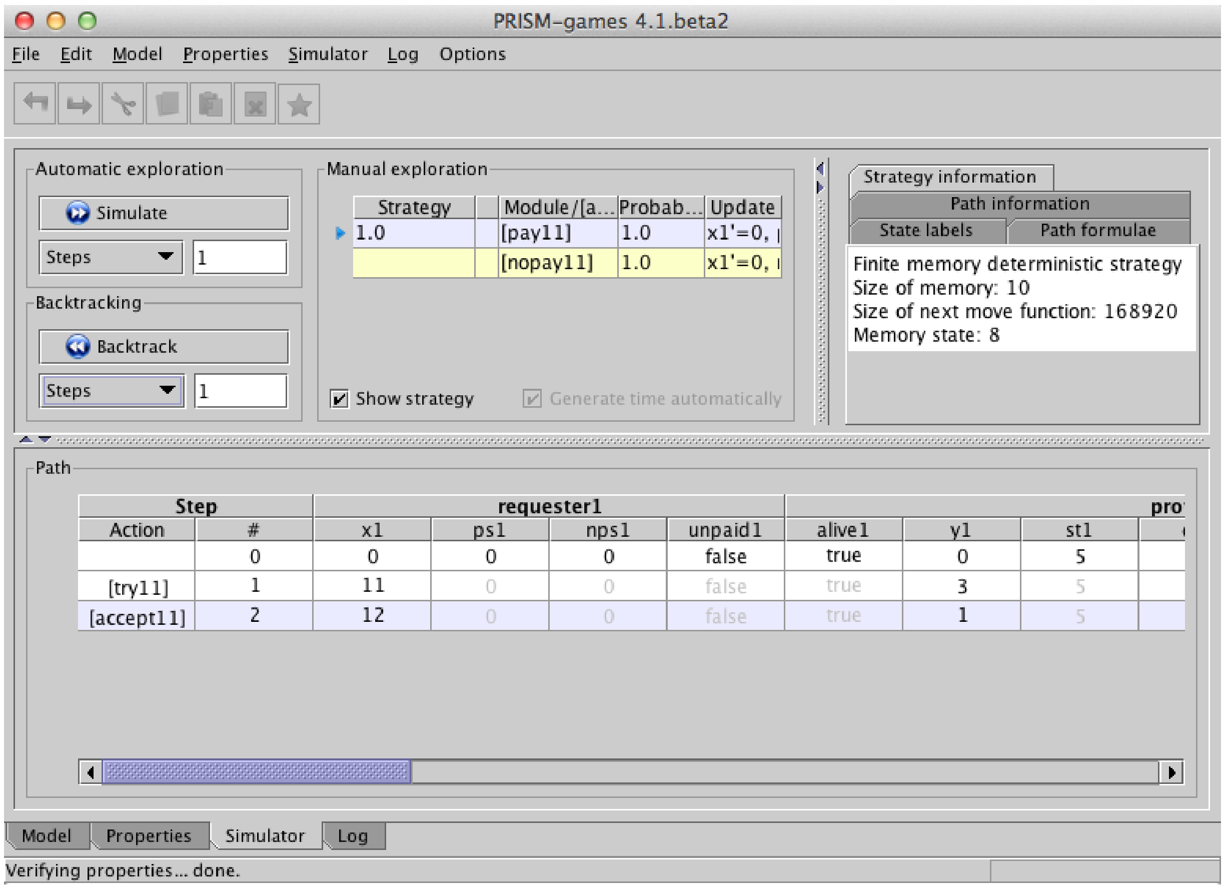


```
$SB.strat-v0.1
// Strategy for step-bounded properties
// format: stateId, b1, c1, b2, c2,..., bn, cn
// (b1>b2>...>bn)
// where: ci  (1<=i<n )is the choice taken when the number of steps remaining bef
ore the bound is exceeded is >=bi and <bi+1
// cn is the choice taken after bn or less steps remain until bound is exceeded.
Strategy:
0 10 0
1 10 0
2 10 0
3 10 0
4 10 0
5 10 0
:
```

Figure E.4: PRISM-games strategy import/export format.

- *Export product* - exports the SMG model, where the choices of the coalition players are fixed according to the strategy that has been generated. This is the same product game that is constructed when using *Verify under strategy* feature.

Finally, we describe the PRISM-games functionality available from the simulator window shown in Figure E.5. After the strategy has been generated using the *Generate strategy* option from the strategy menu described earlier, one can explore it in the simulator window. Strategy choices are displayed in the 'Manual exploration' section, e.g., in Figure E.5 we can see that the optimal strategy chooses action '`[pay11]`' with probability 1. We can step through the model following the choices suggested to manually explore the strategy (we used this functionality to construct the strategy graphs for the case study of the user-centric network protocol shown in Figures 6.9c, 6.10b and 6.11c). Also, the 'Strategy information' panel on the right displays strategy information and the current state (i.e., memory element).



Automatic exploration: Steps 1
Backtracking: Steps 1

Manual exploration:

Strategy		Module/[a...	Probab...	Update
1.0		[pay11]	1.0	x1'=0,
		[nopay11]	1.0	x1'=0,

[✓ Show strategy]

Strategy information:

Finite memory deterministic strategy
Size of memory: 10
Size of next move function: 168920
Memory state: 8

Path:

Step		requester1							pro
Action	#	x1	ps1	nps1	unpaid1	alive1	y1	st1	
	0	0	0	0	false	true	0	5	
[try11]	1	11	0	0	false	true	3	5	
[accept11]	2	12	0	0	false	true	1	5	

Verifying properties... done.

Figure E.5: PRISM-games simulator window.

