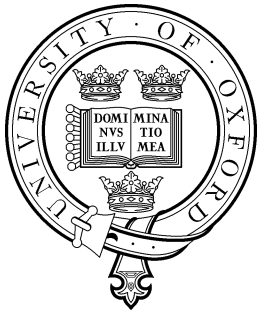
Automatic Verification and Strategy Synthesis for Zero-sum and Equilibria Properties of Concurrent Stochastic Games



Gabriel H. R. Santos

Trinity College

University of Oxford

A thesis submitted for the degree of

Doctor of Philosophy

Ao meu pai, ao meu primo e à minha avó,
que nos deixaram tão cedo.

Acknowledgements

The completion of this work would not have been possible were it not for the guidance and encouragement provided by my supervisor Marta Kwiatkowska and close collaborators Gethin Norman and Dave Parker, to whom I am deeply indebted. Some of the outcomes reported in this thesis are aligned with ongoing research projects led by Marta Kwiatkowska, the EPSRC Programme Grant on Mobile Autonomy (EP/M019918/1) and FUN2MODEL (grant agreement No. 834115), funded by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme. I would also like to acknowledge the Engineering and Physical Sciences Research Council, the Department of Computer Science and Trinity College for supporting my research. Special thanks go to family and friends in Brazil, France and the UK, who were there for me all throughout.

Oxford, 2020.

Abstract

Concepts originating from *game theory* have been employed to formulate and analyse problems from a variety of domains, with applications ranging from economics to biology. In computer science, where they also found a fertile soil, game-theoretic ideas and solutions have been used in AI, network protocols and verification, to name but a few. This thesis sets itself in that context by presenting a framework for verifying *concurrent stochastic* systems, where agents may *compete* or *cooperate*. More specifically, we model systems as *concurrent multi-player stochastic games*, which are mathematical models able to express environmental *uncertainty*, *nondeterminism* and, at each step, the outcomes of the *choices* made by a set of agents. Given that assuming strict competition among agents does not reflect the full range of system behaviours, we also make use of the notion of *social welfare Nash equilibria* in order to perform *quantitative* or *qualitative analysis*, as well as *strategy synthesis*, while considering a distinct set of *objectives*. Equilibria properties and strategies have the advantage of being *stable*, i.e., they are the result of or a description of *rational* behaviour, meaning it is actually in the best interest of the agents to act in a way that leads to such an outcome. Verification is also facilitated by the possibility of specifying *coalitions*, which can translate *communication* and *synchronisation* among agents, along with the presentation of a property specification logic for probabilistic and reward-based temporal objectives. Our model checking algorithms for a variant of stopping games make use of linear and nonlinear optimisation along with SMT solvers and are able to verify *zero-sum* and *nonzero-sum* properties of concurrent stochastic models. For nonzero-sum properties, our methods focus on computing *optimal equilibria*, which maximise or minimise the *probabilities* or *rewards* associated to a given set of objectives and may include a mixture of *finite* and *infinite-horizon* operators. For finite-horizon properties the computation is exact, while for infinite-horizon it is approximate using *value iteration*.

We implement our methods in the PRISM-games 3.0 tool, demonstrating their usefulness and performance on several case studies.

Contents

List of Figures

CHAPTER 1

Introduction

Computerised devices play a part in virtually all activities in modern society. While their presence as a tool to automate tasks or to facilitate human interaction is already established, their role as *artificial agents*, whether interacting with the physical world as self-driving cars, or confined to a virtual environment as trading agents, is only beginning to flourish. In all scenarios, it is paramount that interactions involving these agents, both with humans as well as other artificial agents, are safe and appropriate. In order for that to happen, we have to ensure they abide by a set of requirements, and need to be able to assess their reliability and establish a degree of trust in how they behave. *Formal methods* is a branch of computer science whose very *raison d'être* is addressing issues regarding the correctness, safety and performance of software and hardware systems. Correctness of systems is typically established by employing *formal verification*, which aims to prove that the system model satisfies a given requirement specification. More specifically, *probabilistic verification* and *quantitative analysis* allow for the specification and reasoning about systems whose behaviour is *probabilistic* (also referred to as *stochastic*). Naturally, the mathematical models and methods behind the description and analysis of such systems have to be sophisticated and powerful enough to handle ever growing complexity. In particular, when considering interacting agents, one needs to be able to faithfully represent not only their individual behaviours but also what form that interaction takes, that is, whether they *collaborate* or *compete*, and what their *intentions* or *objectives* are. An attractive solution to tackle these issues has been to adopt concepts from *game theory*, a field of study developed initially to deal with problems in economics and the

military.

Generally speaking, a *game* is a formal description of a scenario in which agents (or *players*) interact through *actions* (or *moves*). The possible combinations of actions chosen by each agent at a given stage can affect their environment in different ways, and thus have an influence on what they are able to do in future, as well as the final *outcome* (*value*) of the game. In addition, games enable the introduction of concepts typically not present in probabilistic models, for example *turns*, *concurrency*, *adversarial behaviour* and *coalitions*.

Other than the specification of actions and outcomes, one of the core concepts behind game-theoretic models is that of *strategies*. A strategy is a description of how each agent should act in different stages of a game, that is, what action to take in each possible scenario when considering a given objective. Therefore, one of the benefits of borrowing from game-theoretic concepts is the possibility of not only reasoning about what may happen during a system's execution, but also synthesising a strategy (or a *controller*) for each agent to achieve a particular outcome or to prevent an *undesirable event* from happening. Techniques such as *model checking* [60, 176] (aka *automated verification*) have commonly been used both for analysing a model and constructing strategies automatically, and to this end a range of model checking tools have been developed and applied on relevant case studies. Of these the most relevant to this thesis is PRISM-games [139], which supports the modelling of multi-player turn-based stochastic games, where players alternate their moves, and reasoning about their coalitional behaviour using rPATL (Probabilistic Alternating-time Temporal Logic with Rewards) [53] as a property specification logic. Unfortunately, existing approaches have been historically focused on *worst-case* scenarios, in which one assumes agents will necessarily play against each other, capturing *zero-sum* objectives only. It is not difficult to see, however, that in many cases this is an overly strong assumption as, in reality, it might be in their best interest as rational agents to collaborate.

Aims. Despite considerable progress in verification and strategy synthesis for stochastic games, previous algorithmic solutions and probabilistic verification tools have focused mainly on *strictly competitive* models and properties. Another feature missing from the software tool landscape was the possibility of modelling concurrent behaviour in games, in the sense of allowing players to make moves concurrently, which constrained the set of applications. The main goal of the work presented in this thesis is to expand the scope of verification methods for stochastic games by developing model checking and strategy synthesis algorithms that incorporate the notions of both *concurrency* and *equilibria* in the context of multi-player games. To achieve this, we

build on PRISM-games [139], which supports turn-based stochastic games with two-coalition properties, in order to develop a comprehensive framework that comprises a formal description of the underlying models and methods, an extension of rPATL as a property specification logic for *zero-sum* or *equilibria* objectives over concurrent stochastic games (including *coalitional two-player* and *multi-player* cases), a substantially improved modelling specification language, a comprehensive set of case studies to illustrate the applicability and performance of the algorithms, as well as a new release of the PRISM-games 3.0 software tool developed as part of this thesis.

Approach. This thesis focuses on the development and integration of explicit and symbolic methods within automated verification of stochastic games. This involves formulating a modelling language, construction of models from the description, and developing and implementing algorithms for model checking. Stochastic games [185] appear in different forms in the literature, depending on the domain. In computer science, turn-based stochastic games played on graphs can be linked to probabilistic automata (PAs) [183] and *Markov Decision Processes* (MDPs) [17], with the difference that nondeterministic choices are controlled by separate agents. Concurrent stochastic games generalise turn-based games by allowing the transitions to be controlled by multiple agents simultaneously. The system's behaviour and how it progresses is then determined by how these agents interact and there is no prior assumption of whether their objectives are opposed or complementary. The different types of goals (zero-sum and equilibria) are then expressed by means of a *property* specification, which relates to system's variables, its agents and how they are grouped into *coalitions*.

One of the main bottlenecks of model checking is *scalability*. Real systems often possess a large number of variables and possible behaviours which, in order to be faithfully translated, may require a considerable number of states and transitions. This poses the challenge of finding efficient methods and data structures in order to build, store and verify these models. Symbolic data structures such as *decision diagrams* have been successfully integrated within probabilistic verification when considering non-competitive models [172], with substantial scalability improvements. Similarly, *SAT* (satisfiability) and *SMT* (satisfiability modulo theory) solvers have been used in model checking tools to address scalability [20, 21, 22]. Recently, the performance of SAT/SMT solvers, in particular, has progressively improved in terms of the number of clauses and variables they are able to handle, which holds promise for their application in this setting.

While there are a number of algorithms and tools for equilibria computation, to the best of our knowledge they are limited to relatively small games, mainly in *normal*

form. Specifically, there is a lack of general and accessible verification frameworks and tools for the computation of values for large games played over *graphs*, especially those considering equilibria for (stochastic) games in *extensive form*. There is also little work on applying satisfiability theory in this context, and even algorithms tool support for reasoning about zero-sum and concurrent stochastic games are limited. This thesis aims to fill some of these gaps by presenting novel theoretical and practical contributions that address some of these shortcomings.

Lastly, in order to showcase the applications and evaluate the effectiveness of our methods, we present a comprehensive set of case studies that illustrate the scope and potential impact of the work developed, as well as a stable and publicly available tool PRISM-games 3.0 that implements them.

Contributions

We develop a framework for model checking and strategy synthesis for both zero-sum and equilibria properties of turn-based and concurrent stochastic games in non-discounted setting, focusing on providing novel algorithms for equilibria-based verification and the integration of symbolic SMT-based methods. For the variants we consider (*zero-sum*, *nonzero-sum*, *two-coalitional*, *multi-coalitional*), we propose iterative techniques for exact and approximate value computation when verifying *finite-horizon* and *infinite-horizon* properties, respectively. This builds on PRISM-games 2.0 framework, which is restricted to temporal, two-coalition, zero-sum properties for turn-based stochastic games. Through several case studies we demonstrate the application of the techniques to modelling of multi-agent systems, highlighting the advantages of considering objectives that are not strictly opposed. The main contributions of this work are presented in Chapters 4-8 and can be summarised as follows:

1. In Chapter 4, we give our model checking and strategy synthesis algorithms for temporal and reward zero-sum properties. We extend rPATL to allow for the specification of strictly competitive objectives for players in concurrent stochastic games and combine *value iteration* and *linear programming* methods to compute/approximate values for models and properties that may include action and state costs or rewards. We enhance the framework with total expected cost and instantaneous reward operators and describe how our algorithms can be used for strategy synthesis.

2. In Chapter 5, we present our model checking and strategy synthesis algorithms for temporal and reward two-player equilibria properties. We reason about equilibria that guarantee optimal choices at every state of the game according to different criteria and make use of an SMT-based encoding and implementation in order to compute these values at a state level. The computation/approximation of values for properties is achieved by combining our approach with previously developed techniques for MDP model checking. In order to guarantee uniqueness, we further extend our property specification logic to consider a combination of distinct objectives that is expressed as the sum of the values associated to them. We further discuss how to perform strategy synthesis, combining finite and infinite-horizon objectives, and the challenges behind equilibria computation.

3. In Chapter 6, we introduce our model checking and strategy synthesis algorithms for temporal and reward multi-player equilibria properties. Computing values for multi-player properties is considerably more involved than for the two-player case and requires a different approach. We develop a solution that makes use of *nonlinear programming* in order to compute equilibria strategies and values at different stages of the game, and further extend our property specification logic to reason about the combination of an arbitrary number of distinct coalitions and properties. We also discuss the challenges of computing values and constructing strategies for multi-player properties.

4. In Chapter 7, we discuss the implementation of the methods detailed in Chapters 4-6 as part of PRISM-games 3.0. This new release substantially extends the functionality of its predecessor with a new modelling language that supports the specification of concurrent stochastic games and the novel strategy synthesis and model checking algorithms for zero-sum and equilibria properties. We give an overview of its main features and present the relevant technical aspects.

5. In Chapter 8, we present a range of case studies covering verification and strategy synthesis for different properties of concurrent stochastic games, as well as a discussion of the tool's efficiency and scalability. Our selection of examples comprises applications from finance, network protocols, attack-defence scenarios, and others. We highlight the particularities of zero-sum and equilibria properties, as well as two- and multi-player coalition variants.

We also present additional details, including a proof of correctness and pseudo-code for the algorithms, in Appendices A and B.

Thesis Outline

In Chapter 2, we present an overview of work related to this thesis, while in Chapter 3, we review the necessary technical concepts and known results. The main theoretical contributions are presented in Chapters 4-6. Chapter 7 is dedicated to a description of the tool we developed to implement our framework. We describe some of the case studies considered throughout in Chapter 8. We conclude with Chapter 9 by presenting a summary of our contributions and pointing out open problems and possible extensions.

Publications

Part of the work presented in this thesis has been published in jointly authored papers. Extending the algorithms and code base of PRISM-games was a considerable undertaking and has benefited from collaboration with Gethin Norman (University of Glasgow) and David Parker (University of Birmingham), the original developers of the framework. The methods described in Chapter 4, along with the relevant case studies of Chapter 8, were presented in the 15th edition of QEST (International Conference on Quantitative Evaluation of Systems) that took place in Bejijing, China, in 2018 [136]. I developed the initial version of the model checking algorithms for probabilistic reachability and total rewards, which were later extended and formalised in collaboration with Gethin Norman, with whom I also collaborated on the elaboration of the case studies. I developed and implemented the model building, model checking and strategy synthesis procedures with the assistance from Dave Parker.

Similarly, the results for equilibria properties presented in Chapter 5 and the relevant case studies made up our successful submission to the 23rd edition of FM (International Symposium on Formal Methods), the main conference of the 3rd World Congress on Formal Methods that took place in Porto, Portugal in 2019 [137]. The work presented at FM led to an invitation to submit a journal paper to a special issue of FMSD (Formal Methods in System Design) [140]. I developed the SMT-based approach for equilibria computation, extended the algorithms presented in [136] to compute equilibria values for bounded model checking formulae, and incorporated

the implementation into PRISM-games with support from Dave Parker. I collaborated with Gethin Norman on the development of model checking algorithms for the unbounded operators, the correctness proof and the case studies.

Additionally, the work detailed in Chapter 7 led to a successful submission of a tool paper to the 32nd edition of CAV (International Conference on Computer-Aided Verification), which was hosted online in Los Angeles, USA in July 2020 [139]. I developed the new modelling language as well as some of the reported case studies in collaboration with Gethin Norman and Dave Parker.

Finally, our method for verifying multi-player equilibria properties has been accepted for publication in the proceedings of the 17th edition of QEST, which took place online in Vienna, Austria, in October 2020 [138]. I developed and implemented the approach for computing values for multi-player games combining SMT and nonlinear programming. I collaborated on the development and formalisation of the model checking algorithms as well as the implementation of the case studies with Gethin Norman.

The beta version release of PRISM-games 3.0, developed and made public with support from David Parker and Gethin Norman, can be found online at `https://www.prismmodelchecker.org/games/`. A website containing a description of the tool, tutorials and webpages for the collection of case studies was set up mainly by Gethin Norman, with support from David Parker and myself, and can be found by following the same link.

CHAPTER 2

Related Work

Contents

Game-theoretic concepts are actively applied to computer science problems as a way to model and synthesise controllers for stochastic reactive systems [62, 63]. The rigorous treatment of game theory itself as an independent field has its origins in the seminal work by John von Neumann [204], although it can be linked all the way back to 1838 with Cournot's work on duopolies and other early work by Zermelo [218] and Borel [28]. Other fundamental work was carried out by Shapley [185], who considered finite concurrent games with *discounted* rewards. A few survey papers have covered the connection between the two areas, such as [124], [187] and more recently [192]. Games remain an active topic of research within formal methods, often appearing in conferences dedicated to formal verification, concurrency, artificial intelligence, networking, electronic commerce, learning and other topics. As a modelling formalism, games are especially relevant to the design and study of systems in which multiple entities with different information sets and objectives interact. Extensive theoretical work has been carried out on the use of games in computer science. There have been numerous papers and theses written on different aspects of games such as the complexity of playing extensive games (games on graphs) for a variety of *objectives*

[43, 201], on computing the result of normal form games (*strategic form*) [171] and mechanism design [165], with dedicated books covering the many topics [166]. Considerable amount of experimental work has also taken place to develop tools and case studies that showcase the usefulness of this formalism. Among the many examples of applications one can cite: autonomous urban driving [55], smart energy management [53], decision making in sensor networks [53], reputation-based services [142], financial markets [157] and control systems [16], including dedicated books for the analysis of wireless protocols [105, 106].

This chapter will cover different developments related to the rigorous treatment of games, equilibria and their application to formal verification. We start in Section 2.1 by setting the problem and presenting results concerning the complexity of computing zero-sum and equilibria values for normal form and extensive games. In Section 2.2 we explore different approaches for combining verification and equilibria. We continue in Section 2.2.1 by discussing the complementary problem of synthesis, another contribution of this thesis. Next, Section 2.2.2 presents a brief overview of the applications of symbolic and hybrid methods to probabilistic verification. We conclude this chapter by summarising other applications and tools dedicated to model checking for games in Section 2.3.

2.1 Games and Equilibria Computation

Nash equilibria (NE) [164] is one of the best known concepts to emerge from game theory. The general idea behind an equilibrium is having a collection of strategies from which players do not have an incentive to deviate. Computing values for equilibria and zero-sum objectives is a difficult task. One can divide that task into two categories: computing values of *normal* (or *strategic*) *form* games and computing values of *extensive form* games. A normal form game is one described by a number of players, an *action* set for each player and a *payoff* or *utility* function that maps all elements of the product of actions sets to a real value (we present a formal description and a detailed discussion in Section 4.1). On the other hand, extensive games are usually described by graphs, with nodes representing a given stage of the game and edges representing the choices made by the players. They are also commonly represented by trees, where nodes can be grouped according to *information sets*, with the game progressing as players make choices at different nodes and the leaves accounting for the payoff each one of them receives. A directed graph can be similarly used as a representation for extensive games and will be employed to describe games in

the contribution chapters of this thesis, with vertices (or *states*) representing distinct information sets in which players choose an action from the set of actions available to them. Differently from game trees, graphs may have self-loops in their structure and may not have terminal nodes. Directed graphs are the common representation for *iterated* or *repeated* games, i.e., games that may consist of an indefinite number of stages. The payoffs for games played over graphs may then be specified by auxiliary structures that associate real values to states or transitions. Uncertainty can be added to these different representations by the means of an extra player, often referred to as *nature*, as well as through probabilistic nodes or probability distributions associated to the transitions.

Although computation for games in extensive form (games played over graphs) can be reduced to that of a normal form game, the nature of the objectives (*reachability*, *expected rewards*, *parity*, *Büchi*, *co-Büchi*, *Rabin-chain*, *Streett*, *Muller...*), in addition to the fact they could be played for an infinite number of rounds, sets them apart when considering their complexity and the algorithmic solutions that are used to calculate or approximate their values. Throughout this document we will refer to different variants of games and different variants of objectives. We present a formal definition of the relevant concepts in Chapter 3. At this stage, it suffices to say that an objective is an outcome that is of particular interest for a player in the game and thus defines how the player should act. A game can then be characterised by a type of objective given we have an assumption on the players' behaviours. We start this section with an overview of results concerning zero-sum and parity games. In the following sections, we focus on methods for computing equilibria.

Zero-sum Objectives. Zero-sum objectives translate scenarios of strict competition among players in a game. As will be further detailed in Chapter 3, zero-sum objectives can also be formally characterised as an equilibrium but are computationally easier. Computing zero-sum values for normal form games reduces to *linear programming*, which can be solved in PTIME [86]. With respect to extensive concurrent games, various results concerning this type of objectives are discussed in [76]. Differently from turn-based games, concurrent games generally require the use of *randomised strategies* and, even for reachability properties, optimal strategies may not exist. Also, for ω-regular objectives (a subclass of Borel objectives widely applied to reactive systems [151, 194]), infinite memory may be required for both optimal and ε-optimal strategies. It is also known that the *value* of a game can be irrational and thus in general we can only hope to compute ε-approximations [76]. Furthermore, for Büchi, co-Büchi and Rabin-chain, the alternation of greatest and least fixpoint operators within known

algorithms leads to approximation procedures that contain nested iterations, and it is unclear how to obtain monotonically converging approximation schemes. Etessami and Yannakakis in [82] comment on the fact that, for finite-state concurrent games, while the maximiser can only hope for ε-optimal strategies, the minimiser does have optimal randomised memoryless strategies as shown in [86, 76]. It is also shown that deciding *qualitative* and *quantitative* termination problems for concurrent stochastic games is in PSPACE. The authors also point out a flaw in [52] and establish that approximating the values of finite-state concurrent reachability games is in PSPACE. *Sure* and *almost-sure* reachability for Büchi and co-Büchi can be decided in PTIME [75, 74].

Parity Objectives. Parity conditions encompass all simpler ω-regular objectives, including Büchi, co-Büchi, safety and reachability [162]. They are therefore very expressive and widely studied. A parity objective essentially establishes a condition on the set of nodes that is visited infinitely often. They are of particular interest due to the fact that any ω-regular objective can be turned into a parity objective following a modification to the game graph [162]. Computing values for this class of objectives is in PSPACE, while decidability for *sure* and *almost-sure* variants is in NP $\cap$ co-NP [48].