Appendix F

PRISM-games models

F.1 Team formation protocol

In this section we present a sample three-agent PRISM-games SMG model of a team formation protocol presented in Section 3.5. The full set of models can be found in `http://www.prismmodelchecker.org/files/clima11/`.

```
smg

// parameters
const int n_resources = 3;
const int n_tasks = 2;
const int n_sensors = 3;

// sensor resources
const int resource1=1;
const int resource2=2;
const int resource3=3;
const int resource4=1;

// network configuration
const int e12=1;
const int e13=1;

const int e21=e12;
const int e23=1;

const int e31=e13;
const int e32=e23;

player p0 controller, [str1], [str2], [str3] endplayer
module controller // schedules the algorithm

        // algorithm status
        status : [0..7];

        // task resource indicator variables
        t1_r1 : [0..1];
        t1_r2 : [0..1];
        t1_r3 : [0..1];

        t2_r1 : [0..1];
        t2_r2 : [0..1];
        t2_r3 : [0..1];

        // schedule placeholders
        turn1 : [0..n_sensors];
        turn2 : [0..n_sensors];
        turn3 : [0..n_sensors];

        // selecting schedule uniformly at random
        [] status=0 -> 1/6 : (turn1'=1) & (turn2'=2) & (turn3'=3) & (status'=1)
                + 1/6 : (turn1'=1) & (turn2'=3) & (turn3'=2) & (status'=1)
                + 1/6 : (turn1'=2) & (turn2'=1) & (turn3'=3) & (status'=1)
                + 1/6 : (turn1'=2) & (turn2'=3) & (turn3'=1) & (status'=1)
                + 1/6 : (turn1'=3) & (turn2'=1) & (turn3'=2) & (status'=1)
                + 1/6 : (turn1'=3) & (turn2'=2) & (turn3'=1) & (status'=1);


        // initialising non-empty tasks uniformly at random
        [] status=1 -> 1/49 : (t1_r1'=0) & (t1_r2'=0) & (t1_r3'=1) & (t2_r1'=0) & (t2_r2'=0) & (t2_r3'=1) & (status'=2)
                + 1/49 : (t1_r1'=0) & (t1_r2'=0) & (t1_r3'=1) & (t2_r1'=0) & (t2_r2'=1) & (t2_r3'=0) & (status'=2)
                + 1/49 : (t1_r1'=0) & (t1_r2'=0) & (t1_r3'=1) & (t2_r1'=0) & (t2_r2'=1) & (t2_r3'=1) & (status'=2)
                + 1/49 : (t1_r1'=0) & (t1_r2'=0) & (t1_r3'=1) & (t2_r1'=1) & (t2_r2'=0) & (t2_r3'=0) & (status'=2)
                + 1/49 : (t1_r1'=0) & (t1_r2'=0) & (t1_r3'=1) & (t2_r1'=1) & (t2_r2'=0) & (t2_r3'=1) & (status'=2)
                + 1/49 : (t1_r1'=0) & (t1_r2'=0) & (t1_r3'=1) & (t2_r1'=1) & (t2_r2'=1) & (t2_r3'=0) & (status'=2)
                + 1/49 : (t1_r1'=0) & (t1_r2'=0) & (t1_r3'=1) & (t2_r1'=1) & (t2_r2'=1) & (t2_r3'=1) & (status'=2)
                + 1/49 : (t1_r1'=0) & (t1_r2'=1) & (t1_r3'=0) & (t2_r1'=0) & (t2_r2'=0) & (t2_r3'=1) & (status'=2)
                + 1/49 : (t1_r1'=0) & (t1_r2'=1) & (t1_r3'=0) & (t2_r1'=0) & (t2_r2'=1) & (t2_r3'=0) & (status'=2)
                + 1/49 : (t1_r1'=0) & (t1_r2'=1) & (t1_r3'=0) & (t2_r1'=0) & (t2_r2'=1) & (t2_r3'=1) & (status'=2)
                + 1/49 : (t1_r1'=0) & (t1_r2'=1) & (t1_r3'=0) & (t2_r1'=1) & (t2_r2'=0) & (t2_r3'=0) & (status'=2)
                + 1/49 : (t1_r1'=0) & (t1_r2'=1) & (t1_r3'=0) & (t2_r1'=1) & (t2_r2'=0) & (t2_r3'=1) & (status'=2)
                + 1/49 : (t1_r1'=0) & (t1_r2'=1) & (t1_r3'=0) & (t2_r1'=1) & (t2_r2'=1) & (t2_r3'=0) & (status'=2)
                + 1/49 : (t1_r1'=0) & (t1_r2'=1) & (t1_r3'=0) & (t2_r1'=1) & (t2_r2'=1) & (t2_r3'=1) & (status'=2)
                + 1/49 : (t1_r1'=0) & (t1_r2'=1) & (t1_r3'=1) & (t2_r1'=0) & (t2_r2'=0) & (t2_r3'=1) & (status'=2)
                + 1/49 : (t1_r1'=0) & (t1_r2'=1) & (t1_r3'=1) & (t2_r1'=0) & (t2_r2'=1) & (t2_r3'=0) & (status'=2)
                + 1/49 : (t1_r1'=0) & (t1_r2'=1) & (t1_r3'=1) & (t2_r1'=0) & (t2_r2'=1) & (t2_r3'=1) & (status'=2)
                + 1/49 : (t1_r1'=0) & (t1_r2'=1) & (t1_r3'=1) & (t2_r1'=1) & (t2_r2'=0) & (t2_r3'=0) & (status'=2)
                + 1/49 : (t1_r1'=0) & (t1_r2'=1) & (t1_r3'=1) & (t2_r1'=1) & (t2_r2'=0) & (t2_r3'=1) & (status'=2)
                + 1/49 : (t1_r1'=0) & (t1_r2'=1) & (t1_r3'=1) & (t2_r1'=1) & (t2_r2'=1) & (t2_r3'=0) & (status'=2)
                + 1/49 : (t1_r1'=0) & (t1_r2'=1) & (t1_r3'=1) & (t2_r1'=1) & (t2_r2'=1) & (t2_r3'=1) & (status'=2)
                + 1/49 : (t1_r1'=1) & (t1_r2'=0) & (t1_r3'=0) & (t2_r1'=0) & (t2_r2'=0) & (t2_r3'=1) & (status'=2)
                + 1/49 : (t1_r1'=1) & (t1_r2'=0) & (t1_r3'=0) & (t2_r1'=0) & (t2_r2'=1) & (t2_r3'=0) & (status'=2)
                + 1/49 : (t1_r1'=1) & (t1_r2'=0) & (t1_r3'=0) & (t2_r1'=0) & (t2_r2'=1) & (t2_r3'=1) & (status'=2)
                + 1/49 : (t1_r1'=1) & (t1_r2'=0) & (t1_r3'=0) & (t2_r1'=1) & (t2_r2'=0) & (t2_r3'=0) & (status'=2)
                + 1/49 : (t1_r1'=1) & (t1_r2'=0) & (t1_r3'=0) & (t2_r1'=1) & (t2_r2'=0) & (t2_r3'=1) & (status'=2)
                + 1/49 : (t1_r1'=1) & (t1_r2'=0) & (t1_r3'=0) & (t2_r1'=1) & (t2_r2'=1) & (t2_r3'=0) & (status'=2)
                + 1/49 : (t1_r1'=1) & (t1_r2'=0) & (t1_r3'=0) & (t2_r1'=1) & (t2_r2'=1) & (t2_r3'=1) & (status'=2)
                + 1/49 : (t1_r1'=1) & (t1_r2'=0) & (t1_r3'=1) & (t2_r1'=0) & (t2_r2'=0) & (t2_r3'=1) & (status'=2)
                + 1/49 : (t1_r1'=1) & (t1_r2'=0) & (t1_r3'=1) & (t2_r1'=0) & (t2_r2'=1) & (t2_r3'=0) & (status'=2)
                + 1/49 : (t1_r1'=1) & (t1_r2'=0) & (t1_r3'=1) & (t2_r1'=0) & (t2_r2'=1) & (t2_r3'=1) & (status'=2)
                + 1/49 : (t1_r1'=1) & (t1_r2'=0) & (t1_r3'=1) & (t2_r1'=1) & (t2_r2'=0) & (t2_r3'=0) & (status'=2)
                + 1/49 : (t1_r1'=1) & (t1_r2'=0) & (t1_r3'=1) & (t2_r1'=1) & (t2_r2'=0) & (t2_r3'=1) & (status'=2)
```

```
                 + 1/49 : (t1_r1'=1) & (t1_r2'=0) & (t1_r3'=1) & (t2_r1'=1) & (t2_r2'=1) & (t2_r3'=0) & (status'=2)
                 + 1/49 : (t1_r1'=1) & (t1_r2'=0) & (t1_r3'=1) & (t2_r1'=1) & (t2_r2'=1) & (t2_r3'=1) & (status'=2)
                 + 1/49 : (t1_r1'=1) & (t1_r2'=1) & (t1_r3'=0) & (t2_r1'=0) & (t2_r2'=0) & (t2_r3'=1) & (status'=2)
                 + 1/49 : (t1_r1'=1) & (t1_r2'=1) & (t1_r3'=0) & (t2_r1'=0) & (t2_r2'=1) & (t2_r3'=0) & (status'=2)
                 + 1/49 : (t1_r1'=1) & (t1_r2'=1) & (t1_r3'=0) & (t2_r1'=0) & (t2_r2'=1) & (t2_r3'=1) & (status'=2)
                 + 1/49 : (t1_r1'=1) & (t1_r2'=1) & (t1_r3'=0) & (t2_r1'=1) & (t2_r2'=0) & (t2_r3'=0) & (status'=2)
                 + 1/49 : (t1_r1'=1) & (t1_r2'=1) & (t1_r3'=0) & (t2_r1'=1) & (t2_r2'=0) & (t2_r3'=1) & (status'=2)
                 + 1/49 : (t1_r1'=1) & (t1_r2'=1) & (t1_r3'=0) & (t2_r1'=1) & (t2_r2'=1) & (t2_r3'=0) & (status'=2)
                 + 1/49 : (t1_r1'=1) & (t1_r2'=1) & (t1_r3'=0) & (t2_r1'=1) & (t2_r2'=1) & (t2_r3'=1) & (status'=2)
                 + 1/49 : (t1_r1'=1) & (t1_r2'=1) & (t1_r3'=1) & (t2_r1'=0) & (t2_r2'=0) & (t2_r3'=1) & (status'=2)
                 + 1/49 : (t1_r1'=1) & (t1_r2'=1) & (t1_r3'=1) & (t2_r1'=0) & (t2_r2'=1) & (t2_r3'=0) & (status'=2)
                 + 1/49 : (t1_r1'=1) & (t1_r2'=1) & (t1_r3'=1) & (t2_r1'=0) & (t2_r2'=1) & (t2_r3'=1) & (status'=2)
                 + 1/49 : (t1_r1'=1) & (t1_r2'=1) & (t1_r3'=1) & (t2_r1'=1) & (t2_r2'=0) & (t2_r3'=0) & (status'=2)
                 + 1/49 : (t1_r1'=1) & (t1_r2'=1) & (t1_r3'=1) & (t2_r1'=1) & (t2_r2'=0) & (t2_r3'=1) & (status'=2)
                 + 1/49 : (t1_r1'=1) & (t1_r2'=1) & (t1_r3'=1) & (t2_r1'=1) & (t2_r2'=1) & (t2_r3'=0) & (status'=2)
                 + 1/49 : (t1_r1'=1) & (t1_r2'=1) & (t1_r3'=1) & (t2_r1'=1) & (t2_r2'=1) & (t2_r3'=1) & (status'=2);

        // executing the schedule

        // 1st round
        [str1] status=2 & turn1=1 -> (status'=2);
        [fin1] status=2 & turn1=1 -> (status'=3);
        [str2] status=2 & turn1=2 -> (status'=2);
        [fin2] status=2 & turn1=2 -> (status'=3);
        [str3] status=2 & turn1=3 -> (status'=2);
        [fin3] status=2 & turn1=3 -> (status'=3);

        // 2nd round
        [str1] status=3 & turn2=1 -> (status'=3);
        [fin1] status=3 & turn2=1 -> (status'=4);
        [str2] status=3 & turn2=2 -> (status'=3);
        [fin2] status=3 & turn2=2 -> (status'=4);
        [str3] status=3 & turn2=3 -> (status'=3);
        [fin3] status=3 & turn2=3 -> (status'=4);

        // 3rd round
        [str1] status=4 & turn3=1 -> (status'=4);
        [fin1] status=4 & turn3=1 -> (status'=5);
        [str2] status=4 & turn3=2 -> (status'=4);
        [fin2] status=4 & turn3=2 -> (status'=5);
        [str3] status=4 & turn3=3 -> (status'=4);
        [fin3] status=4 & turn3=3 -> (status'=5);

        [] status=5 -> (status'=6);
        [] status=6 -> true;

endmodule

player p1 sensor1, [fin1] endplayer
module sensor1

        state1 : [0..3];

        // team membership indicators
        m1_t1 : [0..1];
        m1_t2 : [0..1];

        // task scheduling
        turn1_1 : [0..n_tasks];
        turn2_1 : [0..n_tasks];



        // starting turn, selecting order of tasks
        [str1] state1=0 -> (state1'=1);

        // if there is no team and has required skill - initiating the team
        [] state1=1 & !committed & team_size_t1=0 & has_resource_t1 -> (m1_t1'=1);
        [] state1=1 & !committed & team_size_t2=0 & has_resource_t2 -> (m1_t2'=1);

        // if team already exists and one of the neighbours is in it - joining the team
        [] state1=1 & !committed & team_size_t1>0 & can_join_t1 & has_resource_t1 & !resource_filled_t1 -> (m1_t1'=1);
        [] state1=1 & !committed & team_size_t2>0 & can_join_t2 & has_resource_t2 & !resource_filled_t2 -> (m1_t2'=1);

        [fin1] state1>0 -> (state1'=0);

endmodule

player p2 sensor2, [fin2] endplayer
module sensor2 = sensor1
[
        state1=state2,

        str1=str2,
        fin1=fin2,

        m1_t1=m2_t1,
        m1_t2=m2_t2,

        m2_t1=m1_t1,
        m2_t2=m1_t2,

        turn1_1=turn1_2,
        turn2_1=turn2_2,

        resource1=resource2,
        resource2=resource1,

        e12=e21,
        e13=e23,
        e14=e24,
        e15=e25,

        e21=e12,
        e23=e13,
        e24=e14,
        e25=e15
]
endmodule

player p3 sensor3, [fin3] endplayer
module sensor3 = sensor1
[
        state1=state3,

        str1=str3,
        fin1=fin3,

        m1_t1=m3_t1,
        m1_t2=m3_t2,
        m3_t1=m1_t1,
        m3_t2=m1_t2,
        turn1_1=turn1_3,
        turn2_1=turn2_3,

        resource1=resource3,
        resource3=resource1,

        e12=e32,
        e13=e31,
        e14=e34,
        e15=e35,

        e31=e13,
        e32=e12,
        e34=e14,
        e35=e15
]
endmodule

// agent is committed to some team
formula committed = (m1_t1+m1_t2) > 0;

// formulae to compute team sizes
formula team_size_t1 = m1_t1+m2_t1+m3_t1;
formula team_size_t2 = m1_t2+m2_t2+m3_t2;

// formulae to check whether the agent can join the team
formula can_join_t1 = e12*m2_t1 + e13*m3_t1 > 0;
formula can_join_t2 = e12*m2_t2 + e13*m3_t2  > 0;

// formulae to check whether agent has the resource required by the task
formula has_resource_t1 = ( (t1_r1=1&resource1=1) | (t1_r2=1&resource1=2) | (t1_r3=1&resource1=3) );
formula has_resource_t2 = ( (t2_r1=1&resource1=1) | (t2_r2=1&resource1=2) | (t2_r3=1&resource1=3) );

// formulae to check whether the resource of an agent has been already filled in the team
formula resource_filled_t1 = (m2_t1=1 & resource1=resource2) | (m3_t1=1 & resource1=resource3);
formula resource_filled_t2 = (m2_t2=1 & resource1=resource2) | (m3_t2=1 & resource1=resource3);

// formula to compute team initiation probability (assuming each agent has at least one connection)
formula IP = (e12*(1-((m2_t1+m2_t2)=0?0:1))+e13*(1-((m3_t1+m3_t2)=0?0:1))) / (e12+e13);

// labels and formulae for property specification
formula finished = (status=5);


formula task1_completed = finished
                         & ((t1_r1=1)=>((m1_t1=1&resource1=1)|(m2_t1=1&resource2=1)|(m3_t1=1&resource3=1)))
                         & ((t1_r2=1)=>((m1_t1=1&resource1=2)|(m2_t1=1&resource2=2)|(m3_t1=1&resource3=2)))
                         & ((t1_r3=1)=>((m1_t1=1&resource1=3)|(m2_t1=1&resource2=3)|(m3_t1=1&resource3=3)));

formula task2_completed = finished
                         & ((t2_r1=1)=>((m1_t2=1&resource1=1)|(m2_t2=1&resource2=1)|(m3_t2=1&resource3=1)))
                         & ((t2_r2=1)=>((m1_t2=1&resource1=2)|(m2_t2=1&resource2=2)|(m3_t2=1&resource3=2)))
```

```
                              & ((t2_r3=1)=>((m1_t2=1&resource1=3)|(m2_t2=1&resource2=3)|(m3_t2=1&resource3=3)));

formula agent1_joins_successful_team = (task1_completed & m1_t1=1) | (task2_completed & m1_t2=1);

formula agent2_joins_successful_team = (task1_completed & m2_t1=1) | (task2_completed & m2_t2=1);

formula agent3_joins_successful_team = (task1_completed & m3_t1=1) | (task2_completed & m3_t2=1);

// rewards
rewards "w_1_total"
    agent1_joins_successful_team : 1;
    agent2_joins_successful_team : 1;
    agent3_joins_successful_team : 1;
endrewards

rewards "w_2_total"
    task1_completed : 1;
    task2_completed : 1;
endrewards
```