Equilibria in Normal Form Games. One of the most heavily researched topics within game theory is the computation of equilibria values of normal form games. Within this topic, one has to differentiate between the cases of *two* and *multi*-player (or n-player) games. The problem is considerably simpler for the former as it falls within the class of *linear complementarity problems* (LCP) [64] [80], whereas for the multi-player case, nonlinearities are present due to products of strategy variables of the other players. Papadimitriou established in one of his seminal papers the problem of finding a mixed Nash equilibrium for two-player games to be in PPAD (*polynomial parity argument in a directed graph*) [171] and, to the best of our knowledge, there is still no polynomial algorithm for the general case. Daskalakis, Goldberg and Papadimitriou in [71] and Chen, Deng and Teng in [57] subsequently proved that finding NE for games with two or more players is PPAD-complete. One of the first and better known methods for finding equilibria points and values of two-player games is the Lemke-Howson algorithm [145], although there are equilibria which cannot be located by this method [186]. We refer the reader to [206, 109, 217] for an extensive survey on the subject. For the multi-player case, the Govindan-Wilson algorithm [99, 97, 210] is a well-known solution and we refer the reader to [214] for a comprehensive discussion

on the matter. Other solutions for equilibria computation include *quantal response equilibria* [199, 160], function minimisation [158] and Mangasarian's method [150]. Etessami and Yannakakis in [83] define the complexity class FIXP (for *fix point problems*), whose upper bound would be PSPACE, and show the problem of computing equilibria values for games with 3 or more players to be FIXP-complete. In addition to dedicated algorithmic solutions, numerous search and enumeration-based procedures have been developed for both two and multi-player cases [174, 179, 9]. Porter, Nudelman and Shoham in [174] presented experimental results to support that their combination of iterative strategy elimination algorithms combined with linear and nonlinear optimisation third-party software in most cases outperformed implementations of dedicated algorithms. More closely related to our problem, it has been shown in [95] that determining whether there exists an equilibrium that guarantees a player a payoff of a given bound is NP-complete for a two-player game and NP-hard for the general case. Moreover, Avis, Rosenberg, Savani and von Stegel in [9] demonstrate that a two-player game may have a number of equilibria that is exponential with respect to the number of actions, and thus any method that relies on enumerating them in the worst-case cannot be expected to perform in a running time that is polynomial with respect to the input size. Other recent works concerning equilibria and normal form games include [67] presenting methods to compute approximate Nash equilibria in bimatrix games with different simultaneous criteria, [68] where Czumaj, Fasoulakis and Jurdziński present a polynomial-time algorithm for computing an approximate Nash equilibrium in symmetric n-player games, [69] where the same authors use zero-sum game techniques for designing a polynomial-time algorithm to compute additive approximate Nash equilibria in *bimatrix* games and [66] where a distributed method to compute approximate equilibria for two-player normal form games is developed.

Equilibria in Extensive Games. Probably the earliest method for computing equilibria for games played over trees is *backward induction* [181, 205, 188], which works by propagating equilibria values up the tree starting from the leaves. Despite it being a method that goes back to the very origins of game theory, it is still the basis and continues to be applied in modern algorithms and applications. Kuhn showed in 1950 [133] that every finite turn-based game in extensive form has an equilibrium in pure strategies, which can be found by performing backward induction. A method based on reducing extensive games to normal form games and solving them through linear complementarity is developed in [131]. Turocy [200] extends the method of quantal response equilibria to extensive games. Littman, Ravi, Talwar

and Zinkevich [146] propose a polynomial algorithm for computing optimal equilibria for turn-based games played over trees, while considering different criteria such as *social optimum (utilitarian)*, *fairness (egalitarian)*, *single maximum (republican)* and *player best (libertarian)* values. Blum, Shelton and Koller [25] develop an exact algorithm for computing Nash equilibria for games played over *multi-agent influence diagrams* (MAIDs). Closer to the work reported in this thesis, Chatterjee, Majumdar and Jurdziński in [42, 52] write about a variety of aspects for computing equilibria values, establishing the complexity of computing Nash equilibria for reachability objectives over graphs to be EXPTIME and devising algorithms for computing values for upward-closed objectives. Nash equilibria exist for n-player concurrent games with *safety* objectives for all players [182]. Existence of ε-Nash equilibria for all $\varepsilon > 0$ has also been demonstrated for n-player concurrent games with reachability objectives [52] and for two-player concurrent games with ω-regular objectives [41]. More recently, Brihaye, Bruyère, Goeminne, Raskin and van den Bogaard proved in [35] that the problem of computing *subgame-perfect* equilibria [184] for quantitative reachability objectives is PSPACE-complete. Other recent work include [34], which focus on complexity results for ω-regular Boolean objectives and weak subgame-perfect equilibria over turn-based games.

2.2 Verification and Strategy Synthesis for Games

Previous works have also tackled the issue of combining equilibria computation and verification in computer science. Efforts have been made both in terms of developing property specification logics that reason about equilibria in games and algorithms that compute or approximate values for these properties. ATL (Alternating-time Temporal Logic) [4] was introduced for games in order to offer selective quantification over the *paths* representing the possible outcomes, such as those where the system and the game alternated their moves. It makes it possible then to specify the player whose actions are controllable and write properties that translate safety and reachability objectives, which may also include an infinite-horizon until operator. pATL (Probabilisitic Alternating-time Temporal Logic) enables reasoning about concurrent stochastic multi-player games in a similar way that PCTL (Probabilistic Computation Tree Logic) facilitates the specification of properties for MDPs, but model checking of the specifications reduces to solving concurrent stochastic two-player games [56, 39] and rewards are not supported. SGL (Stochastic Game Logic) 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 [10], as is also the case for a variant of PCTL for stochastic games [31]. One of the first logics to be able to express equilibria objectives is *strategy logic* (SI) [50], which differentiates between *paths* and *strategy* formulae. In SI, path formulae are LTL formulae interpreted over infinite paths, while strategy formulae apply first-order logic quantifiers and logical connectives to the players' strategies, thus being able to express *existence* of both NE and *secure* NE. More recently, Aminof, Kwiatkowska, Maubert, Murano and Rubin [5] introduced *probabilistic strategy logic* (PSL), an extension of SI for stochastic systems. PSL allows one to express several central concepts related to game theory such as ε-NE, *Pareto optimality* and *dominating strategies*. Attempts have also been made to devise algorithms for specific variants of games and other probabilistic models. Fernando, Dong, Jegourel and Dong [85] considered the problem of computing approximate equilibria for probabilistic BAR (distributed systems that includes *Byzantine, Altruistic* and *Rational*) agents. Similarly, Mari, Melatti, Salvo, Tronci, Alvisi, Clement and Li in [152] develop a symbolic model checking algorithm for the verification of Nash equilibria for finite-state mechanisms modelling *Multiple Administrative Domains* (MAD), which are distributed systems with possible coalitions and Byzantine agents.

2.2.1 Strategy Synthesis for Games

Complementary to the problem of computing optimal expected utility values for players in a game, *strategy synthesis* has also been an active research topic within game theory from the very beginning. When considering the original game-theoretic approach and games in strategic form, the problem of computing a value for a player in a game is actually described and implemented as the task of finding strategy profiles that satisfy an equilibrium condition. Strategy synthesis has also received considerable attention within model checking due to the possibility of constructing controllers automatically from a specification. In the verification context, strategies are then not only a witness that a given property is satisfied by a model but also a recipe of how to implement a program that will guarantee that happens. When we consider the wider context of stochastic systems, this means being able to derive a procedure that would guarantee, for instance, a required level of performance with respect to a given metric. The task of synthesising a strategy in a two-player zero-sum game is finding a sequence of moves that would lead to a *winning condition* for one of the players against all possible strategies the other player could choose to adopt, while for equilibria the idea is finding sets of strategies that would be a *mutual best response*.

A winning condition in verification is based on a specification which can be of *qualitative* or *quantitative* nature. A qualitative specification in non-stochastic games can be thought of as a mapping from system runs to Boolean values. Such problems were first stated by Church in [59] and solved by Büchi and Landweber [37], which established the connection between specification logics and game theory. Other more recent application examples include [170] and [213]. For stochastic variants, it means that, when considering all possible runs of a system, which are then the result of the actions made by the players, a property is satisfied with probability 1. Synthesis for qualitative LTL (Linear Temporal Logic) and CTL* (Computational Tree Logic) for stochastic systems was studied by Schewe in [180] and for *parity* objectives by Chatterjee, Jurdziński and Henzinger in [51]. Strategy synthesis for MDPs has also been used as a mechanism design tool for multi-agent systems to synthesise the equilibria strategies [173]. On the other hand, quantitative specifications aim to establish not only whether an objective is achieved or not, but how well it is achieved [24]. In a strictly competitive setting, we traditionally aim to maximise or minimise the expected payoff (also referred to as rewards or costs) for a given player, which is accumulated as the game is played or determined when the game ends. These are known as quantitative due to the fact that the strategies are required to optimise a numerical value according to an objective, which can be *long-run* [96], *mean-payoff* [86], *ratio* [203], ω-regular [76], or *Pareto optimal* with respect to some other objective (when considering multi-objective queries [87]) among other variants.

Strategy complexity is concerned with whether it is necessary to randomise among actions in a state and also if no memory, finite memory or infinite memory elements are required in order to construct a strategy. Strategies can also employ a *stochastic update* mechanism for updating memory elements in order to have reduced representations, as used for turn-based games in [189]. Chatterjee and Henzinger in [48] point to the fact that memoryless randomised strategies are sufficient for ε-optimal zero-sum reachability, safety and co-Büchi objectives. On the other hand, strategies for parity objectives are in the family of randomised, history-dependent and infinite memory strategies, apart from *sure parity* objectives, which only require *pure memoryless* strategies. For the work presented in this thesis, all computed strategies need randomisation and some need finite memory. Furthermore, strategy synthesis is performed using model checking algorithms for the different types of properties, and thus have the same computational complexity.

2.2.2 Symbolic and Hybrid Methods in Synthesis and Verification

Symbolic algorithms and data structures have been applied to and, in some contexts, revolutionised model checking. They helped to expand the scope of applications by providing succinct representation of models and strategies as well as algorithms that deal with entire regions of the state space instead of individual states, which meant considerable gains in performance and scalability. The use of *Binary Decision Diagrams* (BDDs) [130] and *Multi-terminal Decision Diagrams* (MTBDDs) [209] proved to be pivotal for modelling and verifying large-scale models in both deterministic and probabilistic contexts [161, 91, 172]. The implementation of BDD and MTBDD-based model checking procedures are the very backbone of well known model checkers such as SMV and PRISM, and since then other variants have been developed and used to represent and verify both discrete and continuous systems [216, 36]. A prototype of a symbolic MTBDD-based engine for zero-sum verification of turn-based stochastic games was also developed within the context of the work presented in this thesis and used to achieve better scalability on a case study for safe navigation [121]. Similarly, SAT solvers have also been used in verification and some authors have analysed the trade-offs between the different encodings [20, 21]. We refer the reader to [22] for a comprehensive survey on the matter. More recently, exciting work has been done combining SAT solvers and *model counting*. Fremont, Rabe and Seshia in [88] combine sampling techniques with a SAT solver and maximal model counting to compute reachability probability values over runs in a *discrete-time Markov chain*, a method we believe could also be applied for computing optimal equilibria for games in strategic form.

2.3 Model Checking Tools for Games and Applications

Numerous other applications of (stochastic) games have been developed within and outside Computer Science such as in engineering [153], operations research [61], and medical fields [197]. Games have also been applied to derive strategies for patrol planning [215], port defence [6] and infrastructure protection [92]. There is also a clear link between the fields of strategy synthesis and control theory. Although the latter is typically applied to systems with continuous dynamics, those can be approximated through discretisation, which allows for the automatic synthesis of controllers that

satisfy temporal logic specifications in both non-stochastic [18, 132] and stochastic settings [193, 212]. Naturally, the process of discretisation introduces errors but efforts have been made in the direction to bound them and provide formal performance guarantees [190, 195].

A variety of tools have been designed to provide support for probabilistic verification of non-competitive systems. General purpose probabilistic model checkers such as PRISM [135], MRMC [126] (which relies on PRISM or PEPA [113] specifications), STORM [112], iscasMC [103], PAT [191] and the Modest Toolset [110] can be used to verify discrete and continuous-time probabilistic models, including nondeterministic and timed variants such as MDPs and PTAs. PRISM, STORM and MultiGain [122] can also be used for multi-objective probabilistic verification of MDPs. LiQuor [11] is another tool that supports MDP verification for models written as Probmela programs [12] against LTL properties.[1] MCMAS [147], a model checker for non-stochastic multi-agent systems, makes use of an extension of the MOCHA modelling language [2] and supports the verification of ATL specifications, in addition to different epistemic operators. MCMAS and MOCHA both utilise symbolic representations based on BDDs to achieve better scalability. The well-known model checker SPIN [114] supports LTL verification for systems specified in Promela [115] and counts among its many case studies the analysis of multi-agent systems [27] developed by translating AgentSpeak [26] into Promela. Among other relevant and emerging specification formats that can be used for probabilistic models we mention *probabilistic guarded-command language* (pGCL)[156], (conditional probabilistic guarded-command language) cpGCL[100], *dynamic fault trees* (DFT)[81, 178], *generalized stochastic Petri nets* (GSPN)[154] and JANI [38]. Other specialist tools for non-competitive models (with some support for verification of probabilistic models) include PARAM [102], COMICS [119], PROPhESY [78], PLASMA [120], COSMOS [15] and CADP [94].

A limited number of tools have been developed to provide support for zero-sum verification and equilibria computation for normal form and extensive games. Previous versions of PRISM-games [144] already supported verification of *turn-based* stochastic games for zero-sum and *multi-objective* temporal probabilistic and reward-based properties that included *reachability*, *expected reachability*, *long-run average* and *ratio reward*. GIST [49] allows the analysis of ω-regular properties for probabilistic games, but again focuses on turn-based, not concurrent, games. GIST has among its features the ability to perform synthesis of assumptions for the environment so

[1] LiQuor did not appear to be maintained or accessible at the time of writing of this document.

that a given objective can be achieved in case no controller at first can be constructed. GAVS+ [58] is a general-purpose tool for algorithmic game solving, supporting turn-based and (non-stochastic) concurrent games, but not concurrent stochastic games.[2] For non-stochastic games, model checking tools such as PRALINE [33], EAGLE [198] and EVE [118] support Nash equilibria, as does MCMAS-SLK [40] via *strategy logic*. MOCHA [2] supports model checking and synthesis for ATL (Alternating-time Temporal Logic) specification, also for non-stochastic games. General purpose tools such as Gambit [159] can compute a variety of equilibria for normal form and extensive games but, again, not for stochastic games. Gambit can be used both as a standalone tool and a library, and supports games with more than two players by implementing several methods for equilibria computation including support enumeration, simplicial subdivision, function minimisation, and linear complementarity among others. GameTracer is another general purpose tool that can be used either through a command line interface or a C library, whose implementation was also incorporated into Gambit. It can solve games in normal form with two players or more and implements the global Newton method of Govindan and Wilson [97], as well as their iterative polymatrix algorithm [98], which performs better but does not provide any theoretical guarantees of convergence. Nashpy [129] was developed as a Python library and also supports computation for two-player games in normal form implementing Lemke-Howson [145] as well as support and vertex enumeration.

2.4 Summary

In this chapter, we presented an overview of previous works concerning the topics of equilibria computation, model checking and strategy synthesis for stochastic games and the use of different symbolic methods within verification. We have also covered different applications and related tools. Through the referenced works we have identified a gap when it comes to verification algorithms and a framework for concurrent stochastic games that encompasses modelling, model checking and strategy synthesis. Particularly concerning equilibria computation, algorithmic solutions for extensive concurrent games have been few and far between. Moreover, although methods for computing values for zero-sum objectives had been previously developed, there was a lack of tool support and experimental results for verification. This thesis aims to develop a comprehensive framework to address this gap.

[2]Neither GAVS+ nor GIST appear to be maintained or accessible via the links provided in the respective publications at the time of writing of this document.

CHAPTER 3

Background

Contents

In this chapter, we review concepts, definitions and known results that we use in the remainder of this thesis. We start by introducing some preliminary notation and reviewing game-theoretic concepts such as *matrix* and *bimatrix games*, *strategies*, and *Nash equilibria*. Next, we review the probabilistic models DTMCs (*discrete-time Markov chains* and MDPs (*Markov decision processes*). In addition, we present various notions of strategies, objectives and specifications that form the foundation of our strategy synthesis and model checking algorithms. We then go through model checking methods for MDPs, such as reductions to *linear programming* as well as *value iteration*. Finally, we present an overview of the property specification logic PCTL and develop examples to illustrate the connection between its semantics and verification methods.

3.1 Notation

In this section, we introduce background notions and notation that will be used throughout this document.

Tuples and Sequences. Given sets $A_1, A_2, \cdots, A_n$, we define the *Cartesian product* $A_1 \times A_2 \times \cdots \times A_n$ as the set of *tuples* $\{(a_1, a_2, \cdots, a_n) \mid \forall i,\ a_i \in A_i\}$. Given a tuple $a = (a_1, a_2, \cdots, a_n)$, we denote by a_i the ith component, a_{-i} the tuple without the ith component, i.e., $a_{-i} = (a_1, \cdots, a_{i-1}, a_{i+1}, \cdots, a_n)$ and $a_{-i}[a_i'] = (a_1, \cdots, a_{i-1}, a_i', a_{i+1}, \cdots, a_n)$ the tuple resulting from replacing the ith element by a_i'. Given a finite sequence $\rho = a_0 a_1 \cdots a_n$, we denote its length by $|\rho| \stackrel{def}{=} n + 1$ and define $first(\rho) \stackrel{def}{=} a_0$ and $last(\rho) \stackrel{def}{=} a_n$. A *prefix* of a sequence ρ is a finite sequence κ such that $\rho = \kappa\rho'$ for some sequence ρ'.

Probability. A *discrete probability distribution* (also just referred to as *distribution*) over a countable set E is a function $\mu : E \to [0, 1]$ such that $\sum_{e \in E} \mu(e) = 1$. We denote by $Dist(E)$ the set of *discrete probability distributions* over E. A *support* of a distribution $\mu \in Dist(E)$ is then defined as $supp(\mu) \stackrel{def}{=} \{e \in E \mid \mu(e) > 0\}$. A distribution $\mu \in Dist(E)$ is *Dirac* if $\mu(e) = 1$ for some $e \in E$. We define the *product between two distributions* $\mu^1 \in Dist(E^1)$ and $\mu^2 \in Dist(E^2)$ as $\mu^1 \times \mu^2$, where $\mu^1 \times \mu^2(e^1, e^2) = \mu^1(e^1) \cdot \mu^2(e^2)$ for all $e^1 \in E^1$ and $e^2 \in E^2$. We now define concepts related to probability measure as presented in [134]. We refer the reader to [23] for a comprehensive treatment of the topic.

Definition 3.1.1 (σ-algebra)**.** Let Ω be an arbitrary non-empty set and $\mathcal{F}$ a family of subsets of Ω. We say that $\mathcal{F}$ is a field on Ω if:

1. the empty set $\varnothing$ is in $\mathcal{F}$;
2. whenever A is an element of $\mathcal{F}$, then the complement $\Omega \backslash A$ is in $\mathcal{F}$;
3. whenever A and B are elements of $\mathcal{F}$, then $A \cup B$ is in $\mathcal{F}$.

A field of subsets $\mathcal{F}$ is called a σ-algebra if it is a field closed under countable union: whenever $A_i \in \mathcal{F}$ for $i \in \mathbb{N}$, the $\cup_{i \in \mathbb{N}} A_i$ is also in $\mathcal{F}$. The elements of a σ-algebra are called *measurable sets*, and $(\Omega, \mathcal{F})$ is called a *measurable space*. A σ-algebra generated by a family of sets A_i, denoted $\sigma(A)$, is the smallest σ-algebra that contains A.

Definition 3.1.2 (Measurable space)**.** Let $(\Omega, \mathcal{F})$ be a measurable space. A function $\mathcal{P} : \mathcal{F} \to [0, 1]$ is a probability measure on $(\Omega, \mathcal{F})$ and $(\Omega, \mathcal{F}, \mathcal{P})$ a probability space, if $\mathcal{P}$ satisfies the following properties:

1. $\mathcal{P}(\Omega) = 1$;

2. $\mathcal{P}(\cup_i A_i) = \sum_i \mathcal{P}(A_i)$ for any countable sequence of mutually disjoint sets $A_1, A_2, \ldots$ of $\mathcal{F}$;

The set Ω is called the *sample space* and the elements of $\mathcal{F}$ *events*.

Definition 3.1.3 (Random variable)**.** Let $(\Omega, \mathcal{F}, \mathcal{P})$ be a probability space. A function $X : \Omega \to \mathbb{R}$ is said to be a *random variable*. Given a random variable $X : \Omega \to \mathbb{R}$, and the probability space $(\Omega, \mathcal{F}, \mathcal{P})$, the expectation or average value with respect to the measure $\mathcal{P}$ is given by: $E[X] = \int_{\omega \in \Omega} X(\omega) d\mathcal{P}$.

Relations and Functions. A *relation* R between sets A and B is a subset $R \subseteq A \times B$, and is a *function* from A to B, written $f : A \to B$, if no $a \in A$ has more than one $b \in B$ such that $R(a, b)$. A function is *total* if $f(a)$ is defined for all a, and it is *partial* if there can be unassigned $a \in A$.

3.2 Game-theoretic Concepts

In this section, we review key concepts and results from game theory.

Normal form games. Normal form games (NFGs) are simple game representations. They define a set of *players*, a set of *actions* for each player and a *utility function* that describes, for each player, a *payoff* based on the collective choice. NFGs are *one-shot* games, that is, concurrently, each player makes a choice and the game ends. Table 3.1 shows the representation of a two-player prisoner's dilemma in normal form. In the table, columns a, u_1 and u_2 represent the collective choice, player 1's utility and player 2's utility, respectively. In prisoner's dilemma we have the notions of *cooperation* and *defection*, which are represented by c_i and d_i in the action tuples, $i \in \{1, 2\}$. For instance, if player 1 chooses to cooperate and player 2 chooses to defect, resulting in the action tuple (c_1, d_2), the former gets a payoff of -4 while the latter gets 0.

a	u_1	u_2	a	u_1	u_2
(c_1, c_2)	-1	-1	(d_1, c_2)	0	-4
(c_1, d_2)	-4	0	(d_1, d_2)	-3	-3

Table 3.1: Two-player prisoner's dilemma in normal form.

Extensive form games. *Extensive form* are representations that do not necessarily assume players in a game make choices simultaneously. Thus, they can be used

to describe games possibly composed of multiple rounds and in which the outcome of a given round may have an influence in the next. Extensive form games are often represented by trees or graphs. Normal form games can be represented by extensive games and a reverse transformation is also possible. In Figure 3.1, we show the representation of the two-player prisoner's dilemma example in extensive form. We identify player i nodes as p_i. The dotted line between the p_2 nodes indicates they belong to the same information set, meaning that, given we are representing a concurrent game, player 2 makes its choice without knowing the move previously taken by player 1.

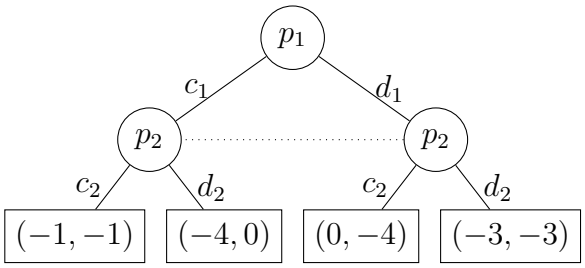


Figure 3.1: Two-player prisoner's dilemma in extensive form.

Strategies and strategy profile. A *strategy* is a way to resolve nondeterminism in both normal form and extensive games. A *mixed* strategy is one described by a probability distribution over a set of actions. A *pure* strategy is one where the probability distribution is Dirac. A *strategy profile* is defined as a tuple that contains a strategy for each player in the game. For instance, considering the two-player prisoner's dilemma, we could define a strategy profile $\sigma = (\sigma_1, \sigma_2)$, with a pure strategy $\sigma_1 = [1 \mapsto c_1]$ for player 1, and a mixed strategy $\sigma_2 = [\frac{1}{2} \mapsto c_2,\ \frac{1}{2} \mapsto d_2]$ for player 2.

Payoff or utility. A payoff or *utility* is a real value that represents the outcome of a game for a given player. In both normal form and extensive games, the collection of payoffs can also be called a *payoff vector*. Given players may adopt different types of strategies, it is possible to compute the vector of expected payoffs as a weighted sum by taking into account the probability of each player playing each action. E.g., for the two-player prisoner's dilemma, we have payoffs defined for players 1 and 2 under the columns indexed by u_i of Table 3.1 or as payoff vectors in the leaf nodes of Figure 3.1.

Dominating and weakly dominating strategies. A *dominating* strategy is a pure strategy that guarantees a player a payoff that is *strictly* higher than that of any other pure strategy, regardless of what strategies other players may play. A *weakly*

dominating strategy on the other hand is a pure strategy that guarantees a payoff *at least as high* as any other pure strategy. Let us consider again the two-player prisoner's dilemma example, then strategy $\sigma_1 = [1 \mapsto d_1]$ is a dominating strategy for player 1. Similarly, picking d_2 with probability 1 is a dominating strategy for player 2.

Zero-sum and nonzero-sum games. A game is said to be *zero-sum* if all players have strictly opposing goals, that is, the payoff of a player is equal to the negation of the sum of the other players in the game. A game is *nonzero-sum* if the sum of the players' payoffs is not necessarily zero. Table 3.2 shows a classic example of a zero-sum game called *matching pennies*. Columns a, u_1 and u_2 represent the collective choice, player 1's utility and player 2's utility, respectively. In this example, each player has a coin for which they may choose the value to be heads or tails. If the coins match, player 1 wins the round, which is indicated by being awarded a payoff of 1, while player 2 gets a payoff of -1. If the coins do not match, their payoffs are inverted. It is easy to see that the sum of payoffs is always zero, no matter what strategy the players may choose to adopt. We refer to the NFG description of the two prisoner's dilemma in Table 3.1 as an example of a nonzero-sum game.

a	u_1	u_2	a	u_1	u_2
(h_1, h_2)	1	-1	(t_1, h_2)	-1	1
(h_1, t_2)	-1	1	(t_1, t_2)	1	-1

Table 3.2: Matching pennies game in normal form.

Matrix and bimatrix games. A *matrix game* is a representation of a two-player zero-sum normal form game in matrix form. It uses the fact that, in a two-player zero-sum game, the payoffs of a player are the negated values of the other. Similarly, a *bimatrix game* is used to describe a two-player nonzero-sum game by defining two separate payoff matrices. Figure 3.2 shows the matching pennies and prisoner's dilemma represented as matrix and bimatrix games, respectively. We use the notation Z to refer to a matrix representation of a game across this document. In Figure 3.2, matrix Z_i represents then the payoffs of player i given the different possible plays indicated by the actions associated to the rows and columns.

Nash equilibrium. A *Nash equilibrium* (NE) is a strategy profile from which it is not advantageous to any player to deviate. Both in normal and extensive form games, there can be multiple equilibria. Also, there could be different types of equilibria. An equilibrium can be maximal with respect to the payoff of a given player, can be

$$\mathsf{Z} = \begin{array}{c} \\ h_1 \\ t_1 \end{array} \begin{array}{c} \begin{array}{cc} h_2 & t_2 \end{array} \\ \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix} \end{array} \qquad \mathsf{Z}_1 = \begin{array}{c} \\ c_1 \\ d_1 \end{array} \begin{array}{c} \begin{array}{cc} c_2 & d_2 \end{array} \\ \begin{pmatrix} -1 & -4 \\ 0 & -3 \end{pmatrix} \end{array} \quad \mathsf{Z}_2 = \begin{array}{c} \\ c_1 \\ d_1 \end{array} \begin{array}{c} \begin{array}{cc} c_2 & d_2 \end{array} \\ \begin{pmatrix} -1 & 0 \\ -4 & -3 \end{pmatrix} \end{array}$$

(a) Matching pennies. (b) Two-player prisoner's dilemma.

Figure 3.2: Matrix representations.

minimal with respect to the difference between the payoffs of the players, can be maximal in terms of the sum of payoffs, etc. The strategy profile where both players *defect* in a prisoner's dilemma is a Nash equilibrium. It is easy to see that, if one of the players chooses to defect, the better option for the other is to also defect.

Social welfare and social cost Nash equilibria. A *social welfare Nash equilibrium* (SWNE) is an NE that maximises the sum of the players' payoffs. A *social cost Nash equilibrium* (SCNE) is an NE that maximises the sum of the players' payoffs considering the game where all payoffs are negated (thus, minimising the sum in the original game). If we consider again the prisoner's dilemma example, a SWNE would be given by the strategy profile where both players defect and get -3 each. Conversely, the only SCNE is given when both players *cooperate* (by playing c_i, that is), which would yield a payoff of -1 for each. Note that, the SWNE equilibrium gives a lower value although it is characterised by the intention to maximise the sum of payoffs. We introduce SCNE for convenience, in addition to working with the classical notion of SWNE. We highlight that our definition of SCNE is based on a broader definition of an NE, that is, a strategy profile from which no player has an incentive to deviate but considering that the players' objective is to minimise their rewards instead of maximising them. Differently from SWNE, which can be computed by finding the set of NE and selecting the one that maximises the sum of payoffs, SCNE is computed over a modified game and therefore does not constitute a subset of the NE found in the original game. A formal definition is presented in Chapter 5.

Backward induction. *Backward induction* is a method for computing Nash equilibria. When applied to a game tree description such as that used for the two-player prisoner's dilemma of Figure 3.1, it works by identifying the equilibria at the leaf nodes of the tree and propagating them up the tree. Figure 3.3 shows the application of backward induction to compute a sample Nash equilibrium for the two-player prisoner's dilemma. We start at nodes identified by the label p_2, where we consider choices of player 2. On the left node, player 2 would choose d_2 over c_2 given that,

assuming that player 1 would play c_1, it would be preferable to get 0 than -1. Similarly, considering the choices on the right node, assuming that player 1 would play d_1, player 2 would also choose d_2 over c_2 as it would be preferable to get -3 than -4. We then propagate the payoff vectors upwards and apply the same reasoning when considering the choices of player 1, arriving at the equilibrium payoff vector of $(-3, -3)$. A detailed presentation and algorithm for backward induction can be found in [188].

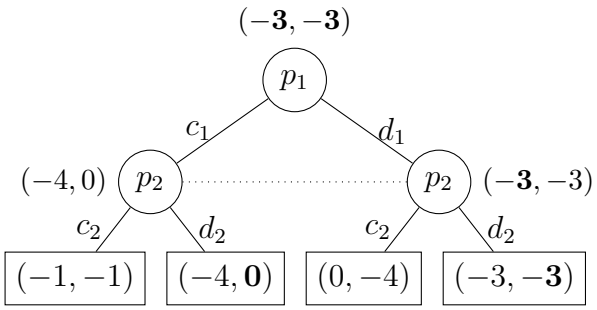


Figure 3.3: Backward induction on two-player prisoner's dilemma.

Iterated games. *Iterated* or *repeated* games are those where players play a known or unknown number of rounds of the same normal form game. For a known number of rounds, these games can also be represented by trees. Otherwise, the extensive game representation using trees is no longer possible, as it could result in an infinite tree and thus without terminal nodes. Given the overall utility function for such games is potentially infinite, it is common to apply concepts such as *average reward* and *discounted reward* to define a player's payoff. Average reward is computed as the limit of the sum of *stage* payoffs divided by the number of rounds, as it grows to infinity. On the other hand, a discounted reward is computed by summing *stage* payoffs multiplied by a factor strictly smaller than one to the power of the number of rounds already played. Provided the utility function for iterated games can be expressed by either criterion, it is also possible to reason about dominated strategies and equilibria in a similar way as for one-shot normal form games. In Chapter 4, we introduce *stochastic games*, a representation that can be used for defining both finite and infinite iterated games.

3.3 Stochastic Models

In this section, we review simple probabilistic models that are needed for developing our model checking and strategy synthesis algorithms. More specifically, we focus

on models capable of specifying nondeterministic choices, which serve as a basis for subsequently defining the stochastic games modelling formalism.

3.3.1 Markov Decision Processes

Markov decision processes (MDPs) are a class of models that allow for the specification of both probabilistic and *nondeterministic* behaviour. MDPs are commonly described as transition systems with *actions* and probability values associated to the connecting edges. The possible next states of a system modelled by and MDP is determined by the action chosen by an agent or *controller* and the corresponding probability distribution. For that reason, this class of models is also known as $1\frac{1}{2}$-games, considering the fact that, although the agent has control over the action chosen at each state, the next state is determined probabilistically according to a distribution over the set of states. Here we consider a *labelled* version of MDPs, where each state, and thus each possible configuration of the system being modelled, can be identified by a set of *labels*. Labels can be defined as predicates over variables or just simple tags. More formally, we have:

Definition 3.3.1 (Markov decision process)**.** A *Markov decision process* (MDP) is a tuple $\mathsf{M} = (S, \bar{S}, A, \delta, AP, L)$ where:

- S is a finite set of states and $\bar{S} \subseteq S$ is a set of initial states;
- A is a finite set of actions;
- $\delta \colon S \times A \to \mathit{Dist}(S)$ is a partial probabilistic transition function;
- AP is a set of atomic propositions;
- $L \colon S \to 2^{AP}$ is a labelling function.

The progression of a system modelled as an MDP depends on the agent for resolving the nondeterministic choices at each state. Naturally, it is possible to define an MDP for which only one choice is present at each state. Probabilistic models with that characteristic are known as Discrete-time Markov chains (DTMCs). DTMCs are also widely used in probabilistic verification and as a modelling paradigm in many domains of science. They possess what is called *Markov* or *memoryless property*, that is, their next state depends solely on the current state and the probabilities associated to the transition function. Here we define DTMCs as a restricted class of MDPs as follows:

Definition 3.3.2 (Discrete-time Markov chain)**.** An MDP $\mathsf{M} = (S, \bar{S}, A, \delta, AP, L)$ is a DTMC if there is precisely one action enabled in a given state. Given there is no nondeterminism in a DTMC, the action set A can also be dropped from the definition.

The behaviours of a system modelled as a MDP are represented by *runs* or *paths* over the MDP. A path over an MDP M is a possible infinite sequence of state-action pairs $\pi = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} \ldots$ such that for all $i \geqslant 0$, $s_i \in S$, $\delta(s_i, a_i)(s_{i+1}) > 0$. In the following, we denote $\pi(i)$ the $(i{+}1)$th state of π and by $\pi[i]$ the action associated with the $(i{+}1)$th transition. Throughout this document, we refer to the set of finite and infinite paths of a MDP M as $FPaths_{\mathsf{M}}$ and $IPaths_{\mathsf{M}}$, respectively. Similarly, we write $FPaths_{\mathsf{M},s}$ and $IPaths_{\mathsf{M},s}$ to denote the sets of finite and infinite paths starting in a state s. The set of terminal states is defined as the set $\{s \in S \mid \forall a \in A, \delta(s,a) = s\}$. An *end-component* of an MDP M is a subset of states $S' \subseteq S$ such that, for all pairs $s, t \in S'$ there exists a strategy for reaching t from s and a strategy for reaching s from t both with probability one. A *maximal end-component* is an end-component that is not contained by any other end-component. Lastly, a *bottom end-component* is a maximal end-component that is closed under probabilistic branching.

In order to investigate and reason about the behaviours of a system modelled as an MDP, a *probability measure* can be defined over its set of infinite paths using the *σ-algebra* generated by the *cylinder sets* of finite paths [128]. With the purpose of defining a probability measure over paths in an MDP, we first need to have a way to resolve nondeterministic choices. As for NFGs, this will also be done by means of a *strategy*.

Definition 3.3.3 (Strategy)**.** A *strategy* σ for an MDP M maps finite paths to a distribution over the action set, i.e., its is a function of the form $\sigma : FPaths_{\mathsf{M}} \to Dist(A)$ such that if $\sigma(\pi)(a) > 0$ then $a \in A(last(\pi))$. The set of all strategies is denoted as Σ.

A strategy σ and an initial state s of an MDP M induces a possibly infinite DTMC, on which a probability measure $Prob_{\mathsf{M},s}$ can be defined. A cylinder set $C(\pi)$ of a finite path $\pi = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} \ldots$ is the set of infinite paths with prefix π. The probability associated to a cylinder set $C^\sigma(\pi)$ of a finite path $\pi = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \ldots \xrightarrow{a_{k-1}} s_k$ of length $k+1$ is defined as:

$$\mathbf{P}^\sigma(\pi) = \prod_{i=0}^{k-1} \sigma(\pi' \cdot s_i)(a_i) \cdot \delta(s_i, a_i)(s_{i-1})$$

where π' is a prefix of π.

We can then associate a probability space $(IPaths^{\sigma}_{\mathsf{M},s}, \mathcal{F}^{\sigma}_{s}, Prob^{\sigma}_{\mathsf{M},s})$ with an MDP M in order to compute the probabilities of events or runs in a system. We define a set of outcomes as the infinite paths $IPaths_{\mathsf{M}}$ of M. The set $\mathcal{F}^{\sigma}_{s}$ is the smallest σ-algebra that contains all cylinder sets of all $\pi \in FPaths_{\mathsf{M}}$, and $Prob^{\sigma}_{\mathsf{M},s}$ is the unique measure such that $Prob^{\sigma}_{\mathsf{M},s}(C^{\sigma}(\pi)) = \mathbf{P}^{\sigma}(\pi)$ for all $\pi \in FPaths^{\sigma}_{\mathsf{M},s}$.

Example 1. Consider the MDP shown in Figure 3.4. It is defined by the set of states $S = \{s_0, s_1, s_2, s_3, s_4, s_5\}$, with an initial state s_0. State s_4 is labelled with atomic propositions a and b ($L(s_4) = \{\mathsf{a}, \mathsf{b}\}$), state s_2 is labelled with atomic proposition c and all other states have an empty set of labels. The model has one terminal state s_3. The probabilistic transition function is defined to accommodate the choice of actions on states, for instance, for state s_4 it can be represented as $\delta(s_4, w) = [\frac{1}{2} \mapsto s_3, \frac{1}{2} \mapsto s_5]$ and $\delta(s_4, e) = [1 \mapsto s_1]$. If we consider a memoryless strategy $\sigma(\pi \cdot s_0) = [1 \mapsto e]$, $\sigma(\pi \cdot s_4) = [1 \mapsto w]$ for all $\pi \in FPaths_{\mathsf{M}}$ and the initial state s_0, we then have an infinite path $s_0 \xrightarrow{e} s_1 \xrightarrow{w} s_4 \xrightarrow{w} (s_3 \xrightarrow{s})^{\omega}$ over the MDP. The probability of this path equals $\frac{1}{2} \cdot \frac{1}{2} = \frac{1}{4}$. We omit the definition of the strategy for states where only one action can be chosen. There are three end-components, $\{s_3\}$ and $\{s_2, s_5\}$ and $\{s_0, s_1, s_4\}$.

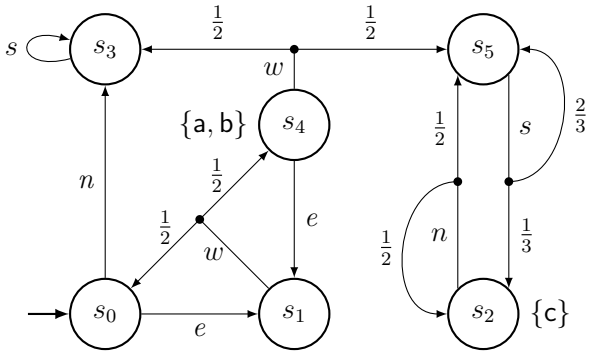


Figure 3.4: An example of an MDP.

■

3.4 Properties of Stochastic Models

In this section, we review a set of commonly studied behavioural properties of probabilistic models. The types of properties considered here can also be applied to stochastic games and will be fundamental to the understanding of the model checking specifications and algorithms presented later. A *specification* is a way to formally

define desirable properties of a model. Based on a specification, it is possible to synthesise strategies that assure the system will behave accordingly, or run qualitative/quantitative analysis over the system's model. Naturally, strategy synthesis only makes sense for nondeterministic systems such as MDPs and games, where it is necessary to also specify whether we are usually interested in computing the maximal or minimal expected value associated to the occurrence of an event, or even consider different criteria such as discounting and average. We will focus on properties based on reachability and reward-based objectives and give the corresponding problem formulation.

3.4.1 Probabilistic Reachability

Reachability is one of the most fundamental types of properties used in model checking. In probabilistic models, verifying reachability properties consists in computing, for a given initial state, the probability of reaching a state or set of states at some point during the execution of the system. More formally, given an initial state s and a set of target states $T \subseteq S$, the probability of eventually reaching T from s in a MDP M for a given strategy σ is the probability measure associated to the paths starting in s that also contains a state in T, that is, $Prob^{\sigma}_{\mathsf{M},s}(\{\pi \in FPaths^{\sigma}_{\mathsf{M},s} \mid \exists i \in \mathbb{N} \,.\, \pi(i) \in T\})$. Probabilistic reachability properties can be *bounded* and *unbounded*. Bounded properties are parameterised by an integer representing the length of the paths to be considered. As we are dealing with discrete models, each transition represents one time step, and therefore limiting the length of the paths corresponds to limiting the time horizon that will be investigated. On the other hand, unbounded properties are computed on potentially infinite paths, which represent the long-run behaviour of a system.

3.4.2 Reward-based Properties

The notion of rewards or *costs* are used throughout verification as a way to measure the impact of occurrences of a particular event during a system's execution. It is also frequently used in AI applications as a way to incentivise or discourage a given behaviour, as it is done for instance in reinforcement learning. We already presented a similar notion in the context of normal form games, which then we called payoffs or utilities. Here, we similarly define a *reward structure* as a tuple of *reward functions* $r = (r_S, r_A)$, where $r_S : S \to \mathbb{Q}$ is a *state reward function* (which maps a state to a rational value that is accumulated whenever the state is reached in a path) and an

action reward function $r_A : S \times A \to \mathbb{Q}$ (that associates state and action tuple pairs to a rational value that is accumulated when the action is selected in the state). It is worth noting that action rewards are undefined for DTMCs.

During an execution of a model, as states are visited and choices are made, rewards are accumulated and we can study properties based on different criteria. In this section we will focus on *instantaneous*, *cumulative*, *total* and *expected reachability* rewards, which will be used later for developing our model checking algorithms. As it is the case for probabilistic reachability, reward properties can be divided into bounded and unbounded, with instantaneous and cumulative belonging to the first group and total and expected reachability to the second. In the following, we define a random variable $X : IPaths \to \mathbb{Q}$, which we call an *objective* (or utility function), and compute rewards by calculating the *expected* value of that variable over paths starting from a state s, denoted $\mathbb{E}_s[X]$, according to the type of property in question.

Cumulative Rewards. Cumulative rewards allow us to compute the sum of payoffs accumulated during the system's execution up to a specific time horizon. In order to compute such values we unroll the execution of the system by calculating the weighted sum of action and state rewards obtained in paths of a given length. Formally, the problem involves defining a random variable $X(\pi) = \sum_{i=0}^{k-1}(r_A(\pi(i), \pi[i]) + r_S(\pi(i)))$ for paths π of length k. The reward values are then computed by multiplying these variables by the probabilities associated to these paths.

Instantaneous Rewards. Instantaneous rewards allow us to reason about a payoff obtained in a particular moment of the system's execution. As states in a discrete system represent a specific point in time, for probabilistic systems the problem is to compute the weighted sum of the state rewards obtained after the number of transitions specified by the bound. Formally, we define a random variable $X(\pi) = r_S(\pi(k))$ for some instant $k \in \mathbb{N}$ and multiply these values by the probability of reaching the corresponding states at $\pi(k)$.

Expected Total Rewards. Expected total reward properties are extensively used in probabilistic model checking. They are often employed as a means to estimate resource consumption and running time over the system execution. Given they are not bounded by a specific event or time horizon, these type of properties may not be applicable to all systems and often rely on assumptions about how rewards are accumulated, the underlying graph structure or are alternatively computed using a *discount factor*. Here, we compute the reward of a path as the sum of action and state rewards on that path. For example, for a MDP M, a strategy σ and state $s \in S$, the

expected total reward under σ is expressed as $\mathbb{E}_s^\sigma[X] = \int_{\pi \in Ipaths^\sigma_{\mathsf{M},s}} X dProb^\sigma_{\mathsf{M},s}$, where $X(\pi) = \sum_{i=0}^{\infty} \big(r_A(\pi(i), \pi[i]) + r_S(\pi(i))\big)$.

Expected Reachability. Expected reachability is unbounded with respect to time but bounded in the sense they are computed up to a specific point of the execution of the system, which is expressed by a set of target states. Naturally, there can be different ways of computing such values, concerning more specifically the measures associated to paths that never reach a target. Here, we will use a definition that associates an infinite value to paths that do not reach a state from the target set.