F.2 Microgrid demand-side management protocol

In this section we present a sample three-household PRISM-games SMG model of a micro-grid demand-side management protocol presented in Section 6.6.1. The full set of models can be found in `http://www.prismmodelchecker.org/files/fmsd-smg/`.

```
smg

// number of households
const int N = 3;

// number of days
const int D = 3;

// number of time intervals in the day
const int K = 16;

// expected number of jobs per household per day
const int Exp_J = 9;

// cost limits for households
const double price_limit = 1.5;

// initiation probabilities for jobs (uuniform distribution)
const double P_J1 = 1/4;
const double P_J2 = 1/4;
const double P_J3 = 1/4;
const double P_J4 = 1/4;

// probability of starting a task independently of the cost
const double P_start = 0.8;

// distribution of the expected demand across intervals
const double D_K1 = 0.0614;
const double D_K2 = 0.0392;
const double D_K3 = 0.0304;
const double D_K4 = 0.0304;
const double D_K5 = 0.0355;
const double D_K6 = 0.0518;
const double D_K7 = 0.0651;
const double D_K8 = 0.0643;
const double D_K9 = 0.0625;
const double D_K10 = 0.0618;
const double D_K11 = 0.0614;
const double D_K12 = 0.0695;
const double D_K13 = 0.0887;
const double D_K14 = 0.1013;
const double D_K15 = 0.1005;
const double D_K16 = 0.0762;

// time limit
const int max_time = K*D+1;

// ----------------------------------------------------

// time counter
global time : [1..max_time];

// jobs of households
global job1 : [0..4];
global job2 : [0..4];
global job3 : [0..4];

// scheduling variable
global sched : [0..N];


player p0
        player0
endplayer

// definition of scheduling module
module player0

        [] sched = 0 -> 1/N : (sched'=1) + 1/N : (sched'=2) + 1/N : (sched'=3);

endmodule

// definitions of households
player p1
        player1, [start1], [backoff1], [nbackoff1]
endplayer

module player1

        job_arrived1 : [0..4];

        [] sched=1 & active = 0 & job1 > 0 & time < max_time -> (job1'=new_j1) & (job2'=new_j2) & (job3'=new_j3) & (time'=time+1) & (sched'=0)

        // initiate the job with probability P_init
        [] sched=1 & active = 0 & job1 = 0 & time < max_time -> P_init*P_J1 : (job_arrived1'=1)
                                                     + P_init*P_J2 : (job_arrived1'=2)
                                                     + P_init*P_J3 : (job_arrived1'=3)
                                                     + P_init*P_J4 : (job_arrived1'=4)
                                                     + (1-P_init) : (job1'=new_j1) & (job2'=new_j2) & (job3'=new_j3) & (time'=time+1) &

        // start job if cost below the limit
        [start1] sched=1 & job_arrived1 > 0 & price <= price_limit & time < max_time-> (job1'=job_arrived1)  & (job2'=new_j2) & (job3'=new_j3)

        // back-off with probability 1-P_start
        [backoff1] sched=1 & job_arrived1 > 0 & price > price_limit & time < max_time->   P_start : (job1'=job_arrived1)  & (job2'=new_j2) & (
                                              + (1-P_start) : (job1'=new_j1) & (job2'=new_j2) & (job3'=new_j3) & (job_arrived1'=0) & (tim

        // don't back-off
        [nbackoff1] sched=1 & job_arrived1 > 0 & price > price_limit & time < max_time -> (job1'=job_arrived1)  & (job2'=new_j2) & (job3'=new_

        // finished
        [] sched=1 & time=max_time -> (job1'=new_j1) & (job2'=new_j2) & (job3'=new_j3) & (sched'=0);
```

```
endmodule

player p2
        player2, [start2], [backoff2], [nbackoff2]
endplayer

module player2

        job_arrived2 : [0..4];

        [] sched=2 & active = 0 & job2 > 0 & time < max_time -> (job1'=new_j1) & (job2'=new_j2) & (job3'=new_j3) & (time'=time+1) & (sched'=0)

        // initiate the job with probability P_init
        [] sched=2 & active = 0 & job2 = 0 & time < max_time -> P_init*P_J1 : (job_arrived2'=1)
                                                            + P_init*P_J2 : (job_arrived2'=2)
                                                            + P_init*P_J3 : (job_arrived2'=3)
                                                            + P_init*P_J4 : (job_arrived2'=4)
                                                            + (1-P_init) : (job1'=new_j1) & (job2'=new_j2) & (job3'=new_j3) & (time'=time+1) &

        // start job if cost below the limit
        [start2] sched=2 & job_arrived2 > 0 & price <= price_limit & time < max_time-> (job2'=job_arrived2)  & (job1'=new_j1) & (job3'=new_j3)

        // back-off with probability 1-P_start
        [backoff2] sched=2 & job_arrived2 > 0 & price > price_limit & time < max_time->   P_start : (job2'=job_arrived2)  & (job1'=new_j1) & (
                                                   + (1-P_start) : (job1'=new_j1) & (job2'=new_j2) & (job3'=new_j3) & (job_arrived2'=0) & (tim

        // don't back-off
        [nbackoff2] sched=2 & job_arrived2 > 0 & price > price_limit & time < max_time -> (job2'=job_arrived2)  & (job1'=new_j1) & (job3'=new_

        // finished
        [] sched=2 & time=max_time -> (job1'=new_j1) & (job2'=new_j2) & (job3'=new_j3) & (sched'=0);

endmodule

player p3
        player3, [start3], [backoff3], [nbackoff3]
endplayer

module player3

        job_arrived3 : [0..4];

        [] sched=3 & active = 0 & job3 > 0 & time < max_time -> (job1'=new_j1) & (job2'=new_j2) & (job3'=new_j3) & (time'=time+1) & (sched'=0)

        // initiate the job with probability P_init
        [] sched=3 & active = 0 & job3 = 0 & time < max_time -> P_init*P_J1 : (job_arrived3'=1)
                                                            + P_init*P_J2 : (job_arrived3'=2)
                                                            + P_init*P_J3 : (job_arrived3'=3)
                                                            + P_init*P_J4 : (job_arrived3'=4)
                                                            + (1-P_init) : (job1'=new_j1) & (job2'=new_j2) & (job3'=new_j3) & (time'=time+1) &

        // start job if cost below the limit
        [start3] sched=3 & job_arrived3 > 0 & price <= price_limit & time < max_time-> (job3'=job_arrived3)  & (job1'=new_j1) & (job2'=new_j2)

        // back-off with probability 1-P_start
        [backoff3] sched=3 & job_arrived3 > 0 & price > price_limit & time < max_time->   P_start : (job3'=job_arrived3)  & (job1'=new_j1) & (
                                                   + (1-P_start) : (job1'=new_j1) & (job2'=new_j2) & (job3'=new_j3) & (job_arrived3'=0) & (tim

        // don't back-off
        [nbackoff3] sched=3 & job_arrived3 > 0 & price > price_limit & time < max_time -> (job3'=job_arrived3)  & (job1'=new_j1) & (job2'=new_

        // finished
        [] sched=3 & time=max_time -> (job1'=new_j1) & (job2'=new_j2) & (job3'=new_j3) & (sched'=0);

endmodule




// probability to initiate the load
formula P_init = Exp_J *
                 (mod(time,K) = 1 ? D_K1 :
                 (mod(time,K) = 2 ? D_K2 :
                 (mod(time,K) = 3 ? D_K3 :
                 (mod(time,K) = 4 ? D_K4 :
                 (mod(time,K) = 5 ? D_K5 :
                 (mod(time,K) = 6 ? D_K6 :
                 (mod(time,K) = 7 ? D_K7 :
                 (mod(time,K) = 8 ? D_K8 :
                 (mod(time,K) = 9 ? D_K9 :
                 (mod(time,K) = 10 ? D_K10 :
                 (mod(time,K) = 11 ? D_K11 :
                 (mod(time,K) = 12 ? D_K12 :
                 (mod(time,K) = 13 ? D_K13 :
                 (mod(time,K) = 14 ? D_K14 :
                 (mod(time,K) = 15 ? D_K15 :
                 D_K16)))))))))))))));

// formula to compute current cost
formula jobs_running = (job1>0?1:0)  + (job2>0?1:0) + (job3>0?1:0);

// formula to identify say that only one agent is active
formula active = job_arrived1 + job_arrived2  + job_arrived3 ;

// formula to update job status
formula new_j1 = job1=0?0:job1-1;
formula new_j2 = job2=0?0:job2-1;
formula new_j3 = job3=0?0:job3-1;

formula price = jobs_running+1;

rewards "cost"
        sched!=0 : jobs_running*jobs_running;
endrewards
```

```
rewards "tasks_started"
        sched!=0 & job1=1 : 1;
        sched!=0 & job2=1 : 1;
        sched!=0 & job3=1 : 1;
endrewards

rewards "value1"
        sched!=0 & job1>0 : 1/jobs_running;
endrewards
rewards "value12"
        sched!=0 & (job1>0|job2>0) : 1/jobs_running;
endrewards
rewards "value123"
        sched!=0 & (job1>0|job2>0|job3>0) : 1/jobs_running;
endrewards

rewards "common_value"
        sched!=0 : jobs_running=0?0:1/jobs_running;
endrewards

rewards "upfront_cost1"

        [start1] true : 1/(job_arrived1*price);
        [backoff1] true : P_start/(job_arrived1*price);
        [nbackoff1] true : 1/(job_arrived1*price);

endrewards

rewards "upfront_tcost"

        [start1] true : 1/(job_arrived1*price);
        [backoff1] true : P_start/(job_arrived1*price);
        [nbackoff1] true : 1/(job_arrived1*price);

        [start2] true : 1/(job_arrived2*price);
        [backoff2] true : P_start/(job_arrived2*price);
        [nbackoff2] true : 1/(job_arrived2*price);

        [start3] true : 1/(job_arrived3*price);
        [backoff3] true : P_start/(job_arrived3*price);
        [nbackoff3] true : 1/(job_arrived3*price);

endrewards
```

F.3 Collective decision making protocol for sensor networks

In this section we present a sample three-sensor PRISM-games SMG model of a collective decision making algorithm for sensor networks presented in Section 6.6.2. The full set of models can be found in `http://www.prismmodelchecker.org/files/fmsd-smg/`.

```
smg

// number of agents
const int N = 3;

// number of sites
const int K = 3;

// number of confidence levels
const int L = 2;

// model parameters
const double Pexp;
const double eta;
const double gamma;
const double lambda;

// quality of the sites
const double Q1;
const double Q2;
const double Q3;

// confidence levels of agents
global confidence1 : [1..L];
global confidence2 : [1..L];
global confidence3 : [1..L];

// site preferences of agents
global preference1 : [0..K] init 0;
global preference2 : [0..K] init 0;
global preference3 : [0..K] init 0;


// scheduling variable
global sched : [0..N];

// scheduler module
module player0

        [] sched = 0 -> 1/N : (sched'=1) + 1/N : (sched'=2) + 1/N : (sched'=3);

endmodule


// non-deterministic agent definitions
player p1
        player1, [exp1], [com1]
endplayer

module player1

        // exploring sites
        [exp1] sched=1 & !all_prefer_2 -> 0 : true
                        // -- evaluating site and changing preference with probability Pswitchxy
                          + 1/K * Pswitch1_1 : (preference1'=1) & (confidence1'=1) & (sched'=0)
                          + 1/K * (1-Pswitch1_1) : (sched'=0)
                                                // -- evaluating site and changing preference with probability Pswitchxy
                          + 1/K * Pswitch1_2 : (preference1'=2) & (confidence1'=1) & (sched'=0)
                          + 1/K * (1-Pswitch1_2) : (sched'=0)
                                                // -- evaluating site and changing preference with probability Pswitchxy
                          + 1/K * Pswitch1_3 : (preference1'=3) & (confidence1'=1) & (sched'=0)
                          + 1/K * (1-Pswitch1_3) : (sched'=0)
                        ;
        // communicating with other agents in the same site with probability 1-Pexp
        [com1] sched=1 & preference1!=0 & !all_prefer_2 -> 0 : true
                        // - trying to communicate with agent
                          + Pmeet_p1 * (preference1=preference2?1:0) : (confidence1'=inc_conf1) & (confidence2'=inc_conf2) & (sched'=0) // sam
                          + Pmeet_p1 * (preference1=preference2?0:1) * Pwin1_2 : (confidence1'=inc_conf1) & (preference2'=preference1) & (conf
                          + Pmeet_p1 * (preference1=preference2?0:1) * (1-Pwin1_2) : (confidence1'=1) & (preference1'=preference2) & (confiden

                                                // - trying to communicate with agent
                          + Pmeet_p1 * (preference1=preference3?1:0) : (confidence1'=inc_conf1) & (confidence3'=inc_conf3) & (sched'=0) // sam
                          + Pmeet_p1 * (preference1=preference3?0:1) * Pwin1_3 : (confidence1'=inc_conf1) & (preference3'=preference1) & (conf
                          + Pmeet_p1 * (preference1=preference3?0:1) * (1-Pwin1_3) : (confidence1'=1) & (preference1'=preference3) & (confiden

                        ;

        // don't do anything
//      [] sched=1 & !all_prefer_2 -> (sched'=0);
[] sched=1 & all_prefer_2-> true;

endmodule

player p2
        player2, [exp2], [com2]
endplayer

module player2

        // exploring sites
        [exp2] sched=2 & !all_prefer_2-> 0 : true
                        // -- evaluating site and changing preference with probability Pswitchxy
                          + 1/K * Pswitch2_1 : (preference2'=1) & (confidence2'=1) & (sched'=0)
                          + 1/K * (1-Pswitch2_1) : (sched'=0)
                                                // -- evaluating site and changing preference with probability Pswitchxy
                          + 1/K * Pswitch2_2 : (preference2'=2) & (confidence2'=1) & (sched'=0)
                          + 1/K * (1-Pswitch2_2) : (sched'=0)
                                                // -- evaluating site and changing preference with probability Pswitchxy
                          + 1/K * Pswitch2_3 : (preference2'=3) & (confidence2'=1) & (sched'=0)
                          + 1/K * (1-Pswitch2_3) : (sched'=0)
                        ;
        // communicating with other agents in the same site with probability 1-Pexp
        [com2] sched=2 & preference2!=0 & !all_prefer_2-> 0 : true
```

```
                        // - trying to communicate with agent
                          + Pmeet_p2 * (preference2=preference1?1:0) : (confidence2'=inc_conf2) & (confidence1'=inc_conf1) & (sched'=0) // sam
                          + Pmeet_p2 * (preference2=preference1?0:1) * Pwin2_1 : (confidence2'=inc_conf2) & (preference1'=preference2) & (conf
                          + Pmeet_p2 * (preference2=preference1?0:1) * (1-Pwin2_1) : (confidence2'=1) & (preference2'=preference1) & (confiden

                        // - trying to communicate with agent
                          + Pmeet_p2 * (preference2=preference3?1:0) : (confidence2'=inc_conf2) & (confidence3'=inc_conf3) & (sched'=0) // sam
                          + Pmeet_p2 * (preference2=preference3?0:1) * Pwin2_3 : (confidence2'=inc_conf2) & (preference3'=preference2) & (conf
                          + Pmeet_p2 * (preference2=preference3?0:1) * (1-Pwin2_3) : (confidence2'=1) & (preference2'=preference3) & (confiden

                        ;

        // don't do anything
//      [] sched=2 & !all_prefer_2-> (sched'=0);

        [] sched=2 & all_prefer_2-> true;

endmodule

player p3
        player3, [exp3], [com3]
endplayer

module player3

        // exploring sites
        [exp3] sched=3 & !all_prefer_2-> 0 : true
                        // -- evaluating site and changing preference with probability Pswitchxy
                          + 1/K * Pswitch3_1 : (preference3'=1) & (confidence3'=1) & (sched'=0)
                          + 1/K * (1-Pswitch3_1) : (sched'=0)
                                                // -- evaluating site and changing preference with probability Pswitchxy
                          + 1/K * Pswitch3_2 : (preference3'=2) & (confidence3'=1) & (sched'=0)
                          + 1/K * (1-Pswitch3_2) : (sched'=0)
                                                // -- evaluating site and changing preference with probability Pswitchxy
                          + 1/K * Pswitch3_3 : (preference3'=3) & (confidence3'=1) & (sched'=0)
                          + 1/K * (1-Pswitch3_3) : (sched'=0)
                        ;
        // communicating with other agents in the same site with probability 1-Pexp
        [com3] sched=3 & preference3!=0 & !all_prefer_2-> 0 : true
                        // - trying to communicate with agent
                          + Pmeet_p3 * (preference3=preference1?1:0) : (confidence3'=inc_conf3) & (confidence1'=inc_conf1) & (sched'=0) // sam
                          + Pmeet_p3 * (preference3=preference1?0:1) * Pwin3_1 : (confidence3'=inc_conf3) & (preference1'=preference3) & (conf
                          + Pmeet_p3 * (preference3=preference1?0:1) * (1-Pwin3_1) : (confidence3'=1) & (preference3'=preference1) & (confiden

                        // - trying to communicate with agent
                          + Pmeet_p3 * (preference3=preference2?1:0) : (confidence3'=inc_conf3) & (confidence2'=inc_conf2) & (sched'=0) // sam
                          + Pmeet_p3 * (preference3=preference2?0:1) * Pwin3_2 : (confidence3'=inc_conf3) & (preference2'=preference3) & (conf
                          + Pmeet_p3 * (preference3=preference2?0:1) * (1-Pwin3_2) : (confidence3'=1) & (preference3'=preference2) & (confiden

;

        // don't do anything
//      [] sched=3 & !all_prefer_2-> (sched'=0);
[] sched=3 & all_prefer_2-> true;

endmodule


// deterministic agent definitions


// formulae to increase agents' confidence levels

        formula inc_conf1 = confidence1=L ? L : (confidence1+1);
        formula inc_conf2 = confidence2=L ? L : (confidence2+1);
        formula inc_conf3 = confidence3=L ? L : (confidence3+1);

// formulae to compute probabilities of agents to meet

        // probability for agent to meet another agent independent of its location
        formula Pmeet_p1 = 1/(N-1);
        formula Pmeet_p2 = 1/(N-1);
        formula Pmeet_p3 = 1/(N-1);


// formulae to get qualities of agents' preferred sites
        formula Q_p1 =  preference1=1 ? Q1 : ( preference1=1 ? Q2 : (Q3) ) ;
        formula Q_p2 =  preference2=1 ? Q1 : ( preference2=1 ? Q2 : (Q3) ) ;
        formula Q_p3 =  preference3=1 ? Q1 : ( preference3=1 ? Q2 : (Q3) ) ;

// formulae for evaluating the sites (Pswitchij = prob of to switch from size i to site j).

        formula Pswitch1_1 = preference1=0 ? 1 : (preference1=1 ? 0 : pow(Q1, eta) / (pow(Q1, eta) + pow(Q_p1, eta)));
        formula Pswitch1_2 = preference1=0 ? 1 : (preference1=2 ? 0 : pow(Q2, eta) / (pow(Q2, eta) + pow(Q_p1, eta)));
        formula Pswitch1_3 = preference1=0 ? 1 : (preference1=3 ? 0 : pow(Q3, eta) / (pow(Q3, eta) + pow(Q_p1, eta)));

        formula Pswitch2_1 = preference2=0 ? 1 : (preference2=1 ? 0 : pow(Q1, eta) / (pow(Q1, eta) + pow(Q_p2, eta)));
        formula Pswitch2_2 = preference2=0 ? 1 : (preference2=2 ? 0 : pow(Q2, eta) / (pow(Q2, eta) + pow(Q_p2, eta)));
        formula Pswitch2_3 = preference2=0 ? 1 : (preference2=3 ? 0 : pow(Q3, eta) / (pow(Q3, eta) + pow(Q_p2, eta)));

        formula Pswitch3_1 = preference3=0 ? 1 : (preference3=1 ? 0 : pow(Q1, eta) / (pow(Q1, eta) + pow(Q_p3, eta)));
        formula Pswitch3_2 = preference3=0 ? 1 : (preference3=2 ? 0 : pow(Q2, eta) / (pow(Q2, eta) + pow(Q_p3, eta)));
        formula Pswitch3_3 = preference3=0 ? 1 : (preference3=3 ? 0 : pow(Q3, eta) / (pow(Q3, eta) + pow(Q_p3, eta)));


// formulae for conducting tournaments

        formula Pwin1_2 = (preference2=0?1:(preference1=0?0:((pow(Q_p1, lambda) * pow(confidence1, gamma)) /
                ((pow(Q_p1, lambda) * pow(confidence1, gamma))+(pow(Q_p2, lambda) * pow(confidence2, gamma))))));
        formula Pwin1_3 = (preference3=0?1:(preference1=0?0:((pow(Q_p1, lambda) * pow(confidence1, gamma)) /
                ((pow(Q_p1, lambda) * pow(confidence1, gamma))+(pow(Q_p3, lambda) * pow(confidence3, gamma))))));

        formula Pwin2_1 = 1-Pwin1_2;
        formula Pwin2_3 = (preference3=0?1:(preference2=0?0:((pow(Q_p2, lambda) * pow(confidence2, gamma)) /
                ((pow(Q_p2, lambda) * pow(confidence2, gamma))+(pow(Q_p3, lambda) * pow(confidence3, gamma))))));

        formula Pwin3_1 = 1-Pwin1_3;
        formula Pwin3_2 = 1-Pwin2_3;


// labeling states

// -- formulae to generate labels

        // agreement on site
        formula all_prefer_1 =  preference1=1 & preference2=1 & preference3=1 ;
        formula all_prefer_2 =  preference1=2 & preference2=2 & preference3=2 ;
        formula all_prefer_3 =  preference1=3 & preference2=3 & preference3=3 ;

        // compute total confidence
        formula total_confidence =  confidence1 + confidence2 + confidence3 ;

        // confidence measures
        formula all_max_conf = total_confidence/N = L;
        formula half_max_conf = (( confidence1=L?1:0 + confidence2=L?1:0 + confidence3=L?1:0 )/N) >= 0.5;

// -- labels

        // agreement on particular sites
        label "all_prefer_1" = all_prefer_1;
        label "all_prefer_2" = all_prefer_2;
        label "all_prefer_3" = all_prefer_3;

        // all agents have max confidence
        label "all_max_conf" = all_max_conf;

        label "half_max_conf" = half_max_conf;

        // agreement on a site
        label "decision_made" =  all_prefer_1 | all_prefer_2 | all_prefer_3 ;

// -- property constants
const int k;

// -- rewards

const int communication_cost = 10;
const int exploration_cost = 1;

// communication n costs
rewards "ncomm1"
        [com1] true : communication_cost;
endrewards
rewards "ncomm12"
        [com1] true : communication_cost;
        [com2] true : communication_cost;
endrewards
rewards "ncomm123"
        [com1] true : communication_cost;
        [com2] true : communication_cost;
        [com3] true : communication_cost;
endrewards

// communication d costs

// exploration n costs
rewards "nexpl1"
        [exp1] true : exploration_cost;
endrewards
rewards "nexpl12"
        [exp1] true : exploration_cost;
        [exp2] true : exploration_cost;
endrewards
rewards "nexpl123"
        [exp1] true : exploration_cost;
        [exp2] true : exploration_cost;
        [exp3] true : exploration_cost;
endrewards

// exploration d costs

// total n costs
rewards "ntot1"
        [exp1] true : exploration_cost;
        [com1] true : communication_cost;
endrewards
rewards "ntot12"
        [exp1] true : exploration_cost;
        [com1] true : communication_cost;
        [exp2] true : exploration_cost;
        [com2] true : communication_cost;
endrewards
rewards "ntot123"
        [exp1] true : exploration_cost;
        [com1] true : communication_cost;
        [exp2] true : exploration_cost;
        [com2] true : communication_cost;
        [exp3] true : exploration_cost;
        [com3] true : communication_cost;
endrewards

// total d costs

rewards "runtime"
        sched!=0 : 1;
endrewards
```