3.4.3 Parity Objectives

Parity objectives can encode any ω-regular path properties such as Büchi and Co-Büchi, which can be used to express *repeated reachability*, *persistence* and other properties. Given a model and a parity function of the form $P : S \to \{0, \ldots, d-1\}$, where d is the number of priorities, a parity objective expresses the probability that a path satisfies the parity condition, that is, that the lowest priority of the states occurring infinitely often in the path is even. We will make use of this type of objectives in pre-computation algorithms for computing zero-sum properties as described in Chapter 4.

3.5 The Logic PCTL

In this section, we present PCTL (Probabilistic Computation Tree Logic) [107, 134], a probabilistic extension of the logic CTL. Although it is used to express properties for both DTMCs and MDPs, here we present the semantics for the latter only. The case for DTMCs follows similarly with the simplification of not quantifying over strategies.

Definition 3.5.1 (Probabilistic Computation Tree Logic)**.**

$$\begin{aligned} \phi &:= \texttt{true} \mid \texttt{a} \mid \neg\phi \mid \phi \wedge \phi \mid \texttt{P}_{\sim q}[\,\psi\,] \mid \texttt{R}^r_{\sim x}[\,\texttt{F}\ \phi\,] \\ \psi &:= \texttt{X}\,\phi \mid \phi\ \texttt{U}\ \phi \end{aligned}$$

where $\texttt{a} \in AP$ is an atomic proposition, $\sim \in \{<, \leqslant, \geqslant, >\}$, $q \in [0,1]$, $x \in \mathbb{Q}$, r is a reward structure.

Similarly to CTL, PCTL is a branching-time temporal logic, where the formulae are divided into state (ϕ) and path (ψ) formulae. In PCTL, we can define the usual equivalences $\texttt{F}\ \phi \equiv \texttt{true}\ \texttt{U}\ \phi$, $\texttt{G}\ \phi \equiv \neg(\texttt{F}\ \neg\phi)$, etc. We write $s \models \phi$, for some state $s \in$

S, to denote that s satisfies the state formula ϕ. Similarly, for a path $\pi \in IPaths_{\mathsf{M}}$, we write $\pi \models \psi$ to denote that π satisfies formula ψ. The satisfaction relation $\models$ for PCTL is defined inductively for all sates of MDP $\mathsf{M} = (S, \bar{S}, A, \delta, AP, L)$ as follows:

$$\begin{aligned}
s &\models \texttt{true} && \text{always} \\
s &\models ap && \Leftrightarrow \quad ap \in L(s) \\
s &\models \neg\phi && \Leftrightarrow \quad s \not\models \phi \\
s &\models \phi_1 \wedge \phi_2 && \Leftrightarrow \quad s \models \phi_1 \text{ and } s \models \phi_2 \\
s &\models \mathtt{P}_{\sim q}[\,\psi\,] && \Leftrightarrow \quad \forall \sigma \in \Sigma \,.\, Prob^{\sigma}_{\mathsf{M},s}(\psi) \sim p \\
s &\models \mathtt{R}^{r}_{\sim x}[\,\mathtt{F}\ \phi\,] && \Leftrightarrow \quad \forall \sigma \in \Sigma \,.\, \mathbb{E}^{\sigma}_{\mathsf{M},s}[X] \sim q
\end{aligned}$$

where, for any path $\pi \in IPaths_{\mathsf{M}}$:

$$\begin{aligned}
\pi &\models \mathtt{X}\,\phi && \Leftrightarrow \quad \pi(1) \models \phi \\
\pi &\models \phi_1\ \mathtt{U}\ \phi_2 && \Leftrightarrow \quad \exists i \in \mathbb{N}.\, (\pi(i) \models \phi_2 \wedge \forall j < i.\, \pi(j) \models \phi_1)
\end{aligned}$$

In the definition above, we denote by $Prob^{\sigma}_{\mathsf{M},s}(\phi)$ the probability of paths satisfying ψ under σ, that is, $Prob^{\sigma}_{\mathsf{M},s}(\psi) = Prob^{\sigma}_{\mathsf{M},s}(\{\pi \in IPaths^{\sigma}_{\mathsf{M},s} \mid \pi \models \psi\})$. The random variable for expected reachability formula $\mathtt{R}^{r}_{\sim x}[\,\mathtt{F}\ \phi\,]$ is as follows:

$$X(\pi) = \begin{cases} \sum_{i=0}^{k_\phi} \left(r_A(\pi(i), \pi[i]) + r_S(\pi(i))\right) & \text{if } \pi(j) \in T \text{ for some } j \in \mathbb{N} \\ \infty & \text{otherwise} \end{cases}$$

where $k_\phi = \min\{j - 1 \mid \pi(j) \in T\}$ and $T = \{s \in S \mid s \models \phi\}$, respectively.

3.5.1 Model Checking For PCTL

Model checking for PCTL formulae follows the same principle as other branching-time logics, where we recursively compute subsets of satisfying states starting from the most deeply nested sub-formulae. Computation of the set for atomic propositions, negation and conjunction is done in the usual manner. Also, we denote by $Sat(\phi) = \{s \in S \mid s \models \phi\}$ the set of states that satisfy formula ϕ. For the case of ϕ being of type $\mathtt{P}_{\sim q}[\,\psi\,]$ we can make use of the following equivalences as in [19, 172]:

$$\mathtt{P}_{\sim q}[\,\psi\,] \Leftrightarrow \inf_{\sigma \in \Sigma} Prob^{\sigma}_{\mathsf{M},s}(\psi) \sim q \quad \text{if } \sim \in \{\geqslant, >\}$$

and

$$\mathtt{P}_{\sim q}[\,\psi\,] \Leftrightarrow \sup_{\sigma \in \Sigma} Prob^{\sigma}_{\mathsf{M},s}(\psi) \sim q \quad \text{if } \sim \in \{\leqslant, <\}$$

which means that it is enough to compute the optimal values for the property over the MDP and compare it to the bound. The computation for $\psi = \mathtt{X}\,\phi$ can be performed as follows:

$$\inf_{\sigma\in\Sigma} Prob^{\sigma}_{\mathsf{M},s}(\mathtt{X}\,\phi) = \begin{cases} 0 & \text{if } \exists a \in A \;.\; supp(\delta(s,a)) \cap Sat(\phi) = \varnothing \\ \min\limits_{a\in A} \sum\limits_{s'\in Sat(\phi)} \delta(s,a)(s') & \text{otherwise.} \end{cases}$$

$$\sup_{\sigma\in\Sigma} Prob^{\sigma}_{\mathsf{M},s}(\mathtt{X}\,\phi) = \begin{cases} 1 & \text{if } \exists a \in A \;.\; supp(\delta(s,a)) \setminus Sat(\phi) = \varnothing \\ \max\limits_{a\in A} \sum\limits_{s'\in Sat(\phi} \delta(s,a)(s') & \text{otherwise.} \end{cases}$$

For the remaining operators, probability ($\mathtt{P}_{\sim q}[\,\phi_1\ \mathtt{U}\ \phi_2\,]$) and rewards ($\mathtt{R}^{r}_{\sim x}[\,\mathtt{F}\ \phi\,]$), model checking can be reduced to probabilistic reachability and expected total rewards, respectively. The computation for $Prob^{\sigma}_{\mathsf{M},s}(\phi_1\ \mathtt{U}\ \phi_2)$ can be done either by linear programming or value iteration as described in Section 3.6 by firstly performing the following reduction: we make all states in $s \notin Sat(\phi_1) \cup Sat(\phi_2)$ absorbing, and for the remaining states we compute the maximal/minimal probability to reach a state in $Sat(\phi_2)$. Lastly, we can make use of a similar equivalence when computing formulae of the type $\mathtt{R}^{r}_{\sim x}[\,\mathtt{F}\ \phi\,]$:

$$\mathtt{R}^{r}_{\sim x}[\,\mathtt{F}\ \phi\,] \Leftrightarrow \inf_{\sigma\in\Sigma} \mathbb{E}^{\sigma}_{\mathsf{M},s}[X] \sim p \quad \text{if } \sim \in \{\geqslant, >\}$$

and

$$\mathtt{R}^{r}_{\sim x}[\,\mathtt{F}\ \phi\,] \Leftrightarrow \sup_{\sigma\in\Sigma} \mathbb{E}^{\sigma}_{\mathsf{M},s}[X] \sim p \quad \text{if } \sim \in \{\leqslant, <\}$$

To then compute $\inf_{\sigma\in\Sigma} \mathbb{E}^{\sigma}_{\mathsf{M},s}[X]$, it is first necessary to identify the states for which this value is infinite. By definition of X, that can only happen if for all strategies the probability to reach reach T is less than one, which can be computed by checking the formula $\mathtt{P}_{<1}[\,\mathtt{F}\ \phi\,]$. For all other states, the value is finite and the computation of the minimum expected reward values can be achieved in the following way. Firstly, we set the reward for all states that satisfy ϕ to zero, that is, we construct a new reward structure r' such that, for all $a \in A$, $r'_S(s) = r'_A(s,a) = 0$ if $s \in Sat(\phi)$, and $r'_S(s) = r_S(s)$ and $r'_A(s,a) = r_A(s,a)$, otherwise. Secondly, we remove zero-reward end components, which can be done using the methods described in [72]. We can then compute the values for minimal expected total rewards over the modified MDP M' by taking into account the new reward structure r' following the methods detailed in Section 3.6.

Finally, in order to compute $\sup_{\sigma\in\Sigma} \mathbb{E}^{\sigma}_{\mathsf{M},s}[X]$, we again compute the set of states that satisfy $\mathtt{P}_{<1}[\,\mathtt{F}\ \phi\,]$. As previously, these states get the value of infinity because

there exists a strategy that does not reach T with probability one. For any other state, we build a new reward structure r' as before and compute maximal expected total reward values over the modified MDP M' following the methods described in Section 3.6. PCTL will be used to express properties for MDPs and we integrate model checking algorithms into the methods developed for computing equilibria properties for concurrent stochastic games in Chapter 5.

3.6 Model Checking Properties of MDPs

In this section, we describe model checking methods for probabilistic reachability and expected total rewards for MDPs. Model checking for these types of properties over DTMCs can be reduced to solving a set of linear equations [65]. The solution of the set of equations consists of the probabilistic reachability or expected total reward values for every state in the DTMC. Classical methods such as Gaussian elimination can be applied to solve the set of linear equations in polynomial time. Other solutions include iterative methods such as Jacobi and Gauss-Seidel. Alternatively, value iteration can also be applied to approximate such values.

As detailed in Section 3.5, verification of different PCTL operators can be reduced to these two types of objectives through modifications to the underlying graphs and reward structures. Here we show methods that can be used for both exact and approximate computation, which will also be applied to games in later sections.

3.6.1 Model Checking for MDPs

Linear Programming. Probabilistic reachability values can be computed for MDPs through linear programming [175, 86]. As for DTMCs, we define the sets S^1, S^0 and $S^?$, with the difference being that the set S^1 contains the states for which there is a strategy to reach the set of target states with probability 1, and the set S^0 the set of states from which it is not possible to reach a state belonging to the target set under any strategy. For all other states, maximal probability values of reaching a target state can be computed by finding the solution to the following linear program:
Minimise $\sum_{s \in S} x_s$ subject to:

$$\begin{array}{ll} x_s \geqslant \sum_{s' \in S} \delta(s,a)(s') \cdot x_{s'} & \text{if } s \in S^?, \text{for all } a \in A(s) \\ x_s = 1 & \text{if } s \in S^1 \\ x_s = 0 & \text{if } s \in S^0 \end{array}$$

which yields a solution containing maximal reachability probability values for every state of the DTMC.

Minimal probability values can be computed by finding the solution of the following linear program:
Maximise $\sum_{s \in S} x_s$ subject to:

$$\begin{array}{ll} x_s \leqslant \sum_{s' \in S} \delta(s,a)(s') \cdot x_{s'} & \text{if } s \in S^?, \text{for all } a \in A(s) \\ x_s = 1 & \text{if } s \in S^1 \\ x_s = 0 & \text{if } s \in S^0 \end{array}$$

which yields a solution containing minimal reachability probability values for every state of the DTMC.

Linear programming can also be used for computing expected total rewards values for MDPs. As for DTMCs, we define the sets of states S^0 and $S^?$, where S^0 is composed of the states from which a state with non-zero rewards is never reached. For every other state, we can compute maximal expected total reward values through the following linear program:
Minimise $\sum_{s \in S} x_s$ subject to:

$$\begin{array}{ll} x_s \geqslant r(s) + \sum_{s' \in S} r(s,a) + \delta(s,a)(s') \cdot x_{s'} & \text{if } s \in S^?, \text{for all } a \in A(s) \\ x_s = 0 & \text{if } s \in S^0 \end{array}$$

which yields the maximal expect total rewards values for each state in the MDP.

Likewise, minimal expected total reward values can be found by solving the following linear program:
Maximise $\sum_{s \in S} x_s$ subject to:

$$\begin{array}{ll} x_s \leqslant r(s) + \sum_{s' \in S} r(s,a) + \delta(s,a)(s') \cdot x_{s'} & \text{if } s \in S^?, \text{for all } a \in A(s) \\ x_s = 0 & \text{if } s \in S^0 \end{array}$$

which yields the minimal expect total rewards values for each state in the MDP.

Value Iteration. Value iteration [47] is a widely used method for approximating optimal values for properties of probabilistic models. Model checking algorithms that are based on value iteration can be described by a *Bellman operator* F, which characterises the set of optimal achievable targets x^* of an infinite horizon optimisation problem via the *Bellman equation* $x^* = F(x^*)$. As for DTMCs, it can be similarly used to approximate maximal and minimal values for probabilistic reachability and expected total reward objectives for MDPs [175], provided the appropriate modifications have been made to ensure convergence as explained in Section 3.5. Its application to MDPs is slightly more complex as, for each state, it is necessary to

iterate through the actions to find the one that maximises or minimises its value in the next iteration. For approximating probabilistic reachability values, we compute at each iteration an under-approximation for each state starting from $x_s = 0$ for all $s \in S \backslash T$. For the maximal case, we define the following Bellman operator:

$$F(x)(s) = \begin{cases} 1 & \text{if } s \in T, \\ \max\limits_{a \in A} \left(\sum\limits_{s' \in supp(\delta(s,a))} \delta(s,a)(s') \cdot x_{s'} \right) & \text{otherwise.} \end{cases}$$

Similarly, minimal expected reachability values can be approximated by applying the following operator:

$$F(x)(s) = \begin{cases} 1 & \text{if } s \in T, \\ \min\limits_{a \in A} \left(\sum\limits_{s' \in supp(\delta(s,a))} \delta(s,a)(s') \cdot x_{s'} \right) & \text{otherwise.} \end{cases}$$

As for probabilistic reachability, for the task of computing expected total rewards the method works by selecting the action that maximises the value for the objective at each state. As we do not have a set of target sates, we start from $x_s = 0$ for all $s \in S$. We then have the following operators:
For maximal expected rewards:

$$F(x)(s) = r_S(s) + \max_{a \in A} \left(r_A(s,a) + \sum_{s' \in supp(\delta(s,a))} \delta(s,a)(s') \cdot x_{s'} \right)$$

And for minimal expected rewards:

$$F(x)(s) = r_S(s) + \min_{a \in A} \left(r_A(s,a) + \sum_{s' \in supp(\delta(s,a))} \delta(s,a)(s') \cdot x_{s'} \right)$$

The bounds on convergence of value iteration algorithms as defined are exponential in the number of states of the MDP [47]. In practice, however, and also for DMTCs, the process is stoped earlier with some convergence criterion, e.g., $\max\limits_{s \in S} |\frac{x_s^k - x_s^{k-1}}{x_s^k}| < \varepsilon$ for *relative* difference or $\max\limits_{s \in S} |x_s^k - x_s^{k-1}| < \varepsilon$ for *absolute* difference. Methods used to solve reachability and other verification problems when encoded as linear systems or LPs can produce results that are more precise than those produced by value iteration. However, value iteration can be used for problems whose state space sizes would be hard for LP and linear solvers to handle while producing very good approximations.

Example 2. Let us examine again the MDP shown in Figure 3.4 and the PCTL property $\mathtt{P}_{max=?}[\,\mathtt{F}\ \mathtt{c}\,]$, which asks to compute the maximal probability of reaching a state s for which $L(s) \cap \{\mathtt{c}\} \neq \varnothing$. As for the DTMC, in this example we have $S^0 = \{s_3\}$, $S^1 = \{s_2, s_5\}$ and $S^? = \{s_0, s_1, s_4\}$. Note that this time we include s_5 in S_1 directly given that, by inspecting the graph, it is possible to see that the only path starting from s_5 that never visits s_2, that is, $\pi = (s_5 \xrightarrow{s})^\omega$ has an associated probability of $(\frac{2}{3})^\infty = 0$. For the states in $S^?$ the maximal reachability values can be computed via the following linear program:
Minimise $x_{s_0} + x_{s_1} + x_{s_4}$ subject to:

$$\begin{aligned} x_{s_0} &\geqslant x_{s_1} \\ x_{s_0} &\geqslant 0 \\ x_{s_1} &\geqslant \frac{1}{2} \cdot x_{s_0} + \frac{1}{2} \cdot x_{s_4} \\ x_{s_4} &\geqslant \frac{1}{2} \\ x_{s_4} &\geqslant x_{s_1} \end{aligned}$$

which gives the values $x_{s_0} = x_{s_1} = x_{s_4} = \frac{1}{2}$. Similarly, if we were interested in computing the value for the property $\mathtt{P}_{min=?}[\,\mathtt{F}\ \mathtt{c}\,]$, which asks for the minimal probability of reaching s_2, values could be computed by solving the LP:
Maximise $x_{s_0} + x_{s_1} + x_{s_4}$ subject to:

$$\begin{aligned} x_{s_0} &\leqslant x_{s_1} \\ x_{s_0} &\leqslant 0 \\ x_{s_1} &\leqslant \frac{1}{2} \cdot x_{s_0} + \frac{1}{2} \cdot x_{s_4} \\ x_{s_4} &\leqslant \frac{1}{2} \\ x_{s_4} &\leqslant x_{s_1} \end{aligned}$$

which gives the values $x_{s_0} = x_{s_1} = x_{s_4} = 0$. It is easy to confirm this is indeed the case as, from s_0, which is reachable with probability 1 from both s_1 and s_4, it is possible to pick the action n to reach s_3 with probability 1 and never leave that state. Value iteration can also be employed for approximating reachability values for MDPs. When computing the maximal probability, the algorithm selects at each state the action that maximises the probability of reaching a target state based on

the values of the previous iteration. For $\mathtt{P}_{max=?}[\,\mathtt{F\ c}\,]$ we would then have:

$$
\begin{array}{l}
\\
k{=}0 \\
k{=}1 \\
k{=}2 \\
k{=}3
\end{array}
\begin{array}{c}
\begin{array}{cccccc} x_{s_0} & x_{s_1} & x_{s_2} & x_{s_3} & x_{s_4} & x_{s_5} \end{array} \\
\begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix} \\
\begin{pmatrix} 0 & 0 & 1 & 0 & 0 & \frac{1}{3} \end{pmatrix} \\
\begin{pmatrix} 0 & 0 & 1 & 0 & \frac{1}{6} & \frac{5}{9} \end{pmatrix} \\
\begin{pmatrix} 0 & \frac{1}{12} & 1 & 0 & \frac{5}{18} & \frac{14}{27} \end{pmatrix}
\end{array}
$$

which would converge to the same values obtained by computing a solution to the LP. On the other hand, when trying to minimise probability values, we would have:

$$
\begin{array}{l}
\\
k{=}0 \\
k{=}1 \\
k{=}2 \\
k{=}3
\end{array}
\begin{array}{c}
\begin{array}{cccccc} x_{s_0} & x_{s_1} & x_{s_2} & x_{s_3} & x_{s_4} & x_{s_5} \end{array} \\
\begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix} \\
\begin{pmatrix} 0 & 0 & 1 & 0 & 0 & \frac{1}{3} \end{pmatrix} \\
\begin{pmatrix} 0 & 0 & 1 & 0 & 0 & \frac{5}{9} \end{pmatrix} \\
\begin{pmatrix} 0 & 0 & 1 & 0 & 0 & \frac{14}{27} \end{pmatrix}
\end{array}
$$

which would converge to the same values found when solving the LP. Notice that the only value that changes across the different iteration is s_5, which would converge according to the equation $x_{s_5}^{k+1} = \frac{2}{3} \cdot x_{s_5}^{k} + \frac{1}{3}$ to 1.

■

3.7 Summary

In this chapter, we discussed auxiliary definitions, concepts and some of the stochastic models as well as model checking methods we will make use of in the remainder of this thesis. We reviewed key concepts from game theory such as normal and extensive form games, strategies and equilibria. Moreover, we presented a formal definition of two classes of probabilistic models: DTMCs and MDPs. MDPs along with the corresponding model checking techniques are particularly relevant to the work presented in this thesis for several reasons: (i) stochastic games extend MDPs by either partitioning the state space among different players, in the case of turn-based games, or by allowing more than one agent to control the transitions made by the

system, in the case of concurrent games; (ii) value iteration and linear programming are also used when verifying games and are the basis for some of the model checking algorithms presented in subsequent chapters; (iii) we make direct use of MDP model checking when verifying two-player equilibria properties.

We have also presented PCTL, a branching-time specification logic for probabilistic models, and illustrated the connection between its semantics and the different model checking methods. Model checking for MDPs can be reduced to linear programming, which can be performed in polynomial time. Naturally, the set of properties and techniques presented in this chapter is not exhaustive and we refer the reader to [175, 13] for a comprehensive treatment of the matter. Finally, we show the application of exact and approximate methods for computing PCTL property values over MDPs using linear programming and value iteration, which illustrates the workings of different parts of the algorithms we will develop in the following chapters.

CHAPTER 4

Zero-sum Properties

Contents

In this chapter, we present the theoretical foundations for solving one-shot zero-sum (matrix) games and develop algorithms for computing zero-sum properties for stochastic games, which we also formally define. Our method for computing/approximating values for concurrent stochastic games builds on top of existing techniques for turn-based stochastic games, with the major difference being on how the value for each local game is computed. We also consider algorithms to compute reward-based properties with state and action rewards, a feature that general solutions as those to compute ω-regular objectives lacked.