F.4 User-centric network protocol

In this section we present a sample PRISM-games SMG model with one requester and three providers of a trust-based reputation and virtual currency mechanism for user-centric networks discussed in Section 6.6.3. The model is an extended version of an MDP model of [1]. All models can be found in `http://www.prismmodelchecker.org/files/sr13trust/`.

```
smg

// Model parameters
const int K;                      // upper bound on the number of delivered servuces
const double alpha_all   = 0.8;   // recommendation influence parameter
const int st_init_all    = med;   // initial service trust for all providers
const int reduct_all     = 1;     // trust decrease for all providers
const bool hide_all      = false; // allow information hiding
const double cancel      = 0.05;  // probability to cancel fair request
const bool init_know_all = false; // sharing of initial trust
const double rev_prob    = 1.0;   // probability to share information
const double die_prob    = 0;     // probability of provider to die after serving a request
const int cmax           = 10;    // maximum service cost
const int cmin           = 2;     // minimum service cost

player requester1
        requester1, [pay11], [nopay11], [pay21], [nopay21], [pay31], [nopay31], [try11], [try21], [try31], [buy]
endplayer

player provider1
        provider1, [accept11], [refuse11], [reveal11], [notreveal11]
endplayer

player provider2
        provider2, [accept21], [refuse21], [reveal21], [notreveal21]
endplayer

player provider3
        provider3, [accept31], [refuse31], [reveal31], [notreveal31]
endplayer

module requester1

        x1 : [0..43] init 0;  // states of the module
        //ns1 : [0..K] init 0; // number of requested services
        ps1 : [0..K] init 0; // number of payed services
        nps1 : [0..K] init 0; // number of unpayed services

        unpaid1 : bool init false;

        // nondeterministic choice of the requestee
        [try11] x1=0 & (y1+y2+y3=0) & ps1+nps1 < K -> (x1'=11) & (unpaid1'=false);
        [try21] x1=0 & (y1+y2+y3=0) & ps1+nps1 < K -> (x1'=21) & (unpaid1'=false);
        [try31] x1=0 & (y1+y2+y3=0) & ps1+nps1 < K -> (x1'=31) & (unpaid1'=false);

        // buy the service off-market
        [buy] x1=0 & (y1+y2+y3=0) -> (x1'=41) & (unpaid1'=false);

        // finished sending requests
        [] x1=0 & (y1+y2+y3=0) -> (x1'=1) & (unpaid1'=false);
        [] x1=1 -> true;

        [accept11] x1=11 -> (x1'=12);
        [refuse11] x1=11 -> (x1'=13);
        [pay11] x1=12 -> (x1'=0) & (ps1'=min(K,ps1+1));
        [nopay11] x1=12 -> (x1'=0) & (nps1'=min(K,nps1+1)) & (unpaid1'=true);
        [] x1=13 -> (x1'=0);

        [accept21] x1=21 -> (x1'=22);
        [refuse21] x1=21 -> (x1'=23);
        [pay21] x1=22 -> (x1'=0) & (ps1'=min(K,ps1+1));
        [nopay21] x1=22 -> (x1'=0) & (nps1'=min(K,nps1+1)) & (unpaid1'=true);
        [] x1=23 -> (x1'=0);

        [accept31] x1=31 -> (x1'=32);
        [refuse31] x1=31 -> (x1'=33);
        [pay31] x1=32 -> (x1'=0) & (ps1'=min(K,ps1+1));
        [nopay31] x1=32 -> (x1'=0) & (nps1'=min(K,nps1+1)) & (unpaid1'=true);
        [] x1=33 -> (x1'=0);

        [] x1=41 -> (x1'=0) & (ps1'=min(K,ps1+1));

endmodule

// factor alpha of the cost formula
const double alpha1 = alpha_all;
const double alpha2 = alpha_all;
const double alpha3 = alpha_all;
// trust formula
formula trusteq1 = min(top, !know21 & !know31 ? trust11 : floor(alpha1*trust11 + (1-alpha1)*recommend));
formula recommend = ((know21 ? trust21 : 0) + (know31 ? trust31 : 0)) / ((know21 ? 1 : 0) + (know31 ? 1 : 0));

// initial knowledge parameters
const bool init_know11 = init_know_all;
const bool init_know21 = init_know_all;
const bool init_know31 = init_know_all;


// initial trust parameters
const int dt_init1 = st_init_all; // dispositional trust
const int st_init1 = st_init_all; // service trust level
const int trust_init1 = dt_init1; // initial trust
const int tth_init1 = high; // trust threshold (see the cost formula)

const int dt_init2 = st_init_all;
const int st_init2 = st_init_all;
const int trust_init2 = dt_init2;
const int tth_init2 = high;

const int dt_init3 = st_init_all;
const int st_init3 = st_init_all;
const int trust_init3 = dt_init3;
const int tth_init3 = high;
```

```
// trust reduction rates (0:NULL; 1:-1; 2:-2)
const int reduct1 = reduct_all;
const int reduct2 = reduct_all;
const int reduct3 = reduct_all;

// enable information withholding
// info about requester 1
const bool hide11 = hide_all;
const bool hide21 = hide_all;
const bool hide31 = hide_all;

module provider1

        alive1 : bool init true;

        y1 : [0..4] init 0; // states of the module
        st1 : [0..level] init st_init1; // service trust level
        dt1 : [0..level] init dt_init1; // dispositional trust
        tth1 : [0..level] init tth_init1; // trust threshold (see the cost formula)

        trust11 : [0..level] init trust_init1; // trust towards the requester
        know11: bool init init_know11; // interaction flag

        // initiate connection with requester
        [try11] alive1 & (y1=0) -> ((trusteq1 < st1) ? 1 : 0) + ((trusteq1 < st1) ? 0 : 1) * cancel : (y1'=4)
                        +  (1-(((trusteq1 < st1) ? 1 : 0) + ((trusteq1 < st1) ? 0 : 1) * cancel)) : (y1'=3);
        [try11] !alive1 & y1=0 -> (y1'=4);

        // accept or refuse requester1
        [accept11] (y1=3) & (trusteq1 >= st1) -> (y1'=1);
        //[refuse11] (y1=4) & (trusteq1 < st1) -> (y1'=0);
        [refuse11] (y1=4) -> (y1'=0);

        // settle payment with requester1
        [pay11] (y1=1) -> (y1'=2) & (trust11' = (trust11 < top ? trust11+1 : top));
        [nopay11] (reduct1=1) & (y1=1) -> (y1'=2) & (trust11' = (trust11 > null ? trust11-1 : null));
        [nopay11] (reduct1=2) &(y1=1) -> (y1'=2) & (trust11' = (trust11 > null ? trust11-2 : null));
        [nopay11] (reduct1=0) & (y1=1) -> (y1'=2) & (trust11'=null);

        // decide to reveal info about requester to other providers or not
        [reveal11] (y1=2) ->  (1-die_prob) * rev_prob : (y1'=0) & (know11'=true)
                           + (1-die_prob) * (1-rev_prob) : (y1'=0) & (know11'=false)
                           + die_prob : (y1'=0) & (alive1'=false) & (trust11'=trust_init1) & (know11'=false);
        [notreveal11] hide11 & (y1=2) -> (y1'=0) & (know11'=false);

endmodule

module provider2 = provider1 [y1=y2, st1=st2, dt1=dt2, tth1=tth2, trust11=trust21, know11=know21, alive1=alive2,
                               alpha1=alpha2, trust21=trust11, know21=know11, // renaming parameters trust formula
                               dt_init1=dt_init2, st_init1=st_init2,
                               trust_init1=trust_init2, tth_init1=tth_init2, try11=try21, init_know11=init_know21,
                               accept11=accept21, refuse11=refuse21, pay11=pay21, nopay11=nopay21,
                               reduct1=reduct2, reveal11=reveal21, notreveal11=notreveal21,
                               hide11=hide21] endmodule

module provider3 = provider1 [y1=y3, st1=st3, dt1=dt3, tth1=tth3, trust11=trust31, know11=know31, alive1=alive3,
                               alpha1=alpha3, trust31=trust11, know31=know11, // renaming parameters trust formula
                               dt_init1=dt_init3, st_init1=st_init3,
                               trust_init1=trust_init3, tth_init1=tth_init3,try11=try31, init_know11=init_know31,
                               accept11=accept31, refuse11=refuse31, pay11=pay31, nopay11=nopay31,
                               reduct1=reduct3, reveal11=reveal31, notreveal11=notreveal31,
                               hide11=hide31] endmodule

// trust level aliases
const int level = 10;
const int null = 0;
const int low = 2;
const int med = 5;
const int high = 8;
const int top = 10;

// highest price in the market
formula max_price = max((trust11 < tth1) ? cmin + ceil(((cmax - cmin) / tth1) * (tth1 - trust11)) : cmin,
                        (trust21 < tth2) ? cmin + ceil(((cmax - cmin) / tth2) * (tth2 - trust21)) : cmin,
                        (trust31 < tth3) ? cmin + ceil(((cmax - cmin) / tth3) * (tth3 - trust31)) : cmin);
//formula max_price = cmax;

// maximum difference between trust
formula max_diff = max(max(trust11-trust21,trust21-trust11), max(trust11-trust31,trust31-trust11), max(trust21-trust31,trust31-trust21));

rewards "cost"
        !unpaid1 & y1=2 : (trust11 < tth1) ? cmin + ceil(((cmax - cmin) / tth1) * (tth1 - trust11)) : cmin;
        !unpaid1 & y2=2 : (trust21 < tth2) ? cmin + ceil(((cmax - cmin) / tth2) * (tth2 - trust21)) : cmin;
        !unpaid1 & y3=2 : (trust31 < tth3) ? cmin + ceil(((cmax - cmin) / tth3) * (tth3 - trust31)) : cmin;
        x1=41 : max_price;
endrewards

rewards "cost2"
        !unpaid1 & y1=2 :  max_diff + ((trust11 < tth1) ? cmin + ceil(((cmax - cmin) / tth1) * (tth1 - trust11)) : cmin);
        !unpaid1 & y2=2 :  max_diff + ((trust21 < tth2) ? cmin + ceil(((cmax - cmin) / tth2) * (tth2 - trust21)) : cmin);
        !unpaid1 & y3=2 :  max_diff + ((trust31 < tth3) ? cmin + ceil(((cmax - cmin) / tth3) * (tth3 - trust31)) : cmin);
        x1=41 : max_diff + max_price;
endrewards

rewards "payed"
        !unpaid1 & y1=2 : 1;
        !unpaid1 & y2=2 : 1;
        !unpaid1 & y3=2 : 1;
endrewards
rewards "nopayed"
        unpaid1 & y1=2 : 1;
        unpaid1 & y2=2 : 1;
        unpaid1 & y3=2 : 1;
endrewards
rewards "accepted"
        [accept11] true : 1;
        [accept21] true : 1;
        [accept31] true : 1;
endrewards
```

Bibliography

[1] A. Aldini and A. Bogliolo. Model checking of trust-based user-centric cooperative networks. In *Proc. International Conference on Advances in Future Internet (AFIN'12)*, pages 32–41, 2012.

[2] A. Aldini and A. Bogliolo. Trading performance and cooperation incentives in user-centric networks. In *Proc. International Workshop on Quantitative Aspects in Security Assurance (QASA'12)*, 2012.

[3] R. Alur and T.A. Henzinger. Reactive modules. *Formal Methods in System Design*, 15(1):7–48, 1999.

[4] R. Alur, T.A. Henzinger, and O. Kupferman. Alternating-time temporal logic. *Journal of the ACM*, 49(5):672–713, 2002.

[5] R. Alur, T.A. Henzinger, F. Mang, S. Qadeer, S. Rajamani, and S. Tasiran. MOCHA: Modularity in model checking. In *Proc. International Conference on Computer Aided Verification (CAV'98)*, volume 1427 of *Lecture Notes in Computer Science*, pages 521–525. Springer, 1998.

[6] S. Andova, H. Hermanns, and J.-P. Katoen. Discrete-time rewards model-checked. In *Proc. Formal Modeling and Analysis of Timed Systems (FORMATS'03)*, volume 2791 of *Lecture Notes in Computer Science*, pages 88–104. Springer, 2003.

[7] A. Aziz, V. Singhal, F. Balarin, R.K. Brayton, and A.L. Sangiovanni-Vincentelli. It usually works: The temporal logic of stochastic systems. In *Proc. International Conference on Computer Aided Verification (CAV'95)*, volume 939 of *Lecture Notes in Computer Science*, pages 155–165. Springer, 1995.

[8] C. Baier, T. Brázdil, M. Größer, and A. Kucera. Stochastic game logic. *Acta Informatica*, 49(4):203–224, 2012.

[9] C. Baier, F. Ciesinski, and M. Größer. PROBMELA: A modeling language for communicating probabilistic processes. In *Proc. International Conference on Formal Methods and Models for Co-Design (MEMOCODE'04)*, pages 57–66. IEEE, 2004.

[10] C. Baier and J.-P. Katoen. *Principles of Model Checking.* MIT Press, 2008.

[11] P. Ballarini, M. Fisher, and M.J. Wooldridge. Automated game analysis via probabilistic model checking: A case study. *Electronic Notes in Theoretical Computer Science*, 149(2):125 – 137, 2006.

[12] T. Basar and G.J. Olsder. *Dynamic noncooperative game theory.* SIAM, 1995.

[13] G. Behrmann, A. David, K.G. Larsen, J. Håkansson, P. Pettersson, W. Yi, and M. Hendriks. Uppaal 4.0. In *Proc. International Conference on the Quantitative Evaluation of Systems (QEST'06)*, pages 125–126. IEEE, 2006.

[14] A. Bianco and L. de Alfaro. Model checking of probabilistic and nondeterministic systems. In *Proc. Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'95)*, volume 1026 of *Lecture Notes in Computer Science*, pages 499–513. Springer, 1995.

[15] A. Bogliolo, P. Polidori, A. Aldini, W. Moreira, P. Mendes, M. Yildiz, C. Ballester, and J. Seigneur. Virtual currency and reputation-based cooperation incentives in user-centric networks. In *Proc. International Wireless Communications and Mobile Computing Conference (IWCMC'12)*, pages 895–900. IEEE, 2012.

[16] A. Bohy, V. Bruyère, E. Filiot, and J.-F. Raskin. Synthesis from LTL specifications with mean-payoff objectives. In *Proc. International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'13)*, volume 7795 of *Lecture Notes in Computer Science*, pages 169–184. Springer, 2013.

[17] R.H. Bordini, M. Fisher, C. Pardavila, and M.J. Wooldridge. Model checking AgentSpeak. In *Proc. International Conference on Autonomous Agents and Multiagent Systems (AAMAS'03)*, pages 409–416. ACM, 2003.

[18] R.H. Bordini, M. Fisher, W. Visser, and M.J. Wooldridge. Verifying multi-agent programs by model checking. *Autonomous Agents and Multi-Agent Systems*, 12(2):239–256, 2006.

[19] T. Brázdil, V. Brožek, V. Forejt, and A. Kučera. Stochastic games with branching-time winning objectives. In *Proc. Symposium on Logic in Computer Science (LICS'06)*, pages 349–358. IEEE, 2006.

[20] T. Brázdil, V. Brozek, K. Chatterjee, V. Forejt, and A. Kucera. Two views on multiple mean-payoff objectives in Markov decision processes. In *Proc. Symposium on Logic in Computer Science (LICS'11)*, pages 33–42. IEEE, 2011.

[21] T Brázdil, K. Chatterjee, V. Forejt, and A. Kucera. Trading performance for stability in Markov decision processes. In *Proc. Symposium on Logic in Computer Science (LICS'13)*, pages 331–340. IEEE, 2013.

[22] N. Bulling and W. Jamroga. What agents can probably enforce. *Fundamenta Informaticae*, 93(1–3):81–96, 2009.

[23] A. Chakrabarti, L. de Alfaro, T.A. Henzinger, and M. Stoelinga. Resource interfaces. In *Proc. International Conference on Embedded Software (EMSOFT'03)*, volume 2855 of *Lecture Notes in Computer Science*, pages 117–133. Springer, 2003.

[24] K. Chatterjee. Concurrent games with tail objectives. *Theoretical Computer Science*, 388(13):181–198, 2007.

[25] K. Chatterjee, L. de Alfaro, and T.A. Henzinger. The complexity of stochastic Rabin and Streett games. In *Proc. International Colloquium on Automata, Languages and Programming (ICALP'05)*, volume 3580 of *Lecture Notes in Computer Science*, pages 878–890. Springer, 2005.

[26] K. Chatterjee and L. Doyen. Energy and mean-payoff parity Markov decision processes. In *Proc. International Symposium on Mathematical Foundations of Computer Science (MFCS'11)*, volume 6907 of *Lecture Notes in Computer Science*, pages 206–218. Springer, 2011.

[27] K. Chatterjee, L. Doyen, T.A. Henzinger, and J.-F. Raskin. Generalized mean-payoff and energy games. In *Proc. Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'10)*, volume 8 of *Leibniz International Proceedings in Informatics*, pages 505–516, 2010.

[28] K. Chatterjee and T.A. Henzinger. Strategy improvement for stochastic Rabin and Streett games. In *Proc. International Conference on Concurrency Theory (CONCUR'06)*, volume 4137 of *Lecture Notes in Computer Science*, pages 375–389. Springer, 2006.

[29] K. Chatterjee and T.A. Henzinger. Value iteration. In *25 Years of Model Checking*, volume 5000 of *Lecture Notes in Computer Science*, pages 107–138. Springer, 2008.

[30] K. Chatterjee, T.A. Henzinger, B. Jobstmann, and A. Radhakrishna. Gist: A solver for probabilistic games. In *Proc. International Conference on Computer Aided Verification (CAV'10)*, volume 6174 of *Lecture Notes in Computer Science*, pages 665–669. Springer, 2010.

[31] K. Chatterjee, T.A. Henzinger, and M. Jurdzinski. Mean-payoff parity games. In *Proc. Symposium on Logic in Computer Science (LICS'05)*, pages 178–187. IEEE, 2005.

[32] K. Chatterjee, T.A. Henzinger, and M. Jurdziński. Games with secure equilibria. *Theoretical Computer Science*, 365(1-2):67–82, 2006.

[33] K. Chatterjee, M. Jurdzinski, and T.A. Henzinger. Quantitative stochastic parity games. In *Proc. Symposium on Discrete Algorithms (SODA'04)*, pages 121–130. SIAM, 2004.

[34] K. Chatterjee, R. Majumdar, and T.A. Henzinger. Stochastic limit-average games are in EXPTIME. *International Journal of Game Theory*, 37(2):219–234, 2008.

[35] K. Chatterjee, M. Randour, and J.-F. Raskin. Strategy synthesis for multi-dimensional quantitative objectives. In *Proc. International Conference on Concurrency Theory (CONCUR'12)*, volume 7454 of *Lecture Notes in Computer Science*, pages 115–131. Springer, 2012.

[36] T. Chen, V. Forejt, M.Z. Kwiatkowska, D. Parker, and A. Simaitis. Automatic verification of competitive stochastic systems. In *Proc. International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'12)*, volume 7214 of *Lecture Notes in Computer Science*, pages 315–330. Springer, 2012.

[37] T. Chen, V. Forejt, M.Z. Kwiatkowska, D. Parker, and A. Simaitis. Automatic verification of competitive stochastic systems. *Formal Methods in System Design*, 43(1):61–92, 2013.

[38] T. Chen, V. Forejt, M.Z. Kwiatkowska, D. Parker, and A. Simaitis. PRISM-games: A model checker for stochastic multi-player games. In *Proc. International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'13)*, volume 7795 of *Lecture Notes in Computer Science*, pages 187–193. Springer, 2013.

[39] T. Chen, V. Forejt, M.Z. Kwiatkowska, A. Simaitis, A. Trivedi, and M. Ummels. Playing stochastic games precisely. In *Proc. International Conference on Concurrency Theory (CONCUR'12)*, volume 7454 of *Lecture Notes in Computer Science*, pages 348–363. Springer, 2012.

[40] T. Chen, V. Forejt, M.Z. Kwiatkowska, A. Simaitis, and C. Wiltsche. On stochastic games with multiple objectives. In *Proc. International Symposium on Mathematical Foundations of Computer Science (MFCS'13)*, volume 8087 of *Lecture Notes in Computer Science*, pages 266–277. Springer, 2013.

[41] T. Chen, M.Z. Kwiatkowska, D. Parker, and A. Simaitis. Verifying team formation protocols with probabilistic model checking. In *Proc. International Workshop on Computational Logic in Multi-Agent Systems (CLIMA'11)*, volume 6814 of *Lecture Notes in Computer Science*, pages 190–207. Springer, 2011.

[42] T. Chen, M.Z. Kwiatkowska, A. Simaitis, and C. Wiltsche. Synthesis for multi-objective stochastic games: An application to autonomous urban driving. In *Proc. International Conference on Quantitative Evaluation of Systems (QEST'13)*, volume 8054 of *Lecture Notes in Computer Science*, pages 322–337. Springer, 2013.

[43] T. Chen and J. Lu. Probabilistic alternating-time temporal logic and model checking algorithm. In *Proc. International Conference on Fuzzy Systems and Knowledge Discovery (FSKD'07)*, pages 35–39. IEEE, 2007.

[44] C.-H. Cheng, A. Knoll, M. Luttenberger, and C. Buckl. GAVS+: An open platform for the research of algorithmic game solving. In *Proc. International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'11)*, volume 6605 of *Lecture Notes in Computer Science*, pages 258–261. Springer, 2011.

[45] F. Ciesinski and C. Baier. LiQuor: A tool for qualitative and quantitative linear time analysis of reactive systems. In *Proc. International Conference on Quantitative Evaluation of Systems (QEST'06)*, pages 131–132. IEEE, 2006.

[46] E.M. Clarke and E.A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Proc. Workshop on Logics of Programs*, volume 131 of *Lecture Notes in Computer Science*, pages 52–71. Springer, 1982.

[47] E.M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1986.

[48] A. Condon. The complexity of stochastic games. *Information and Computation*, 96(2):203–224, 1992.

[49] C. Courcoubetis and M. Yannakakis. Verifying temporal properties of finite state probabilistic programs. In *Proc. Symposium on Foundations of Computer Science (FOCS'88)*, pages 338–345. IEEE, 1988.

[50] C. Courcoubetis and M. Yannakakis. Markov decision processes and regular events. In *Proc. International Colloquium on Automata, Languages and Programming (ICALP'90)*, volume 443 of *Lecture Notes in Computer Science*, pages 336–349. Springer, 1990.

[51] C. Courcoubetis and M. Yannakakis. The complexity of probabilistic verification. *Journal of the ACM*, 42(4):857–907, 1995.

[52] L. de Alfaro. *Formal Verification of Probabilistic Systems.* PhD thesis, Stanford University, 1997.

[53] L. de Alfaro. Computing minimum and maximum reachability times in probabilistic systems. In *Proc. International Conference on Concurrency Theory (CONCUR'99)*, volume 1664 of *Lecture Notes in Computer Science*, pages 66–81. Springer, 1999.

[54] K. Deb. *Multi-objective optimization using evolutionary algorithms.* John Wiley & Sons Hoboken, NJ, 2001.

[55] M. Duflot, M.Z. Kwiatkowska, G. Norman, and D. Parker. A formal analysis of Bluetooth device discovery. *Journal on Software Tools for Technology Transfer*, 8(6):621–632, 2006.