Concurrent stochastic games are the most general class among probabilistic models and subsumes turn-based games, MDPs and DTMCs. At each state, multiple players can choose from a set of actions and the next state in the system is determined probabilistically according to the collective set of actions. In this chapter, we focus on

strictly competitive scenarios, which divide players into those who want to *maximise* the value associated to an objective and those who want to *minimise* that value. We formally define the two groups as separate *coalitions*, i.e., sets of players that may cooperate by synchronising their actions, which produces what we call a *coalitional game*.

We extend the logic rPATL (Probabilistic Alternating-time Temporal Logic with Rewards) to express properties over concurrent stochastic games, introduce an operator to compute instantaneous rewards and provide formal semantics for the property specification logic. We also discuss the pre-computation algorithms for computing unbounded properties and detail our method for strategy synthesis. The chapter is concluded by a discussion on the complexity and correctness of the algorithms. The material presented in this chapter is based on [136, 140].

4.1 Normal Form Games

In this section, we formally define games in normal or strategic form. Normal form games are simple one-shot games where players make their choices concurrently. The payoff for each player is then determined according to the collective choice.

Definition 4.1.1 (Normal form game)**.** A (finite, n-player) *normal form game* (NFG) is a tuple $\mathsf{N} = (N, A, u)$ where:

- $N = \{1, \ldots, n\}$ is a finite set of players;
- $A = A_1 \times \cdots \times A_n$ and A_i is a finite set of actions available to player $i \in N$;
- $u = (u_1, \ldots, u_n)$ and $u_i : A \to \mathbb{Q}$ is a utility function for player $i \in N$.

In a game N, players select actions simultaneously, with player $i \in N$ choosing from the action set A_i. If each player i selects action a_i, then player j receives the utility $u_j(a_1, \ldots, a_n)$.

Players may have a preference on which action to play and also choose to randomise over their set of actions, which can be formalised as a (mixed) strategy. A collection of individual strategies constitutes a strategy profile. A strategy profile eliminates nondeterminism in a normal form game and makes it possible to determine its outcome, i.e., the vector of expected payoffs.

Definition 4.1.2 (Strategies and strategy profile)**.** A (mixed) strategy σ_i for player i is a distribution over its action set, i.e. $\sigma_i \in \mathit{Dist}(A_i)$. A strategy is said to be *pure* if

its probability distribution is a Dirac distribution, and *mixed* otherwise. We denote by η_{a_i} the pure strategy that selects action a_i with probability 1 and Σ_{N}^i denote the set of strategies for player i. A *strategy profile* $\sigma{=}(\sigma_1, \ldots, \sigma_n)$ is a tuple of strategies, one for each player.

Under a strategy profile $\sigma{=}(\sigma_1, \ldots, \sigma_n)$ of a NFG N, the expected utility of player i is defined as follows:

$$u_i(\sigma) \stackrel{\text{def}}{=} \sum_{(a_1,\ldots,a_n)\in A} u_i(a_1, \ldots, a_n) \cdot \left(\prod_{j=1}^{n} \sigma_j(a_j)\right) .$$

4.1.1 Representing Games with Matrices

A two-player NFG is also called a *bimatrix game* as it can be represented by two distinct matrices $\mathsf{Z}_1, \mathsf{Z}_2 \in \mathbb{Q}^{l\times m}$, where $A_1{=}\{a_1, \ldots, a_l\}$, $A_2{=}\{b_1, \ldots, b_m\}$, z_{ij}^k is the element at the ith row and jth column of matrix Z_k, $z_{ij}^1 = u_1(a_i, b_j)$ and $z_{ij}^2 = u_2(a_i, b_j)$.

Definition 4.1.3 (Constant-sum and Zero-sum Games)**.** A two-player NFG is called a *constant-sum* game if there exists $c \in \mathbb{Q}$ such that $u_1(\alpha){+}u_2(\alpha) = c$ for all $\alpha \in A$ and *zero-sum* if $c = 0$.

A zero-sum NFG is often called a *matrix game* as it can be represented by a single matrix $\mathsf{Z} \in \mathbb{Q}^{l\times m}$, where $A_1 = \{a_1, \ldots, a_l\}$, $A_2 = \{b_1, \ldots, b_m\}$ and $z_{ij} = u_1(a_i, b_j) = -u_2(a_i, b_j)$. For zero-sum two-player NFGs, in bimatrix game representation, we then have $A_1 = -A_2$. In matrix and bimatrix games, it is also common to refer the *row* player and the *column* player when talking about players 1 and 2, respectively.

Example 3. Consider the NFG shown in Figure 4.1. It corresponds to the matrix game representation of a two-player *rock-paper-scissors* game. We identify each player's possible actions (*rock*, *paper* and *scissors*) by their first letters, which are also used to identify rows and columns. In this example, entry z_{12} represents the outcome when the row player plays rock while the column player plays paper, which yields a payoff of -1 to the row player indicating that he lost that round (and a corresponding payoff of $-z_{12}$ for the column player, indicating it won that round).

If we consider a mixed strategy profile $\sigma = (\sigma_1, \sigma_2)$, with $\sigma_1 = (\frac{2}{3}, \frac{1}{6}, \frac{1}{6})$ and $\sigma_2 = (\frac{1}{6}, \frac{1}{6}, \frac{2}{3})$, where $(\sigma_i)_j$ is the probability of the row/column player playing the

$$\mathsf{Z} = \begin{array}{c} \\ r \\ p \\ s \end{array} \begin{array}{c} \begin{array}{ccc} r & p & s \end{array} \\ \begin{pmatrix} 0 & -1 & 1 \\ 1 & 0 & -1 \\ -1 & 1 & 0 \end{pmatrix} \end{array}$$

Figure 4.1: Representation of a rock-paper-scissors game as a matrix game.

actions corresponding to row/column j, the payoffs for player 1 can be calculated as:

$$\begin{aligned} u_1(\sigma) = {} & u_1(r,p) \cdot \frac{2}{3} \cdot \frac{1}{6} + u_1(r,s) \cdot \frac{2}{3} \cdot \frac{2}{3} \\ & + u_1(p,r) \cdot \frac{1}{6} \cdot \frac{1}{6} + u_1(p,s) \cdot \frac{1}{6} \cdot \frac{2}{3} \\ & + u_1(s,r) \cdot \frac{1}{6} \cdot \frac{1}{6} + u_1(s,p) \cdot \frac{1}{6} \cdot \frac{1}{6} \\ = {} & \frac{1}{4} \end{aligned}$$

Given that rock-paper-scissors is a zero-sum game, the payoff for the column player is the negation of that of the row player, i.e., $u_2(\sigma) = -\frac{1}{4}$. It is easy to see that this strategy profile is prejudicial for the column player as the row player is more likely to play *rock* while he is less likely to player *paper*. As the rock-paper-scissors is both zero-sum and *symmetric*, there should be a strategy profile for which the payoff of both players is zero, which would indeed indicate they are both as likely to win a round. The payoff computed for the row player indicates, however, that he would win 25% of the rounds if both players followed this strategy profile.

■

Example 4. A stag hunt game is a classic example from game theory representing a situation in which two hunters go hunting in a forest. If the players cooperate, they can catch a stag, which is considered to be a high payoff. If one of the players chooses to cooperate while the other defects, the player who decided to cooperate gets nothing while the one who defects still has a chance of catching a hare. We modify this game as to have 3 hunters and two *coalitions*. Generally, a coalition is a grouping of players who can coordinate their actions and share a combined payoff (we present a formal definition in Section 4.4.1). In this example, we have two hunters in a coalition represented by the column player and one hunter represented by the row player. The players action sets are a_i and b_i with i representing the number of hunters in that coalition who are willing to cooperate. Figure 4.2 shows the representation of this scenario as a bimatrix game.

$$\mathsf{Z}_1 = \begin{array}{c} \\ a_0 \\ a_1 \end{array} \begin{array}{c} \begin{array}{ccc} b_0 & b_1 & b_2 \end{array} \\ \begin{pmatrix} 2 & 2 & 2 \\ 0 & 4 & 6 \end{pmatrix} \end{array} \qquad \mathsf{Z}_2 = \begin{array}{c} \\ a_0 \\ a_1 \end{array} \begin{array}{c} \begin{array}{ccc} b_0 & b_1 & b_2 \end{array} \\ \begin{pmatrix} 4 & 2 & 0 \\ 4 & 6 & 9 \end{pmatrix} \end{array}$$

Figure 4.2: Representation of a coalitional stag hunt game as a bimatrix game.

If we consider a mixed strategy profile where any number of players are as likely to cooperate, that is, $\sigma = (\sigma_1, \sigma_2)$ where $\sigma_1 = (\frac{1}{2}, \frac{1}{2})$ and $\sigma_2 = (\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$, the payoff for player i can be computed as:

$$u_i(\sigma) = (u_i(a_0, b_0) + u_i(a_0, b_1) + u_i(a_0, b_2) + u_i(a_1, b_0) + u_i(a_1, b_1) + u_i(a_1, b_2)) \cdot \frac{1}{2} \cdot \frac{1}{3}$$

which evaluates to $\frac{8}{3}$ and $\frac{25}{6}$ for the row and column players, respectively. It is easy to see that both coalitions could do better if all hunters cooperated or if half of the hunters in the coalition represented by the column player and the hunter represented by the row player cooperated, as they would get payoffs of 6 and 9, or 4 and 6, respectively.

■

NFGs are a powerful formalism despite having a rather simple definition. Repeated games or games played over graphs can also be represented as NFGs by combining the actions taken at different stages into single actions and indexing corresponding payoffs accordingly. It is not hard to see, however, that tables and matrices might not be the ideal representation if we are to consider games with millions of separate stages and actions. In Section 4.3, we consider formalisms which are better suited for that task.

4.2 Zero-sum Games and Minimax Strategies

In this section, we show how linear programming can be used to compute players' payoffs and optimal strategies in two-player zero-sum normal form games. We start by presenting a classical result from game theory concerning matrix games, which introduces the notion of the *value* of a matrix game (zero-sum NFGs).

Theorem 1 (Minimax theorem [204, 205])**.** For any zero sum NFG $\mathsf{N}=(N, A, u)$ and corresponding matrix game Z, there exists $v^\star \in \mathbb{Q}$, called the *value* of the game and denoted $val(\mathsf{Z})$, such that:

- there is a strategy $\sigma_1^\star$ for player 1, called an optimal strategy of player 1, such that under this strategy the player's expected utility is at least $v^\star$ regardless of the strategy of player 2, i.e. $\inf_{\sigma_2 \in \Sigma_{\mathsf{N}}^2} u_1(\sigma_1^\star, \sigma_2) \geqslant v^\star$;
- there is a strategy $\sigma_2^\star$ for player 2, called an optimal strategy of player 2, such that under this strategy the player's expected utility is at least $-v^\star$ regardless of the strategy of player 1, i.e. $\inf_{\sigma_1 \in \Sigma_{\mathsf{N}}^1} u_2(\sigma_1, \sigma_2^\star) \geqslant -v^\star$.

The optimal strategies $\sigma_i^\star$ are usually referred to as minimax strategy, which is one that minimises the other player's best case payoff. Similarly, the optimal value $v^\star$ is called the *minimax value*, corresponding to the maximum amount that the other player could achieve under player i's minimax strategy.

4.2.1 Computing Values of Matrix Games

The value of a matrix game $\mathsf{Z} \in \mathbb{Q}^{l \times m}$ can be found by solving the following linear programming (LP) problem [204, 205]. Maximise v subject to the constraints:

$$\begin{array}{rcl} v & \leqslant & x_1 \cdot z_{1j} + \cdots + x_l \cdot z_{lj} \text{ for } 1 \leqslant j \leqslant m \\ x_i & \geqslant & 0 \text{ for } 1 \leqslant i \leqslant l \\ x_1 + \cdots + x_l & = & 1 \end{array}$$

In addition, the solution for $(x_1, \ldots, x_l)$ yields an optimal strategy for player 1. The value of the game can also be calculated as the solution of the following dual LP problem. Minimise v subject to the constraints:

$$\begin{array}{rcl} v & \geqslant & y_1 \cdot z_{i1} + \cdots + y_m \cdot z_{im} \text{ for } 1 \leqslant i \leqslant l \\ y_j & \geqslant & 0 \text{ for } 1 \leqslant j \leqslant m \\ y_1 + \cdots + y_m & = & 1 \end{array}$$

and the solution $(y_1, \ldots, y_m)$ yields an optimal strategy for player 2.

Example 5. Let us again consider the rock-paper-scissor example previously discussed in Example 3 of Section 4.1.1. We remind the reader of its matrix game representation, where each player $i \in \{1, 2\}$ choses *rock* (r_i), *paper* (p_i) or *scissors* (s_i):

$$\begin{array}{c} \\ r \\ p \\ s \end{array} \begin{array}{c} \begin{array}{ccc} r & p & s \end{array} \\ \begin{pmatrix} 0 & -1 & 1 \\ 1 & 0 & -1 \\ -1 & 1 & 0 \end{pmatrix} \end{array}$$

The value for the matrix game is the solution of the following LP problem. Maximise v subject to the constraints:

$$\begin{aligned} v &\leqslant x_2 - x_3, \\ v &\leqslant x_3 - x_1, \\ v &\leqslant x_1 - x_2, \\ x_1 + x_2 + x_3 &= 1, \\ x_1, x_2, x_3 &\geqslant 0 \end{aligned}$$

which yields the solution $v^\star = 0$ with corresponding optimal strategy $\sigma_1^\star = (\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$ for player 1 (the optimal strategy for player 2 is the same).

■

4.3 Stochastic Games

Stochastic games are the class of models which will be the main focus of this thesis. Differently from MDPs, stochastic games allow for multiple agents to make choices at a given state and/or to differentiate among the set of agents that have a choice. Agents can be grouped into coalitions and synchronise their actions. Games can model both competition and cooperation between agents, in presence of uncertainty. In this section, we will cover the two main classes of stochastic games: *concurrent* and *turn-based*.

Definition 4.3.1 (Concurrent stochastic game)**.** A *concurrent stochastic multi-player game* (CSG) is a tuple $\mathsf{G} = (N, S, \bar{S}, A, \Delta, \delta, AP, L)$ where:

- $N = \{1, \ldots, n\}$ is a finite set of players;
- S is a finite set of states and $\bar{S} \subseteq S$ is a set of initial states;
- $A = (A_1 \cup \{\bot\}) \times \cdots \times (A_n \cup \{\bot\})$ where A_i is a finite set of actions available to player $i \in N$ and $\bot$ is an idle action disjoint from the set $\cup_{i=1}^{n} A_i$;
- $\Delta \colon S \to 2^{\cup_{i=1}^{n} A_i}$ is an action assignment function;
- $\delta \colon S \times A \to \mathit{Dist}(S)$ is a (partial) probabilistic transition function;
- AP is a set of atomic propositions and $L \colon S \to 2^{AP}$ is a labelling function.

In concurrent games, players move simultaneously, while in turn-based games, they alternate their moves. We define turn-based stochastic games as a restricted class of CSGs as follows:

Definition 4.3.2 (Turn-based stochastic game)**.** A *turn-based stochastic multi-player game* (TSG) is a special case of concurrent stochastic games in which the action assignment function can be defined as $\Delta\colon S \to 2^{A_i}$, i.e., at any given state at most one player has a choice. This restriction has also been commonly expressed by a partition of the state space $(S_i)_{i\in N}$ for which $\forall j \in N, s \in S_i, j \neq i \implies \Delta(s) \cap A_j = \varnothing$.

A CSG G starts in an initial state $\bar{s} \in \bar{S}$ and, when in state s, each player $i \in N$ selects an action from its available actions $A_i(s) \stackrel{\text{def}}{=} \Delta(s) \cap A_i$ if this set is non-empty, and from $\{\bot\}$ otherwise. For any state s and action tuple $a = (a_1, \ldots, a_n)$, the partial probabilistic transition function δ is defined for (s, a) if and only if $a_i \in A_i(s)$ for all $i \in N$. A path π of G is a sequence $\pi = s_0 \xrightarrow{\alpha_0} s_1 \xrightarrow{\alpha_1} \cdots$ where $s_i \in S$, $\alpha_i \in A$ and $\delta(s_i, \alpha_i)(s_{i+1}) > 0$ for all $i \geqslant 0$. A similar reasoning can be applied and the definitions can be extended to TSGs following the appropriate restrictions. A game is non-stochastic if $\delta(s, \alpha)(t) \in \{0, 1\}$ for all $s, t \in S$ and $\alpha \in A$. We follow the same nomenclature for paths as defined for MDPs and let $FPaths_{\mathsf{G}}$ and $IPaths_{\mathsf{G}}$ ($FPaths_{\mathsf{G},s}$ and $IPaths_{\mathsf{G},s}$) be the sets of finite and infinite paths (starting in state s). A CSG G is said to be *stopping* [54] if the set of terminal states is reached with probability one under any strategy profile.

In Examples 6, 7 and 8 we illustrate the differences between MDPs, TSGs and CSGs with a model of a robot moving over a grid. Uncertainty is introduced in this model to model obstacles on the grid that may divert the robot from its trajectory when it attempts to move in a given direction. We extend this example into a case study and further discuss it in Chapter 8.

Example 6. Consider the MDP shown in Figure 4.3. It illustrates a simplified scenario of a robot moving over a grid. Starting from the bottom left corner, its goal is to reach its base on the top right corner by moving north (n), east (e) or northeast (ne) at each time step. Whenever attempting to move in direction d, it succeeds with probability q_d and ends up in an adjacent cell with probability $1 - q_d$. The resulting model is an MDP M with four states $S = \{s_0, s_1, s_2, s_3\}$, one initial state $\bar{S} = \{s_0\}$, three possible actions $A = \{n, e, ne\}$ and a labelling function $L(s_3) = \{goal\}$. The state s_3 is the only terminal state and the set of available actions at each state changes in order to prevent it from going over the grid, for instance, while $\delta(s_0, a)$ is defined for all $a \in A$, $\delta(s_1, n)$ is undefined. An example of an infinite path in M is $\pi = s_0 \xrightarrow{e} (s_2 \xrightarrow{n} s_1 \xrightarrow{e})^{\omega}$, which has an associated probability of $q_e((1-q_n)(1-q_e))^{\infty}$.

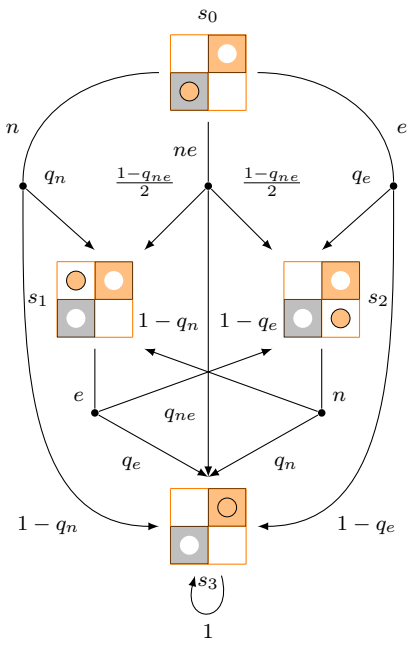


Figure 4.3: An MDP model of a robot moving over a grid.

■

Example 7. Consider the example shown in Figure 4.4. It models a scenario of two robots moving over a 2×2 grid as a two-player TSG. We denote by $robot_1$ and $robot_2$ the robot whose initial position is the bottom left corner and the top right corner of the grid, respectively. The robots move one cell at a time, via probabilisitic moves. We assign a unique value q_i ($i \in \{1, 2\}$) for each robot to end up in an adjacent cell regardless of the direction in it might be attempting to move in and their moves are prevented at the edge of the grid. The robots take turns at moving, with the colour of the grid indicating which robot is about to move, and a black grid indicating terminal states. The overall state space can then be partitioned into two sets $S_1 = \{s_0, s_4, s_6, s_8, s_9\}$ and $S_2 = \{s_1, s_2, s_{10}, s_{11}, s_{12}\}$, with the set of initial states $\bar{S} = \{s_0\}$. We assume that, if a robot moves into a cell already occupied by the other robot, they crash and are unable to move further. Their goal is to end up at a cell of their corresponding colour. To that effect, we define $L(s_{10}) = L(s_{11}) = L(s_{12}) = \{goal_1\}$, $L(s_6) = L(s_9) = L(s_{11}) = \{goal_2\}$ and $L(s_3) = L(s_5) = L(s_7) = \{crash\}$. We also have two action sets, with A_i being the actions available for $robot_i$, $A_1 = \{n, e, ne\}$ and $A_2 = \{w, s, sw\}$. An example of an infinite path over the TSG is $\pi = s_0 \xrightarrow{e} s_2 \xrightarrow{sw} s_8 \xrightarrow{n} s_{12} \xrightarrow{s} (s_{11} \xrightarrow{1})^\omega$.

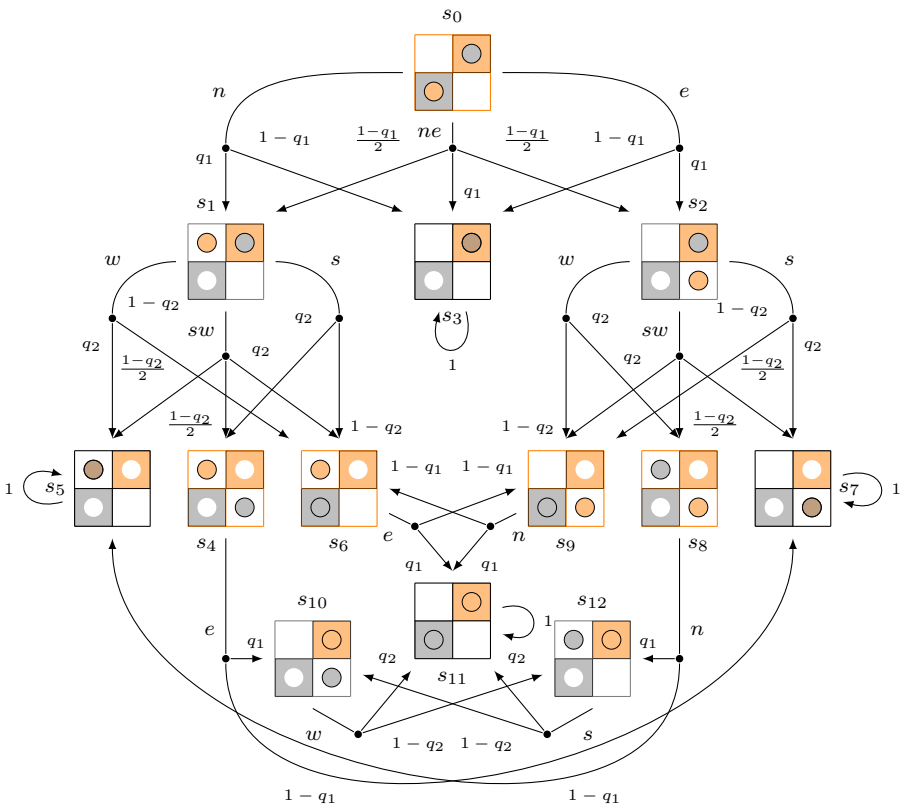


Figure 4.4: Turn-based example of two robots moving over a grid.

■

Example 8. The example shown in Figure 4.5 modifies the game of Figure 4.4 by considering simultaneous movement between the robots and thus transforming the model from a turn-based into a concurrent stochastic game. We refrain from showing diagonal movements and restrict the transitions to the initial state as the graphical representation of complete model is rather involved; for instance, we would have 9 transitions leaving from the initial state. There is one initial state $\bar{S} = \{s_0\}$, transitions are labelled by a vector of actions (a_1, a_2), with $a_i \in A_i$ and A_i the action set of the player representing $robot_i$. The corresponding probability values are placed either at the top or the bottom of the next states. Note that the next state probabilities are now the product of the individual probabilities of each robot failing or succeeding at moving in the direction it intends to move. Differently from a turn-based game, the probabilistic transition function is defined over states and products of actions sets. It is possible then to define a matrix game to represent the transition

function for each state.

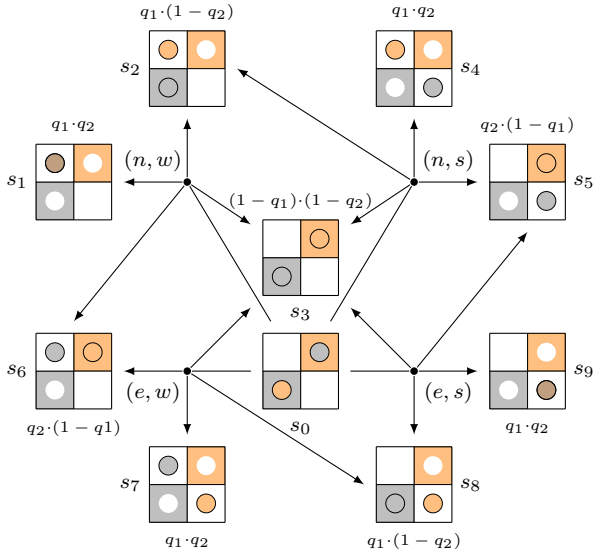


Figure 4.5: Concurrent example of two robots moving over a grid.

■

Definition 4.3.3 (Strategy)**.** A strategy for player i in a CSG G is a function of the form $\sigma_i \colon FPaths_{\mathsf{G}} \to Dist(A_i \cup \{\bot\})$ such that if $\sigma_i(\pi)(a_i) > 0$, then $a_i \in A_i(last(\pi))$. We denote by Σ^i_{G} the set of all strategies for player i.

As for NFGs, a *strategy profile* for G is a tuple $\sigma{=}(\sigma_1, \ldots, \sigma_n)$ of strategies for all players and, for a player i and strategy σ'_i, we define the sequence σ_{-i} and profile $\sigma_{-i}[\sigma'_i]$ in the same way. For a strategy profile σ and state s, we let $IPaths^{\sigma}_{\mathsf{G},s}$ denote the infinite paths from s under the choices of σ. A strategy σ and initial state $\bar{s}$ induce a possible infinite DTMC on which we can define the probability measure in the standard fashion.

Strategies for CSGs can be divided into different categories according to stochasticity and memory. A strategy for a CSG is said to be pure if *all* its probability distributions are Dirac distributions and mixed otherwise. Furthermore, a strategy is *memoryless* if the choices made only depend on the current state and use *finite* or *infinite* memory if the choices made also depend on the previous states that were visited up to a point. In the work developed in this thesis we will focus on memoryless and finite memory strategies. In order for a controller to implement a mixed strategy it needs to be capable of generating random numbers. Examples 9 and 10

illustrate the application of strategies to the problems considered in Examples 7 and 8, respectively.

Example 9. Let us consider again the TSG shown in Figure 4.4. If we consider that, under the restriction of staying within the boundaries of the grid, $robot_1$ always moves *north* when possible and otherwise moves *east*, and $robot_2$ always moves *south* when possible and otherwise moves *west*, we would have the following DTMC resulting from the application of this strategy profile to the TSG:

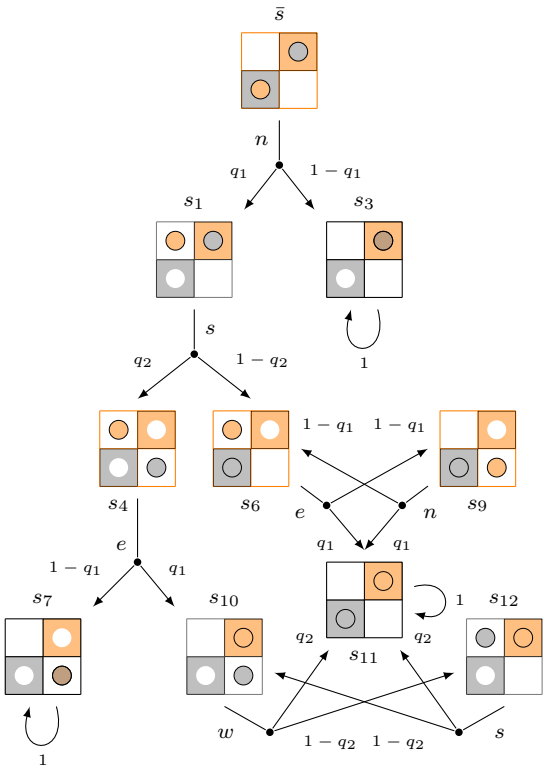


Figure 4.6: Turn-based example of two robots moving over a grid with a fixed strategy.

It is easy to see that this strategy is pure and memoryless for both players as it does not randomise among actions and only depends on the restrictions imposed by each state.

■

Example 10. Consider again the CSG shown in Figure 4.5. We again focus on the transitions from the initial state s_0. If we fix a pure strategy for $robot_1$, $\sigma_1(s_0) = [1 \mapsto e]$ and a mixed strategy for $robot_2$ $\sigma_2(s_0) = [y_1 \mapsto w, y_2 \mapsto s]$, that is, $robot_1$ will move *east* with probability 1 while $robot_2$ will move *east* with probability y_1 and *west*

with probability y_2, the resulting model would be the DTMC indicated in Figure 4.7. In the figure, the transitions that were labelled by the action-pairs (e, w) and (e, s) are now labelled with the resulting probabilities of applying both strategies from s_0. We also indicate the distribution over the next states below each grid, by considering the product of the strategies and the probabilities of ending up in a adjacent cell.

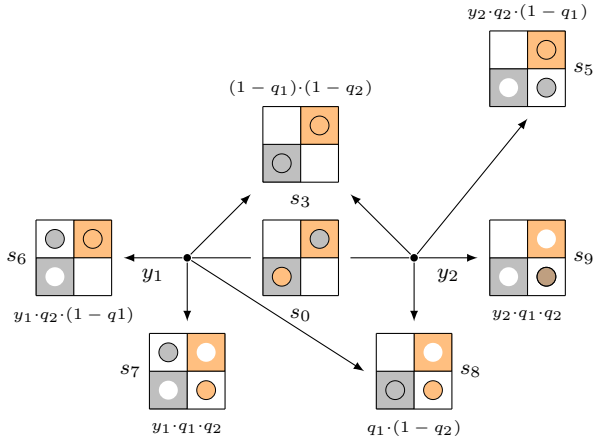


Figure 4.7: Concurrent example of two robots moving over a grid with a fixed strategy.

■

We can define a probability measure $Prob^{\sigma}_{\mathsf{G},s}$ over the infinite paths $IPaths^{\sigma}_{\mathsf{G},s}$. As before, this construction is based on first defining the probabilities for finite paths from the probabilistic transition function and choices of the strategies in the profile. More precisely, for a finite path $\pi = s_0 \xrightarrow{\alpha_0} s_1 \xrightarrow{\alpha_1} \cdots \xrightarrow{\alpha_{m-1}} s_m$ where $s_0 = s$, the probability of π under the profile σ is defined by:

$$\mathbf{P}^{\sigma}(\pi) \stackrel{\text{def}}{=} \prod_{j=0}^{m-1} \left(\left(\prod_{i=1}^{n} \sigma_i(\pi' \cdot s_j)(\alpha_j(i)) \right) \cdot \delta(s_j, \alpha_j)(s_{j+1}) \right).$$

were π' is a prefix of π. Next, for each finite path π, we define the basic cylinder $C^{\sigma}(\pi)$ that consists of all infinite paths in $IPaths^{\sigma}_{\mathsf{G},s}$ that have π as a prefix. Finally, using properties of cylinders, we can then construct the probability space $(IPaths^{\sigma}_{\mathsf{G},s}, \mathcal{F}^{\sigma}_{s}, Prob^{\sigma}_{\mathsf{G},s})$, where $\mathcal{F}^{\sigma}_{s}$ is the smallest σ-algebra generated by the set of basic cylinders $\{C^{\sigma}(\pi) \mid \pi \in FPaths^{\sigma}_{\mathsf{G},s}\}$ and $Prob^{\sigma}_{\mathsf{G},s}$ is the unique measure such that $Prob^{\sigma}_{\mathsf{G},s}(C^{\sigma}(\pi)) = \mathbf{P}^{\sigma}(\pi)$ for all $\pi \in FPaths^{\sigma}_{\mathsf{G},s}$.

Definition 4.3.4 (Reward structure)**.** We define reward structures for CSGs as for MDPs, i.e., they are composed of a pair functions of the form $r = (r_A, r_S)$, where $r_A : S \times A \to \mathbb{Q}$ is an action reward function and $r_S : S \to \mathbb{Q}$ is a state reward

function. Note that, as action functions are defined over the product set, a player's choice may influence the reward obtained by other players. We allow both positive and negative rewards; however, we will later impose certain restrictions to ensure the correctness of the presented model checking algorithm as detailed in Section 4.5.

As for NFGs (see Definition 1), for a two-player CSG G and a given objective X, we can consider the case where player 1 tries to maximise the expected value of X, while player 2 tries to minimise it. The above definition yields the *value* of G with respect to X if it is determined, i.e., if the maximum value that player 1 can ensure equals the minimum value that player 2 can ensure. Since the CSGs we consider are finite state and finitely-branching, it follows that they are determined for all the objectives we consider [155]. Formally we have the following.

Definition 4.3.5 (Determinacy and optimality)**.** For a two-player CSG G and objective X, we say that the game G is *determined* with respect to X if, for any state s:

$$\sup_{\sigma_1 \in \Sigma^1} \inf_{\sigma_2 \in \Sigma^2} \mathbb{E}^{\sigma_1,\sigma_2}_{\mathsf{G},s}(X) \quad = \quad \inf_{\sigma_2 \in \Sigma^2} \sup_{\sigma_1 \in \Sigma^1} \mathbb{E}^{\sigma_1,\sigma_2}_{\mathsf{G},s}(X)\,.$$

and call this the *value* of G in state s with respect to X, denoted $val_{\mathsf{G}}(s, X)$. Furthermore, a strategy $\sigma_1^\star$ of player 1 is *optimal* with respect to X if we have $\mathbb{E}^{\sigma_1^\star,\sigma_2}_{\mathsf{G},s}(X) \geqslant val_{\mathsf{G}}(s, X)$ for all $s \in S$ and $\sigma_2 \in \Sigma^2$ and a strategy of player 2 is *optimal* with respect to X if $\mathbb{E}^{\sigma_1,\sigma_2^\star}_{\mathsf{G},s}(X) \leqslant val_{\mathsf{G}}(s, X)$ for all $s \in S$ and $\sigma_1 \in \Sigma^1$.

Zero-sum games are inherently two-player games, given players are partitioned into two separate coalitions with opposing goals. The determinacy argument does not apply for multi-player games in general as players may consider different objectives entirely, as will be seen for nonzero-sum games in Chapters 5 and 6.

Example 11. Let us consider again the scenario in which robots moves concurrently over a grid (as in Example 8) of size $l \times l$. The robots start in diagonally opposite corners and try to reach the corner from which the other starts. A robot can move either diagonally, horizontally or vertically towards its goal. As before, obstacles which hinder the robots as they move from location to location are modelled stochastically according to a parameter q: when a robot moves, there is a probability that it instead moves in an adjacent direction. For $i \in \{1, 2\}$, we let goal_i be the atomic proposition labelling those states of G_l in which $robot_i$ has reached its goal and crash the atomic proposition labelling the states in which the robots have crashed, i.e., are in the same grid location.

Consider the objectives (X_1^k, X_2^k) where, for any infinite path π of G_l:

$$X_i^k(\pi) = \begin{cases} 1 & \text{if } \mathsf{goal}_i \in L(\pi(j)) \text{ for some } j \leqslant k \text{ and } \mathsf{crash} \notin L(\pi(m)) \text{ for all } m<j \\ 0 & \text{otherwise} \end{cases}$$

i.e., objective X_i^k returns 1 for paths on which $robot_i$ reaches its goal within k steps without crashing. So, the expected value of X_i^k for a given strategy profile equals the probability that $robot_i$ reaches its goal within k steps without crashing. Next, we consider the objectives Y_1 and Y_2 where, for any infinite path π of G_l:

$$Y_i(\pi) = \begin{cases} \sum_{j=0}^{k^i_{\min}} r(\pi(i)) & \text{if } \mathsf{goal}_i \in L(\pi(j)) \text{ for some } j \\ \infty & \text{otherwise} \end{cases}$$

where $k^i_{\min} = \min\{j \in \mathbb{N} \mid \mathsf{goal}_i \in L(\pi(j))\}$ and, for any state s, we have $r(s)=10$ if $\mathsf{crash} \in L(s)$ and $r(s)=1$ otherwise. The expected value of objective Y_i for a given strategy profile equals the expected number of steps that player i requires to reach their goal, where crashing incurs a delay of 10 steps.

■

4.4 Property Specification

In this section, we discuss the temporal logic rPATL (**P**robabilistic **A**lternating-time **T**emporal **L**ogic with **R**ewards), previously proposed for specifying properties of TSGs [53], and adapt it to CSGs.

Definition 4.4.1 (Extended rPATL syntax for zero-sum properties)**.** The syntax of rPATL is given by the grammar:

$$\begin{array}{lcl} \phi & \coloneqq & \mathtt{true} \mid \mathsf{a} \mid \neg\phi \mid \phi \wedge \phi \mid \langle\!\langle C \rangle\!\rangle \mathtt{P}_{\sim q}[\,\psi\,] \mid \langle\!\langle C \rangle\!\rangle \mathtt{R}^r_{\sim x}[\,\rho\,] \\ \psi & \coloneqq & \mathtt{X}\,\phi \mid \phi\ \mathtt{U}^{\leqslant k}\ \phi \mid \phi\ \mathtt{U}\ \phi \\ \rho & \coloneqq & \mathtt{I}^{=k} \mid \mathtt{C}^{\leqslant k} \mid \mathtt{C} \mid \mathtt{F}^c\ \phi \mid \mathtt{F}\ \phi \end{array}$$

where $\mathsf{a} \in AP$ is an atomic proposition, $C \subseteq N$ is a coalition of players, $\sim \in \{<, \leqslant, \geqslant, >\}$, $q \in [0,1]$, $x \in \mathbb{Q}$, r is a reward structure and $k \in \mathbb{N}$.

The logic rPATL is a branching-time temporal logic that combines the probabilistic operator $\mathtt{P}$ of PCTL [107], PRISM's reward operator $\mathtt{R}$ [135], and the coalition operator $\langle\!\langle C \rangle\!\rangle$ of ATL [4]. The rPATL syntax distinguishes between state (ϕ), path (ψ) and reward (ρ) formulae. State formulae are evaluated over states of a CSG, while path and reward formulae are both evaluated over paths.

The core operators from the existing version of rPATL [53] are $\langle\langle C\rangle\rangle \mathtt{P}_{\sim q}[\,\psi\,]$ and $\langle\langle C\rangle\rangle \mathtt{R}^{r}_{\sim x}[\,\rho\,]$. A state satisfies a formula $\langle\langle C\rangle\rangle \mathtt{P}_{\sim q}[\,\psi\,]$ if the coalition of players C can ensure the probability of the path formula ψ being satisfied is $\sim q$ and satisfies a formula $\langle\langle C\rangle\rangle \mathtt{R}^{r}_{\sim x}[\,\rho\,]$ if the players in C can ensure the expected value of the reward formula ρ for rewards structure r is $\sim x$, regardless of the actions of the other players $(N \backslash C)$ in the game. Such properties are inherently *zero-sum* in nature as one coalition tries to maximise an objective (e.g., the probability of ψ) and the other tries to minimise it; hence, we call these *zero-sum formulae*. The probabilistic objectives are specified by path formulae and the reward objectives are specified by both a reward structure and reward formulae.

For path formulae, we follow the existing rPATL syntax from [53] and allow *next* $(\mathtt{X}\,\phi)$, *bounded until* $(\phi\ \mathtt{U}^{\leqslant k}\ \phi)$ and *unbounded until* $(\phi\ \mathtt{U}\ \phi)$. We also allow the usual equivalences such as $\mathtt{F}\ \phi \equiv \mathtt{true}\ \mathtt{U}\ \phi$ (i.e., *probabilistic reachability*) and $\mathtt{F}^{\leqslant k}\ \phi \equiv \mathtt{true}\ \mathtt{U}^{\leqslant k}\ \phi$ (i.e., *bounded probabilistic reachability*). For reward formulae, the logic's syntax differs somewhat with respect to the existing syntax from [53] and that presented in [140]. Here, we allow instantaneous (state) reward at the kth step (*instantaneous rewards* $\mathtt{I}^{=k}$), reward accumulated over k steps (*bounded cumulative rewards* $\mathtt{C}^{\leqslant k}$), total cumulative reward (*total rewards* $\mathtt{C}$) and reward accumulated until a formula is satisfied (*expected reachability* $\mathtt{F}^{c}\ \phi$ and $\mathtt{F}\ \phi$). The first two, adapted from the property specification language of PRISM [135], were not previously included in rPATL, but proved to be useful for the case studies we present later in Chapter 8. The third, expected total rewards, was not included in earlier versions of rPATL and was only introduced in [136]. For the fourth, expected reachability, the rPATL variant presented in [53] defines for turn-based games three variants ($\mathtt{F}^{c}\ \phi$, $\mathtt{F}\ \phi$ and $\mathtt{F}^{0}\ \phi$), while the work presented in [140] only considers one ($\mathtt{F}^{c}\ \phi$). The semantics of the formulae $\mathtt{F}^{c}\ \phi$ and $\mathtt{F}\ \phi$, which we consider here, differs in how they define the rewards for paths that do not reach a state satisfying ϕ (for $\mathtt{F}^{c}$ it is the infinite sum of the state- and action-rewards of the path, while for $\mathtt{F}$ it is infinity). Adding the variant $\mathtt{F}^{0}\ \phi$ is straightforward based on the algorithm of [53]. Here we focus on the variants that are most commonly used and relevant for the experimental results reported in Chapter 8. The following example illustrates different types of zero-sum properties that can be investigated with rPATL.

Example 12. Recall the robot coordination problem from Example 8. The following are properties are examples of zero-sum formulae that could expressed with rPATL:

- $\langle\langle robot_1 \rangle\rangle \mathsf{P}_{\max=?}[\, \neg \mathsf{crash}\ \mathsf{U}^{\leqslant 10}\ \mathsf{goal}_1\,]$ which asks what is the maximum probability the first robot can ensure that it reaches its goal location within 10 steps and without crashing, no matter how the second robot behaves;
- $\langle\langle robot_2 \rangle\rangle \mathsf{R}^{r_{crash}}_{\leqslant 1.5}[\, \mathsf{F}\ \mathsf{goal}_2\,]$ states no matter the behaviour of the first robot, the second robot can ensure the expected number of times it crashes before reaching its goal is less that or equal to 1.5 (r_{crash} is a reward structure that assigns 1 to states labelled crash and 0 to all other states).

■

In order to give the semantics of the logic, we require the notion of *coalition games*, which was originally introduced for CSGs in [136]. The definition of coalition games presented in [136, 137] focused exclusively on *two-coalitional* games and is sufficient for the logics and algorithms presented in this chapter and Chapter 5. However, as we later extend our approach to *multi-coalitional* properties in Chapter 6, here we present an extension of the original definition to an *m-coalitonal* game [138], which accommodates multi-coalitional logic variant and the algorithms proposed in this thesis.

4.4.1 Coalition Games

Our formalism supports game playing for agents as well as coalitions of agents. We define *coalition games* which, given a CSG G and partition $\mathcal{C}$ of the players into m coalitions, reduces G to an m-player coalition game, where each player corresponds to one of the coalitions in $\mathcal{C}$. Without loss of generality, we assume $\mathcal{C}$ is of the form $\{\{1,\dots,n_1\},\{n_1{+}1,\dots n_2\},\dots,\{n_{m-1}{+}1,\dots n_m\}\}$ and let $j_{\mathcal{C}}$ denote player j's position in its coalition.