[56] S. Dziembowski, M. Jurdzinski, and I. Walukiewicz. How much memory is needed to win infinite games? In *Proc. Symposium on Logic in Computer Science (LICS'97)*, pages 99–110. IEEE, 1997.

[57] K. Etessami, M.Z. Kwiatkowska, M.Y. Vardi, and M. Yannakakis. Multi-objective model checking of Markov decision processes. *Logical Methods in Computer Science*, 4(4), 2008.

[58] K. Etessami, D. Wojtczak, and M. Yannakakis. Recursive stochastic games with positive rewards. In *Proc. International Colloquium on Automata, Languages and Programming (ICALP'08), Part I*, volume 5125 of *Lecture Notes in Computer Science*, pages 711–723. Springer, 2008.

[59] J. Filar and K. Vrieze. *Competitive Markov Decision Processes.* Springer, 1997.

[60] V. Forejt, M.Z. Kwiatkowska, G. Norman, and D. Parker. Automated verification techniques for probabilistic systems. In *Proc. International School on Formal Methods for the Design of Computer, Communication and Software Systems (SFM'11)*, volume 6659 of *Lecture Notes in Computer Science*, pages 53–113. Springer, 2011.

[61] V. Forejt, M.Z. Kwiatkowska, and D. Parker. Pareto curves for probabilistic model checking. In *Proc. International Symposium on Automated Technology for Verification and Analysis (ATVA'12)*, volume 7561 of *Lecture Notes in Computer Science*, pages 317–332. Springer, 2012.

[62] M.E. Gaston and M. desJardins. Agent-organized networks for dynamic team formation. In *Proc. International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'05)*, pages 230–237. ACM, 2005.

[63] H. Gimbert, Y. Oualhadj, and S. Paul. Computing optimal strategies for Markov decision processes with parity and positive-average conditions. Technical report, LaBRI, Universite de Bordeaux II, 2011.

[64] M. Grötschel, L. Lovász, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization.* Springer, 2nd edition, 1993.

[65] L. Grunske. Specification patterns for probabilistic quality properties. In *Proc. International Conference on Software Engineering (ICSE'08)*, pages 31–40. ACM, 2008.

[66] H. Hansson and B. Jonsson. A logic for reasoning about time and reliability. *Formal Aspects of Computing*, 6(5):512–535, 1994.

[67] D. Harel. Effective transformations on infinite trees, with applications to high undecidability, dominoes, and fairness. *Journal of the ACM*, 33(1):224–248, 1986.

[68] H. Hildmann and F. Saffre. Influence of variable supply and load flexibility on demand-side management. In *Proc. International Conference on the European Energy Market (EEM'11)*, pages 63–68, 2011.

[69] J. Hillston. *PEPA: Performance enhanced process algebra.* PhD thesis, University of Edinburgh, Department of Computer Science, 1993.

[70] G.J. Holzmann. Design and validation of protocols: A tutorial. *Computer Networks and ISDN Systems*, 25(9):981–1017, 1993.

[71] G.J. Holzmann. The model checker SPIN. *IEEE Transactions on Software Engineering*, 23(5):279–295, 1997.

[72] J.-P. Katoen, E. M. Hahn, H. Hermanns, D. Jansen, and I. Zapreev. The ins and outs of the probabilistic model checker MRMC. *Performance Evaluation*, 68(2):90–104, 2011.

[73] M. Kattenbelt, M.Z. Kwiatkowska, G. Norman, and D. Parker. A game-based abstraction-refinement framework for Markov decision processes. *Formal Methods in System Design*, 36(3):246–280, 2010.

[74] J.G. Kemeny, J.L. Snell, and A.W. Knapp. *Denumerable Markov chains.* Van Nostrand Princeton, 1966.

[75] M.Z. Kwiatkowska, G. Norman, and D. Parker. Stochastic model checking. In *International School on Formal Methods for the Design of Computer, Communication and Software Systems: Performance Evaluation (SFM'07)*, volume 4486 of *Lecture Notes in Computer Science*, pages 220–270. Springer, 2007.

[76] M.Z. Kwiatkowska, G. Norman, and D. Parker. Stochastic games for verification of probabilistic timed automata. In *Proc. International Conference on Formal Modelling and Analysis of Timed Systems (FORMATS'09)*, volume 5813 of *Lecture Notes in Computer Science*, pages 212–227. Springer, 2009.

[77] M.Z. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *Proc. International Conference on Computer Aided Verification (CAV'11)*, volume 6806 of *Lecture Notes in Computer Science*, pages 585–591. Springer, 2011.

[78] M.Z. Kwiatkowska, G. Norman, and D. Parker. Probabilistic verification of Hermans self-stabilisation algorithm. *Formal Aspects of Computing*, 24(4):661–670, 2012.

[79] M.Z. Kwiatkowska, G. Norman, D. Parker, and H. Qu. Assume-guarantee verification for probabilistic systems. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'10)*, volume 6105 of *Lecture Notes in Computer Science*, pages 23–37. Springer, 2010.

[80] M.Z. Kwiatkowska, G. Norman, D. Parker, and J. Sproston. Performance analysis of probabilistic timed automata using digital clocks. *Formal Methods in System Design*, 29:33–78, 2006.

[81] M.Z. Kwiatkowska, D. Parker, and A. Simaitis. Strategic analysis of trust models for user-centric networks. In *Proc. International Workshop on Strategic Reasoning (SR'13)*, volume 112 of *Electronic Proceedings in Theoretical Computer Science*, pages 53–60, 2013.

[82] M. Lahijanian, J. Wasniewski, S.B. Andersson, and C. Belta. Motion planning and control from temporal logic specifications with probabilistic satisfaction guarantees.

In *Proc. International Conference on Robotics and Automation (ICRA'10)*, pages 3227–3232. IEEE, 2010.

[83] F. Laroussinie, N. Markey, and G. Oreiby. On the expressiveness and complexity of ATL. *Logical Methods in Computer Science*, 4(2), 2008.

[84] D. Lehmann and S. Shelah. Reasoning with time and chance. *Information and Control*, 53(3):165–198, 1982.

[85] A. Lomuscio, H. Qu, and F. Raimondi. MCMAS: A model checker for the verification of multi-agent systems. In *Proc. International Conference on Computer Aided Verification (CAV'09)*, volume 5643 of *Lecture Notes in Computer Science*, pages 682–688. Springer, 2009.

[86] R.T. Marler and J.S. Arora. Survey of multi-objective optimization methods for engineering. *Structural and Multidisciplinary Optimization*, 26(6):369–395, 2004.

[87] D. Martin. The determinacy of Blackwell games. *Journal of Symbolic Logic*, 63(4):1565–1581, 1998.

[88] A. McIver and C. Morgan. Results on the quantitative mu-calculus qMu. *ACM Transactions on Computational Logic*, 8(1), 2007.

[89] G.L. Nemhauser and L.A. Wolsey. *Integer and combinatorial optimization.* Wiley New York, 1988.

[90] G. Norman and V. Shmatikov. Analysis of probabilistic contract signing. *Journal of Computer Security*, 14(6):561–589, 2006.

[91] M.J. Osborne and A. Rubinstein. *Bargaining and markets.* San Diego: Academic press, 1990.

[92] Christos H Papadimitriou. *Computational complexity.* John Wiley and Sons Ltd., 2003.

[93] D. Parker. *Implementation of Symbolic Model Checking for Probabilistic Systems.* PhD thesis, University of Birmingham, 2002.

[94] D.C. Parkes and S.P. Singh. An MDP-based approach to online mechanism design. In *Advances in Neural Information Processing Systems (NIPS'03).* MIT Press, 2003.

[95] A. Pnueli. On the extremely fair treatment of probabilistic algorithms. In *Proc. Symposium on Theory of Computing (STOC'83)*, pages 278–290. ACM, 1983.

[96] M.L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming.* Wiley, 1994.

[97] J.-P. Queille and J. Sifakis. Specification and verification of concurrent systems in CESAR. In *Proc. International Symposium on Programming*, volume 137 of *Lecture Notes in Computer Science*, pages 337–351. Springer, 1982.

[98] T.E.S. Raghavan and J.A. Filar. Algorithms for stochastic games - A survey. *Zeitschrift für Operations Research*, 35(6):437–472, 1991.

[99] F. Saffre and A. Simaitis. Host selection through collective decision. *ACM Transactions on Autonomous and Adaptive Systems*, 7(1):4, 2012.

[100] S. Salamah, A.Q. Gates, V. Kreinovich, and S. Roach. Verification of automatically generated pattern-based LTL specifications. In *Proc. International Symposium on High Assurance Systems Engineering (HASE'07)*, pages 341–348. IEEE, 2007.

[101] A. Sesic, S. Dautovic, and V. Malbasa. Dynamic power management of a system with a two-priority request queue using probabilistic-model checking. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27(2), 2008.

[102] L. S. Shapley. Stochastic games. *Proceedings of the National Academy of Sciences of the United States of America*, 39(10):1095, 1953.

[103] A.P. Sistla and E.M. Clarke. The complexity of propositional linear temporal logics. *Journal of the ACM*, 32(3):733–749, 1985.

[104] B. Suman and P. Kumar. A survey of simulated annealing as a tool for single and multiobjective optimization. *Journal of the Operational Research Society*, 57(10):1143–1160, 2005.

[105] C.-K. Tham and R. Buyya. Sensorgrid: Integrating sensor networks and grid computing. *CSI Communications*, 29(1):24–29, 2005.

[106] Wolfgang Thomas. *Languages, automata, and logic.* Springer, 1997.

[107] M. Ummels and D. Wojtczak. The complexity of Nash equilibria in simple stochastic multiplayer games. In *Proc. Internatilonal Collogquium on Automata, Languages and Programming (ICALP'09), Part II*, volume 5556 of *Lecture Notes in Computer Science*, pages 297–308. Springer, 2009.

[108] M.Y. Vardi. Automatic verification of probabilistic concurrent finite-state programs. In *Proc. Symposium on Foundations of Computer Science (FOCS'85)*, pages 327–338. IEEE, 1985.

[109] Y. Velner, K. Chatterjee, L. Doyen, T.A. Henzinger, A. Rabinovich, and J.-F. Raskin. The complexity of multi-mean-payoff and multi-energy games. *CoRR*, abs/1209.3234, 2012.