Definition 4.4.2 (Coalition game)**.** For CSG $\mathsf{G}{=}(N, S, \bar{s}, A, \Delta, \delta, AP, L)$ and partition of the players into m coalitions $\mathcal{C} = \{C_1, \dots, C_m\}$, we define the *coalition game* $\mathsf{G}^{\mathcal{C}}{=}(M, S, \bar{s}, A^{\mathcal{C}}, \Delta^{\mathcal{C}}, \delta^{\mathcal{C}}, AP, L)$ as an m-player CSG where:

- $M = \{1, \dots, m\}$;
- $A^{\mathcal{C}} = (A^{\mathcal{C}}_1 \cup \{\bot\}) \times \cdots \times (A^{\mathcal{C}}_m \cup \{\bot\})$;
- $A^{\mathcal{C}}_i = \big(\prod_{j \in C_i}(A_j \cup \{\bot\}) \setminus \{(\bot, \dots, \bot)\}\big)$ for all $i \in M$;
- for any $s \in S$ and $i \in M$: $a^{\mathcal{C}}_i \in \Delta^{\mathcal{C}}(s)$ if and only if either $\Delta(s) \cap A_j = \varnothing$ and $a^{\mathcal{C}}_i(j_{\mathcal{C}}) = \bot$ or $a^{\mathcal{C}}_i(j_{\mathcal{C}}) \in \Delta(s)$ for all $j \in C_i$;

- for any $s \in S$ and $(a_1^{\mathcal{C}}, \ldots, a_m^{\mathcal{C}}) \in A^{\mathcal{C}}$: $\delta^{\mathcal{C}}(s, (a_1^{\mathcal{C}}, \ldots, a_m^{\mathcal{C}})) = \delta(s, (a_1, \ldots, a_n))$ where for $i \in M$ and $j \in C_i$ if $a_i^{\mathcal{C}}{=}\bot$, then $a_j{=}\bot$ and otherwise $a_j{=}a_i^{\mathcal{C}}(j_{\mathcal{C}})$.

Furthermore, for a reward structure $r = (r_A, r_S)$, by abuse of notation we use $r = (r_A^{\mathcal{C}}, r_S^{\mathcal{C}})$ for the corresponding reward structure of $\mathsf{G}^{\mathcal{C}}$ where:

- for any $s \in S$, $a_i^C \in A_i^C$: $r_{A^{\mathcal{C}}}^{\mathcal{C}}(s, (a_1^C, \ldots, a_m^C)) = r_A(s, (a_1, \ldots, a_n))$ where for $i \in M$ and $j \in C_i$, if $a_i^{\mathcal{C}} = \bot$, then $a_j{=}\bot$ and otherwise $a_j{=}a_i^{\mathcal{C}}(j_{\mathcal{C}})$;
- for any $s \in S$: $r_S^{\mathcal{C}}(s){=}r_S(s)$.

Our logic includes both *finite-horizon* ($\mathtt{X}$, $\mathtt{U}^{\leqslant k}$, $\mathtt{I}^{=k}$, $\mathtt{C}^{\leqslant k}$) and *infinite-horizon* ($\mathtt{U}$, $\mathtt{F}$, $\mathtt{C}$, $\mathtt{F}^c$) temporal operators. For the latter, the existence of optimal strategies is not guaranteed [75] but we can check for ε-optimality for any ε. Hence, we define the semantics of the logic in the context of a particular ε which we assume to be specified beforehand. Throughout this chapter and the next, given we are concerned exclusively with two-player formulae, we fix two coalitions C and $N\backslash C$ by letting $\mathcal{C} = \{C, N\backslash C\}$ when building the coalition game $\mathsf{G}^{\mathcal{C}}$.

Definition 4.4.3 (Extended rPATL semantics for zero-sum properties)**.** The satisfaction relation $\models$ of our rPATL extension is defined inductively on the structure of the formula. The propositional logic fragment ($\mathtt{true}$, $\mathtt{a}$, $\neg$, $\wedge$) is defined in the usual way. For temporal operators and a state $s \in S$ in CSG G, we have:

$$\begin{aligned} s \models \langle\!\langle C \rangle\!\rangle \mathtt{P}_{\sim q}[\,\psi\,] &\;\Leftrightarrow\; \exists \sigma_1 \in \Sigma^1.\, \forall \sigma_2 \in \Sigma^2.\, \mathbb{E}^{\sigma_1,\sigma_2}_{\mathsf{G}^{\mathcal{C}},s}(X^{\psi}) \sim q \\ s \models \langle\!\langle C \rangle\!\rangle \mathtt{R}^{r}_{\sim x}[\,\rho\,] &\;\Leftrightarrow\; \exists \sigma_1 \in \Sigma^1.\, \forall \sigma_2 \in \Sigma^2.\, \mathbb{E}^{\sigma_1,\sigma_2}_{\mathsf{G}^{\mathcal{C}},s}(X^{r,\rho}) \sim x \end{aligned}$$

where, for $\pi \in \mathit{IPaths}^{\sigma_1,\sigma_2}_{\mathsf{G}^{\mathcal{C}},s}$:

$$\begin{aligned} X^{\psi}(\pi) &= 1 \text{ if } \pi \models \psi \text{ and } 0 \text{ otherwise} \\ X^{r,\rho}(\pi) &= \mathit{rew}(r,\rho)(\pi) \end{aligned}$$

For a temporal formula and path $\pi \in \mathit{IPaths}_{\mathsf{G}^{\mathcal{C}},s}$:

$$\begin{aligned} \pi \models \mathtt{X}\,\phi &\;\Leftrightarrow\; \pi(1) \models \phi \\ \pi \models \phi_1\ \mathtt{U}^{\leqslant k}\ \phi_2 &\;\Leftrightarrow\; \exists i \leqslant k.\, (\pi(i) \models \phi_2 \wedge \forall j < i.\, \pi(j) \models \phi_1) \\ \pi \models \phi_1\ \mathtt{U}\ \phi_2 &\;\Leftrightarrow\; \exists i \in \mathbb{N}.\, (\pi(i) \models \phi_2 \wedge \forall j < i.\, \pi(j) \models \phi_1) \end{aligned}$$

For a reward structure r, reward formula and path $\pi \in IPaths_{\mathsf{G}^{\mathcal{C}},s}$:

$$\begin{array}{rcl}
rew(r, \mathtt{I}^{=k})(\pi) & = & r_S(\pi(k)) \\
rew(r, \mathtt{C}^{\leqslant k})(\pi) & = & \sum_{i=0}^{k-1} \big(r_A(\pi(i), \pi[i]) + r_S(\pi(i))\big) \\
rew(r, \mathtt{C})(\pi) & = & \sum_{i=0}^{\infty} \big(r_A(\pi(i), \pi[i]) + r_S(\pi(i))\big) \\
rew(r, \mathtt{F}^{c}\ \phi)(\pi) & = & \begin{cases} \sum_{i=0}^{\infty} \big(r_A(\pi(i), \pi[i]) + r_S(\pi(i))\big) & \text{if } \forall j \in \mathbb{N}.\, \pi(j) \not\models \phi \\ \sum_{i=0}^{k_\phi} \big(r_A(\pi(i), \pi[i]) + r_S(\pi(i))\big) & \text{otherwise} \end{cases} \\
rew(r, \mathtt{F}\ \phi)(\pi) & = & \begin{cases} \infty & \text{if } \forall j \in \mathbb{N}.\, \pi(j) \not\models \phi \\ \sum_{i=0}^{k_\phi} \big(r_A(\pi(i), \pi[i]) + r_S(\pi(i))\big) & \text{otherwise} \end{cases}
\end{array}$$

and $k_\phi = \min\{k-1 \mid \pi(k) \models \phi\}$.

As CSGs are *determined* (see Definition 4.3.5) with respect to the zero-sum properties we consider [155], for any CSG G, coalitions C and $N\backslash C$, state s, path formula ψ, reward structure r and reward formula ρ, the values $val_{\mathsf{G}^{\mathcal{C}}}(s, X^{\psi})$ and $val_{\mathsf{G}^{\mathcal{C}}}(s, X^{r,\rho})$ for either coalition of the game $\mathsf{G}^{\mathcal{C}}$ in state s with respect to the objectives X^{ψ} and $X^{r,\rho}$ are well defined. Moreover, by abuse of notation, we refer to $val_{\mathsf{G}^{C}}(s, X)$ and $val_{\mathsf{G}^{N\backslash C}}(s, X)$ as the value of the game $\mathsf{G}^{\mathcal{C}}$ with respect to objective X for coalitions C and $N\backslash C$, respectively. We can represent negated path formulae by inverting the probability threshold, e.g.: $\langle\!\langle C \rangle\!\rangle \mathtt{P}_{\geqslant q}[\,\neg\psi\,] \equiv \langle\!\langle C \rangle\!\rangle \mathtt{P}_{\leqslant 1-q}[\,\psi\,]$, notably allowing the 'globally' operator to be specified: $\mathtt{G}\ \phi \equiv \neg(\mathtt{F}\ \neg\phi)$. As for other probabilistic temporal logics, it is useful to consider *numerical* state formulae. In the case of zero-sum state formulae these take the form $\langle\!\langle C \rangle\!\rangle \mathtt{P}_{\min=?}[\,\psi\,]$, $\langle\!\langle C \rangle\!\rangle \mathtt{P}_{\max=?}[\,\psi\,]$, $\langle\!\langle C \rangle\!\rangle \mathtt{R}^{r}_{\min=?}[\,\rho\,]$ and $\langle\!\langle C \rangle\!\rangle \mathtt{R}^{r}_{\max=?}[\,\rho\,]$. For example, for state s we have:

$$\begin{array}{rcl}
\langle\!\langle C \rangle\!\rangle \mathtt{P}_{\min=?}[\,\psi\,] & \stackrel{\text{def}}{=} & \inf_{\sigma_1 \in \Sigma^1_{\mathsf{G}^C}} \sup_{\sigma_2 \in \Sigma^2_{\mathsf{G}^C}} \mathbb{E}^{\sigma_1,\sigma_2}_{\mathsf{G}^C,s}(X^{\psi}) \\
\langle\!\langle C \rangle\!\rangle \mathtt{P}_{\max=?}[\,\psi\,] & \stackrel{\text{def}}{=} & \sup_{\sigma_1 \in \Sigma^1_{\mathsf{G}^C}} \inf_{\sigma_2 \in \Sigma^2_{\mathsf{G}^C}} \mathbb{E}^{\sigma_1,\sigma_2}_{\mathsf{G}^C,s}(X^{\psi})\,.
\end{array}$$

The above determinacy result also yields the following equivalences:

$$\langle\!\langle C \rangle\!\rangle \mathtt{P}_{\max=?}[\,\psi\,] \equiv \langle\!\langle N\backslash C \rangle\!\rangle \mathtt{P}_{\min=?}[\,\psi\,] \;\text{ and }\; \langle\!\langle C \rangle\!\rangle \mathtt{R}^{r}_{\max=?}[\,\rho\,] \equiv \langle\!\langle N\backslash C \rangle\!\rangle \mathtt{R}^{r}_{\min=?}[\,\rho\,]\,.$$

If ϕ is a zero-sum formula of the form $\langle\!\langle C \rangle\!\rangle \mathtt{P}_{\sim q}[\,\psi\,]$ or $\langle\!\langle C \rangle\!\rangle \mathtt{R}^{r}_{\sim x}[\,\rho\,]$, this reduces to computing values for a two-player CSG $\mathsf{G}^{\mathcal{C}}$ (for either C or $N\backslash C$) with respect to X^{ψ} or $X^{r,\rho}$. In particular, for $\sim \in \{\geqslant, >\}$ and $s \in S$ we have:

$$\begin{array}{rcl}
s \models \langle\!\langle C \rangle\!\rangle \mathtt{P}_{\sim q}[\,\psi\,] & \Leftrightarrow & val_{\mathsf{G}^C}(s, X^{\psi}) \sim q \\
s \models \langle\!\langle C \rangle\!\rangle \mathtt{R}^{r}_{\sim x}[\,\rho\,] & \Leftrightarrow & val_{\mathsf{G}^C}(s, X^{r,\rho}) \sim x\,.
\end{array}$$

computing the value for C and, since CSGs are determined for the zero-sum properties we consider, for $\sim \in \{<, \leqslant\}$ we have:

$$s \models \langle\langle C \rangle\rangle \mathtt{P}_{\sim q}[\,\psi\,] \quad \Leftrightarrow \quad val_{\mathsf{G}^{N\setminus C}}(s, X^{\psi}) \sim q$$
$$s \models \langle\langle C \rangle\rangle \mathtt{R}^{r}_{\sim x}[\,\rho\,] \quad \Leftrightarrow \quad val_{\mathsf{G}^{N\setminus C}}(s, X^{r,\rho}) \sim x\,.$$

computing the value for $N\setminus C$. Without loss of generality, for such formulae we focus on computing $val_{\mathsf{G}^C}(s, X^{\psi})$ and $val_{\mathsf{G}^C}(s, X^{r,\rho})$ and, to simplify the presentation, we denote these values by $\mathtt{V}_{\mathsf{G}^C}(s, \psi)$ and $\mathtt{V}_{\mathsf{G}^C}(s, r, \rho)$ respectively.

4.5 Model Checking Zero-sum Properties

We now present an algorithm for model checking the extended rPATL logic, presented in the previous section, on a CSG G. Since rPATL is a branching-time logic, the basic algorithm works by recursively computing the set $Sat(\phi)$ of states satisfying formula ϕ over the structure of ϕ, as is done for rPATL on TSGs [53]. For the remainder of this section we assume that the available actions of players 1 and 2 of the (two-player) CSG $\mathsf{G}^{\mathcal{C}}$ in state s are $\{a_1, \ldots, a_l\}$ and $\{b_1, \ldots, b_m\}$, respectively. We use value iteration to compute/approximate the values of finite/infinite horizon zero-sum state formulae. Pseudocode for the model checking procedures is presented in Section B.2 of Appendix B. We require the following assumptions on CSGs for zero-sum formulae:

Assumption 1. For total reward formulae we require that all negative rewards are associated with state-action pairs or states such that the states reach an absorbing state (a state where all rewards are zero and which cannot be left) with probability 1 under all strategy profiles.

Assumption 2. For a zero-sum formula of the form $\langle\langle C \rangle\rangle \mathtt{R}^{r}_{\sim x}[\,\mathtt{F}\ \phi\,]$, from any state s where $r_S(s) < 0$ or $r_A(s, a) < 0$ for some action a, under all profiles of G, with probability 1 we reach either a state satisfying ϕ or a state where all rewards are zero and which cannot be left with probability 1 under all profiles.

Without this assumptions, the values computed during value iteration can oscillate, and therefore fail to converge (see Appendices C.1 and C.2). This restriction is not applied in the existing rPATL model checking algorithms for TSGs [53] since that work assumes that all rewards are non-negative.

4.5.1 Computing Values of Zero-sum Finite-horizon Formulae

Finite-horizon properties are defined over a bounded number of steps: the next or bounded until operators for probabilistic formulae, and the instantaneous or bounded cumulative reward operators. Computation of the values $\mathtt{V}_{\mathsf{G}^{\mathcal{C}}}(s,\psi)$ or $\mathtt{V}_{\mathsf{G}^{\mathcal{C}}}(s,r,\rho)$ for these is done recursively, based on the step bound, using backward induction and solving matrix games in each state at each iteration.

Below we show how the matrix games are constructed. The actions of each matrix game correspond to the actions available in that state; the utilities are constructed from the transition probabilities $\delta^{\mathcal{C}}$ of the game $\mathsf{G}^{\mathcal{C}}$, the reward structure r (in the case of reward formulae) and the values already computed recursively for successor states.

Next. This is the simplest operator, over just one step, and so in fact requires no recursion, just a solution of a matrix game for each state. If $\psi = \mathtt{X}\,\phi$, then for any state s we have that $\mathtt{V}_{\mathsf{G}^{\mathcal{C}}}(s,\psi) = val(\mathsf{Z})$ where $\mathsf{Z} \in \mathbb{Q}^{l\times m}$ is the matrix game with:

$$z_{i,j} = \textstyle\sum_{s'\in Sat(\phi)} \delta^{\mathcal{C}}(s,(a_i,b_j))(s')\,.$$

Bounded Until. If $\psi = \phi_1\ \mathtt{U}^{\leqslant k}\ \phi_2$, we compute the values for the path formulae $\psi_n = \phi_1 \mathtt{U}^n\ \phi_2$ for $0 \leqslant n \leqslant k$ recursively. For any state s:

$$\mathtt{V}_{\mathsf{G}^{\mathcal{C}}}(s,\psi_n) = \begin{cases} 1 & \text{if } s \in Sat(\phi_2) \\ 0 & \text{else if } s \notin Sat(\phi_1) \\ 0 & \text{else if } n{=}0 \\ val(\mathsf{Z}) & \text{otherwise} \end{cases}$$

where $val(\mathsf{Z})$ equals the value of the matrix game $\mathsf{Z} \in \mathbb{Q}^{l\times m}$ with:

$$z_{i,j} = \textstyle\sum_{s'\in S} \delta^{\mathcal{C}}(s,(a_i,b_j))(s') \cdot v_{n-1}^{s'}$$

and $v_{n-1}^{s'} = \mathtt{V}_{\mathsf{G}^{\mathcal{C}}}(s',\psi_{n-1})$ for all $s' \in S$.

Instantaneous Rewards. If $\rho = \mathtt{I}^{=k}$, then for the reward structure r we compute the values for the reward formulae $\rho_n = \mathtt{I}^{=n}$ for $0 \leqslant n \leqslant k$ recursively. For any state s:

$$\mathtt{V}_{\mathsf{G}^{\mathcal{C}}}(s,r,\rho_n) = \begin{cases} r_S(s) & \text{if } n = 0 \\ val(\mathsf{Z}) & \text{otherwise} \end{cases}$$

where $val(\mathsf{Z})$ equals the value of the matrix game $\mathsf{Z} \in \mathbb{Q}^{l\times m}$ with:

$$z_{i,j} = \textstyle\sum_{s'\in S} \delta^{\mathcal{C}}(s,(a_i,b_j))(s') \cdot v_{n-1}^{s'}$$

and $v_{n-1}^{s'} = \mathtt{V}_{\mathsf{G}^c}(s', r, \rho_{n-1})$ for all $s' \in S$.

Bounded Cumulative Rewards. If $\rho = \mathtt{C}^{\leqslant k}$, then for the reward structure r we compute the values for the reward formulae $\rho_n = \mathtt{C}^{\leqslant n}$ for $0 \leqslant n \leqslant k$ recursively. For any state s:

$$\mathtt{V}_{\mathsf{G}^c}(s, r, \rho_n) = \begin{cases} 0 & \text{if } n = 0 \\ val(\mathsf{Z}) & \text{otherwise} \end{cases}$$

where $val(\mathsf{Z})$ equals the value of the matrix game $\mathsf{Z} \in \mathbb{Q}^{l \times m}$ with:

$$z_{i,j} = r_A(s, (a_i, b_j)) + r_S(s) + \textstyle\sum_{s' \in S} \delta^{\mathcal{C}}(s, (a_i, b_j))(s') \cdot v_{n-1}^{s'}$$

and $v_{n-1}^{s'} = \mathtt{V}_{\mathsf{G}^c}(s', r, \rho_{n-1})$ for all $s' \in S$.

4.5.2 Computing Values of Zero-sum Infinite-horizon Formulae

We now discuss how to compute the values $\mathtt{V}_{\mathsf{G}^c}(s, \psi)$ and $\mathtt{V}_{\mathsf{G}^c}(s, r, \rho)$ for infinite-horizon properties, i.e., when the path formula ψ is an until operator, or for the expected reachability variant of the reward formulae ρ. In both cases, we approximate these values using value iteration, adopting a similar recursive computation to the finite-horizon cases above, solving matrix games in each state and at each iteration, which converges in the limit to the desired values.

Following the approach typically taken in probabilistic model checking tools to implement value iteration, we estimate convergence of the iterative computation by checking the maximum relative difference between successive iterations. However, it is known [101] that, even for simpler probabilistic models such as MDPs, this convergence criterion cannot be used to guarantee that the final computed values are accurate to within a specified error bound. Alternative approaches that resolve this by computing lower and upper bounds for each state have been proposed for MDPs (e.g. [101, 32]) and extended to both single- and multi-objective solution of TSGs [127, 7]; extensions could be investigated for CSGs. Another possibility is to use *policy iteration* (see, e.g., [44]).

Until. If $\psi = \phi_1 \ \mathtt{U} \ \phi_2$, the probability values can be approximated through value iteration using the fact that $\langle \mathtt{V}_{\mathsf{G}^c}(s, \phi_1 \ \mathtt{U}^{\leqslant k} \ \phi_2) \rangle_{k \in \mathbb{N}}$ is a non-decreasing sequence converging to $\mathtt{V}_{\mathsf{G}^c}(s, \phi_1 \ \mathtt{U} \ \phi_2)$. We compute $\mathtt{V}_{\mathsf{G}^c}(s, \phi_1 \ \mathtt{U}^{\leqslant k} \ \phi_2)$ for increasingly large k and estimate convergence as described above, based on the difference between values in successive iterations. However, we can potentially speed up convergence by first precomputing the set of states S_0^{ψ} for which the value of the zero-sum objective X^{ψ} is 0 and the

set of states S_1^ψ for which the value is 1 using standard graph algorithms [74]. We can then apply value iteration to approximate $\mathtt{V}_{\mathsf{G}^\mathcal{C}}(s, \phi_1 \ \mathtt{U} \ \phi_2) = \lim_{k\to\infty} \mathtt{V}_{\mathsf{G}^\mathcal{C}}(s, \phi_1 \ \mathtt{U} \ \phi_2, k)$ where:

$$\mathtt{V}_{\mathsf{G}^\mathcal{C}}(s, \phi_1 \ \mathtt{U} \ \phi_2, n) = \begin{cases} 1 & \text{if } s \in S_1^\psi \\ 0 & \text{else if } s \in S_0^\psi \\ 0 & \text{else if } n = 0 \\ val(\mathsf{Z}) & \text{otherwise} \end{cases}$$

where $val(\mathsf{Z})$ equals the value of the matrix game $\mathsf{Z} \in \mathbb{Q}^{l\times m}$ with:

$$z_{i,j} = \sum_{s'\in S} \delta^\mathcal{C}(s, (a_i, b_j))(s') \cdot v_{n-1}^{s'}$$

and $v_{n-1}^{s'} = \mathtt{V}_{\mathsf{G}^\mathcal{C}}(s', \phi_1 \ \mathtt{U} \ \phi_2, n-1)$ for all $s' \in S$.

Expected Total Rewards. If $\rho = \mathtt{C}$, we first find the states for which the expected reward values are infinite. Similarly to [53], we can use the qualitative algorithms of [74] to find these states as they can also be defined as those states for which the coalition C can ensure the probability of reaching states/performing actions with positive reward infinitely often is greater than 0. After removing these states from $\mathsf{G}^\mathcal{C}$, the remaining values for the reward structure r can be computed as the limit of the (non-decreasing sequence of) bounded cumulative reward values:

$$\mathtt{V}_{\mathsf{G}^\mathcal{C}}(s, r, \mathtt{C}) = \lim_{k\to\infty} \mathtt{V}_{\mathsf{G}^\mathcal{C}}(s, r, \mathtt{C}^{\leqslant k}) \,.$$

Expected Reachability. Here we have two reachability reward operators to consider. If $\rho = \mathtt{F}^c \ \phi$, then we first make all states of $\mathsf{G}^\mathcal{C}$ satisfying ϕ absorbing. Next, as in the case above, we find the states of $\mathsf{G}^\mathcal{C}$ for which the expected reward values are infinite and remove them from the game. The values for the remaining states can then be computed through value iteration where $\mathtt{V}_{\mathsf{G}^\mathcal{C}}(s, r, \rho) = \lim_{k\to\infty} \mathtt{V}_{\mathsf{G}^\mathcal{C}}(s, r, \rho_k)$ where:

$$\mathtt{V}_{\mathsf{G}^\mathcal{C}}(s, r, \rho_n) = \begin{cases} 0 & \text{if } s \in Sat(\phi) \\ val(\mathsf{Z}) & \text{otherwise} \end{cases}$$

where $val(\mathsf{Z})$ equals the value of the matrix game $\mathsf{Z} \in \mathbb{Q}^{l\times m}$ with:

$$z_{i,j} = r_A(s, (a_i, b_j)) + r_S(s) + \sum_{s'\in S} \delta^\mathcal{C}(s, (a_i, b_j))(s') \cdot v_{n-1}^{s'}$$

and $v_{n-1}^{s'} = \mathtt{V}_{\mathsf{G}^\mathcal{C}}(s', r, \rho_{n-1})$ for all $s' \in S$.

If $\rho = \mathtt{F} \ \phi$ and the reward structure is r, then we first make all states of $\mathsf{G}^\mathcal{C}$ satisfying ϕ absorbing, i.e., we remove all outgoing transitions from such states. Second, we find the set of states S_∞^ρ for which the reward is infinite; as in [53], this involves finding the

set of states satisfying the formula $\langle\langle C\rangle\rangle \mathtt{P}_{<1}[\,\mathtt{F}\ \phi\,]$ and we can use the graph algorithms of [74] to find these states. Again following [53], to deal with zero-reward cycles we need to use value iteration to compute a greatest fixed point. This involves first computing upper bounds on the actual values, by changing all zero reward values to some value $\gamma > 0$ to construct the reward structure $r_\gamma = (r_A^\gamma, r_A^\gamma)$ and then applying value iteration to approximate $\mathtt{V}^c_{\mathsf{G}}(s, r_\gamma, \rho) = \lim_{k\to\infty} \mathtt{V}^c_{\mathsf{G}}(s, r_\gamma, \rho_k)$ where:

$$\mathtt{V}^c_{\mathsf{G}}(s, r_\gamma, \rho_n) = \begin{cases} 0 & \text{if } s \in Sat(\phi) \\ \infty & \text{if } s \in S^\rho_\infty \\ val(\mathsf{Z}) & \text{otherwise} \end{cases}$$

where $val(\mathsf{Z})$ equals the value of the matrix game $\mathsf{Z} \in \mathbb{Q}^{l\times m}$ with:

$$z_{i,j} = r_A^\gamma(s,(a_i,b_j)) + r_S^\gamma(s) + \textstyle\sum_{s'\in S} \delta^{\mathcal{C}}(s,(a_i,b_j))(s') \cdot v_{n-1}^{s'}$$

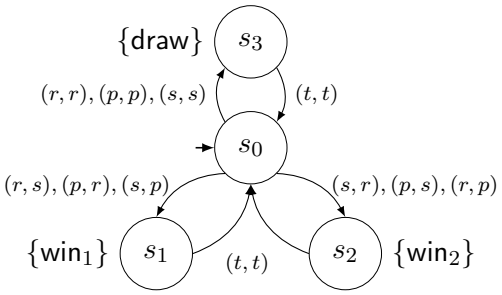
and $v_{n-1}^{s'} = \mathtt{V}^c_{\mathsf{G}}(s', r_\gamma, \rho_{n-1})$ for all $s' \in S$. Finally, using these upper bounds as the initial values we again perform value iteration as above, except now using the original reward structure r, i.e., to approximate $\mathtt{V}^c_{\mathsf{G}}(s, r, \rho) = \lim_{k\to\infty} \mathtt{V}^c_{\mathsf{G}}(s, r, \rho_k)$. The choice of γ can influence value iteration computations in opposing ways: increasing γ can speed up convergence when computing over-approximations, while potentially slowing it down when computing the actual values.

In Example 13, we illustrate how the model checking algorithm works for a bounded probabilistic until formula (and the corresponding unbounded variant by considering the limit as the bound grows to infinity). The example shows the construction of the local matrix games and how computation combines local optimisation (by solving LPs at the state level) and value iteration.

Example 13. Consider the problem of computing the value for the zero-sum property $\phi = \langle\langle p_1\rangle\rangle \mathtt{P}_{\max=?}[\,\neg\mathsf{win}_2\ \mathtt{U}^{\leqslant k}\ \mathsf{win}_1\,]$ over the *rock-paper-scissors* game, where p_1 represents the row player. We focus on computing the value for the initial state s_0 for a given k. For all n, we have $v_n^{s_1} = 1$, $v_n^{s_2} = 0$ and $v_n^{s_3} = v_{n-1}^{s_0}$. The matrix game for state s_0 at iteration n can be written as in Figure 4.8 (right).

At each round, we have to solve the following LP for state s_0:
Maximise $v_n^{s_0}$ subject to:

$$\begin{aligned} v_n^{s_0} &\leqslant x_1 \cdot v_{n-1}^{s_0} + x_2 \\ v_n^{s_0} &\leqslant x_2 \cdot v_{n-1}^{s_0} + x_3 \\ v_n^{s_0} &\leqslant x_3 \cdot v_{n-1}^{s_0} + x_1 \\ x_1 + x_2 + x_3 &= 1 \end{aligned}$$



(a) Rock-paper-scissors CSG.

$$\mathsf{Z}^n = \begin{array}{c} \\ r \\ p \\ s \end{array} \begin{array}{c} \begin{array}{ccc} r & p & s \end{array} \\ \begin{pmatrix} v_{n-1}^{s_0} & 0 & 1 \\ 1 & v_{n-1}^{s_0} & 0 \\ 0 & 1 & v_{n-1}^{s_0} \end{pmatrix} \end{array}$$

(b) Matrix game for state s_0.

Figure 4.8: Rock-paper-scissors represented in extensive and normal forms (one round).

Solving the LP, we see that the optimal strategy computed for player 1 in state s_0 is always the same and consists of playing r, p or s with equal probability, i.e, $\sigma_1 = (\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$. Furthermore, $v_{n+1}^{s_0} = \frac{1}{3} \cdot v_n^{s_0} + \frac{1}{3}$. Given that $v_0^{s_0} = 0$, we then have:

$$\mathsf{Z}^1 = \begin{array}{c} \\ r \\ p \\ s \end{array} \begin{array}{c} \begin{array}{ccc} r & p & s \end{array} \\ \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \end{array}, \mathsf{Z}^2 = \begin{array}{c} \\ r \\ p \\ s \end{array} \begin{array}{c} \begin{array}{ccc} r & p & s \end{array} \\ \begin{pmatrix} \frac{1}{3} & 0 & 1 \\ 1 & \frac{1}{3} & 0 \\ 0 & 1 & \frac{1}{3} \end{pmatrix} \end{array}, \mathsf{Z}^3 = \begin{array}{c} \\ r \\ p \\ s \end{array} \begin{array}{c} \begin{array}{ccc} r & p & s \end{array} \\ \begin{pmatrix} \frac{4}{9} & 0 & 1 \\ 1 & \frac{4}{9} & 0 \\ 0 & 1 & \frac{4}{9} \end{pmatrix} \end{array}, \ldots$$

and $v_1^{s_0} = \frac{1}{3}$, $v_2^{s_0} = \frac{4}{9}$, $v_3^{s_0} = \frac{13}{27}$, $v_4^{s_0} = \frac{40}{81}$, $v_5^{s_0} = \frac{121}{243}$ and so forth. In the limit, we have $\lim_{n\to\infty} v_n^{s_0} = \frac{1}{2}$.

■

4.5.3 Pre-computation Algorithms

We make use of the algorithms developed by de Alfaro and Henzinger in [74] as a pre-computation step to speed up the computation for unbounded probabilistic reachability or guarantee convergence in case of expected total rewards and expected reachability. In [74], the authors consider different winning conditions expressed by LTL formulas, with atomic propositions corresponding to subsets of states. Algorithms to compute the different sets of winning states are defined using μ-calculus notation, where for a given set A, νA and μA computes the *greatest* and the *least* fixed point over A. The authors define different winning modes $\lambda \in \{$*sure*, *almost*, *limit*, *bounded*, *positive*, *exist*$\}$ for formulae of the type $\langle\langle i \rangle\rangle_\lambda\ [\varphi]$, which denote the set of states from which player i can win in mode λ the game with winning condition ϕ. We make use of the algorithms defined for the winning modes *sure*, meaning that player i has a strategy to win regardless of what strategy the other player may choose to adopt, and *almost*, which establishes that player i wins *almost surely*,

that is, the player has a strategy to win with probability 1. For formulae of type $\langle\langle C\rangle\rangle \mathtt{P}_{\max=?}[\,\mathtt{F}\ \phi\,]$, we compute $\langle\langle N\backslash C\rangle\rangle_{sure}\ [\mathtt{G}\ \neg\phi]$ and $\langle\langle C\rangle\rangle_{almost}[\mathtt{F}\ \phi]$ to determine the states from which we are sure never to satisfy ϕ and those from which C has a strategy to reach states satisfying ϕ with probability 1. When checking for $\langle\langle C\rangle\rangle \mathtt{P}_{\min=?}[\,\mathtt{F}\ \phi\,]$, we compute $\langle\langle C\rangle\rangle_{sure}[\mathtt{G}\ \neg\phi]$ and $\langle\langle N\backslash C\rangle\rangle_{almost}[\mathtt{F}\ \phi]$. Formulae of type $\langle\langle C\rangle\rangle \mathtt{R}^r_{\max}[\,\mathtt{F}\ \phi\,]$ and $\langle\langle C\rangle\rangle \mathtt{R}^r_{\min}[\,\mathtt{F}\ \phi\,]$ ask to compute $S\backslash\langle\langle N\backslash C\rangle\rangle_{almost}[\mathtt{F}\ \phi]$ and $S\backslash\langle\langle C\rangle\rangle_{almost}[\mathtt{F}\ \phi]$, respectively. Finally, when model checking for $\langle\langle C\rangle\rangle \mathtt{R}^r_{\max}[\,\mathtt{F}^c\ \phi\,]$ and $\langle\langle C\rangle\rangle \mathtt{R}^r_{\max}[\,\mathtt{C}\,]$ or $\langle\langle C\rangle\rangle \mathtt{R}^r_{\min}[\,\mathtt{F}^c\ \phi\,]$ and $\langle\langle C\rangle\rangle \mathtt{R}^r_{\min}[\,\mathtt{C}\,]$, we need to first find $S\backslash\langle\langle N\backslash C\rangle\rangle_{almost}[\mathtt{F}\ \mathtt{G}\ \varphi]$ and $S\backslash\langle\langle C\rangle\rangle_{almost}[\mathtt{F}\ \mathtt{G}\ \varphi]$, respectively, where φ is the set of states that do not accumulate rewards. Pseudo-codes for our implementation of pre-computation algorithms can be found in Appendix B.

4.6 Strategy Synthesis

In addition to verifying formulae in our extension of rPATL, it is typically also very useful to perform *strategy synthesis*, i.e., to construct a witness to the satisfaction of a property. For each zero-sum formula $\langle\langle C\rangle\rangle \mathtt{P}_{\sim q}[\,\psi\,]$ or $\langle\langle C\rangle\rangle \mathtt{R}^r_{\sim x}[\,\rho\,]$ appearing as a sub-formula, this comprises optimal strategies for the players in coalition C (or, equivalently, for player 1 in the coalition game $\mathsf{G}^{\mathcal{C}}$) for the objective X^{ψ} or $X^{r,\rho}$.

All strategies synthesised are randomised for zero-sum formulae; this is in contrast to checking the equivalent properties against TSGs [53], where deterministic strategies are sufficient. For infinite-horizon objectives, we synthesise memoryless strategies, i.e., a distribution over actions for each state of the game. For finite-horizon objectives, strategies are finite-memory, with a separate distribution required for each state and each time step.

We can perform strategy synthesis by adapting the model checking algorithms described in the previous sections which compute the values of zero-sum objective. The type of strategy needed (deterministic or randomised; memoryless or finite-memory) depends on the types of objectives. As discussed previously (in Section 4.5.2), for infinite-horizon objectives our use of value iteration means we cannot guarantee that the values computed are within a particular error bound of the actual values; so, the same will be true of the optimal strategy that we synthesise for such a formula.

For both types of objectives, we synthesise the strategies whilst computing values using the approach presented in Section 4.5: from the matrix game solved for each state, we extract not just the value of the game, but also an optimal (randomised) strategy for player 1 of $\mathsf{G}^{\mathcal{C}}$ in that state. It is also possible to extract the optimal

strategy for player 2 in the state by solving the dual LP problem for the matrix game (see Section 4.2.1). For finite-horizon objectives, we retain the choices for all steps; for infinite-horizon objectives, just those from the final step of value iteration are needed as memoryless strategies are sufficient for the properties we consider and the final iteration contains the closest approximations according to the stopping criterion.

Example 14. Consider the problem of synthesising a strategy for the *rock-paper-scissors* game previously discussed in Example 13. In the example, we considered the property $\phi = \langle\langle p_1 \rangle\rangle \mathsf{P}_{\max=?}[\, \neg \mathsf{win}_2 \; \mathsf{U}^{\leqslant k} \; \mathsf{win}_1 \,]$, which asks to maximise the probability of player p_1 being the first to win a round within k rounds. We showed that the optimal strategy for player p_1 in s_0 is to play r, p or s with equal probability, and given that there are no choices in s_1, s_2 or s_3, the minimax strategy resulting from solving the LP for s_0 succinctly describes the strategy for the game irrespective of k. As strategy synthesis makes use of model checking algorithms, the same strategy would be computed for s_0 until the difference between the values $v_n^{s_0}$ and $v_{n-1}^{s_0}$ was smaller than ε according to a given convergence criterion. Unsurprisingly, given that the outcome of one round has no influence on the strategy a player should adopt in the next, although we use value iteration to synthesise the strategy, the error bound for the computed strategy is zero.

■

4.7 Correctness and Complexity

We conclude this section with a discussion of correctness and complexity. The overall (recursive) approach and the reduction to solution of a two-player game is essentially the same as for TSGs [53], and therefore the same correctness arguments apply. The correctness of value iteration for unbounded properties follows from [177] and for bounded properties from Definition 4.4.3 and the solution of matrix games (see Section 4.2.1).

Regarding complexity, due to the recursive nature of the algorithm, it is linear in the size of the formula ϕ, while in the worst case finding the optimal values of a 2-player CSG is in PSPACE [48]. In practice, we use value iteration, which solves an LP problem of size $|A|$ for each state at each iteration, with the number of iterations depending on the convergence criterion. Such problems can be solved using the simplex method, which is PSPACE-complete using Dantzig's pivot rule [70, 84], but performs well on average [196]. Karmarkar's algorithm [125] can also be used, which is PTIME. The efficiency in practice is reported in Chapters 7 and 8.

4.8 Summary

In this chapter, after reviewing classical notions and methods of game theory such as minimax strategies and solving matrix games through linear programming, we presented an extension of the branching-time temporal logic rPATL to express zero-sum properties over CSGs, detailing its syntax and semantics. We then introduced value iteration-based algorithms to approximate values for various types of objectives, which can be expressed as a combination of probabilistic or reward operators and different temporal operators.

We considered the problem of strategy synthesis for both quantitative and qualitative properties. For infinite-horizon properties, we can synthesise strategies that are memoryless by storing state-local minimax strategies, which may require randomisation. Finite-horizon properties require both memory and randomisation. We conclude this chapter with a discussion on correctness and complexity and review some methods and results on computing optimal values through linear programming, an essential step in model checking for zero-sum properties.

CHAPTER 5

Two-player Equilibria Properties

Contents

In this chapter, we extend our set of model checking techniques for zero-sum properties with algorithms for the computation of nonzero-sum equilibria properties. We start by giving a formal definition of *best response*, *subgame-perfect Nash equilibria*, *social welfare Nash equilibria* (SWNE) and *social cost Nash equilibria* (SCNE).

We also present an extension of rPATL to accommodate equilibria properties as well as the semantics and syntax of the new operators. The model checking algorithms for the new temporal and rewards operators follow next, including the model transformation necessary to verify properties with a mixture of bounded and unbounded operators. A discussion on strategy synthesis and the correctness and complexity of our approach conclude the chapter.

The methodology presented here sets itself apart from existing literature by enabling the computation/approximation of *optimal* equilibria over extensive concurrent stochastic games by computing local optimal equilibria at each state. Another

distinctive feature is the computation of equilibria properties for unbounded reachability properties without a discount factor, while maximising/minimising collective rewards/cost. Lastly, we encode the inequalities representing *regions of best response* through SMT assertions and find equilibria values through a mixture of explicit precomputation algorithms, backward induction, value iteration and to SMT-based procedures. The material presented in this chapter is based on [137, 140].

5.1 Nonzero-sum Games and Equilibria Strategies

We utilise the concept of *Nash equilibria* (NE) in order to investigate properties of nonzero-sum games. A strategy profile characterises a Nash equilibrium if it is not beneficial for any player to unilaterally change their strategy. In particular, we will use variants called *social welfare optimal* NE and *social cost optimal* NE. These variants are equilibria that maximise or minimise, respectively, the total or collective utility of the players, i.e. the sum of the individual player utilities. We start by defining auxiliary concepts of best and least responses, which are then used to formally define a Nash equilibrium.

Definition 5.1.1 (Best and Least Response)**.** For an NFG $\mathsf{N}=(N, A, u)$, a strategy profile $\sigma=(\sigma_1, \ldots, \sigma_n)$ and player i strategy σ_i', we define the sequence $\sigma_{-i} = (\sigma_1, \ldots, \sigma_{i-1}, \sigma_{i+1}, \ldots, \sigma_n)$ and profile $\sigma_{-i}[\sigma_i'] = (\sigma_1, \ldots, \sigma_{i-1}, \sigma_i', \sigma_{i+1}, \ldots, \sigma_n)$. For player i and strategy sequence σ_{-i}:

- a *best response* for player i to σ_{-i} is a strategy $\sigma_i^\star$ for player i such that $u_i(\sigma_{-i}[\sigma_i^\star]) \geqslant u_i(\sigma_{-i}[\sigma_i])$ for all strategies σ_i of player i;
- a *least response* for player i to σ_{-i} is a strategy $\sigma_i^\star$ for player i such that $u_i(\sigma_{-i}[\sigma_i^\star]) \leqslant u_i(\sigma_{-i}[\sigma_i])$ for all strategies σ_i of player i.

Definition 5.1.2 (Nash equilibrium)**.** For NFG $\mathsf{N}=(N, A, u)$, a strategy profile $\sigma^\star$ is a *Nash equilibrium* (NE) and $\langle u_i(\sigma^\star)\rangle_{i\in N}$ *NE values* if $\sigma_i^\star$ is a best response to $\sigma_{-i}^\star$ for all $i \in N$.

Definition 5.1.3 (Social Welfare NE)**.** For NFG $\mathsf{N}=(N, A, u)$, a strategy profile $\sigma^\star$ is a *social welfare optimal NE* (SWNE) and $\langle u_i(\sigma^\star)\rangle_{i\in N}$ the corresponding *SWNE values* if $\sigma^\star$ is an NE and $u_1(\sigma^\star)+\cdots+u_n(\sigma^\star) \geqslant u_1(\sigma)+\cdots+u_n(\sigma)$ for all NE σ of N.

Definition 5.1.4 (Social Cost NE)**.** For NFG $\mathsf{N}=(N, A, u)$, a strategy profile $\sigma^\star$ is a *social cost optimal NE* (SCNE) and $\langle u_i(\sigma^\star)\rangle_{i\in N}$ the corresponding *SCNE values* if $\sigma^\star$ is an NE of $\mathsf{N}' = (N, A, u')$, where $u' = -u$, and $u_1(\sigma^\star)+\cdots+u_n(\sigma^\star) \leqslant u_1(\sigma)+\cdots+u_n(\sigma)$ for all NE σ of N'.

The notion of SWNE is standard and corresponds to the case where utility values correspond to profits or rewards. We introduce the dual notion of SCNE in that case that utility values correspond to losses or costs. We could just negate all utilities and use the notion of SWNE. However, the notion of SCNE is more natural, particularly in the setting of probabilistic model checking where, for example, players may try to minimise the probability of reaching an error state or minimise the expected time to reach a goal state. The following lemma demonstrates the relationship between SWNE and SCNE.

Lemma 1. For NFG $\mathsf{N} = (N, A, u)$, a strategy profile $\sigma^\star$ of N is an NE of $\mathsf{N}^- = (N, A, -u)$ if and only if $\sigma^\star_i$ is a least response to $\sigma^\star_{-i}$ of player i in N for all $i \in N$. Furthermore, $\sigma^\star$ is a SWNE of N^- if and only if $\sigma^\star$ is a SCNE of N.

We remind the reader that by η_{a_i} we denote the pure strategy that selects action a_i with probability 1. Then, for a NFG G and NE $\sigma^\star$, player i is *indifferent* between actions $a_i, b_i \in A_i$ with respect to σ if $u_i(\sigma^\star_{-i}(\eta_a)) = u_i(\sigma^\star_{-i}(\eta_b)) = u_i(\sigma^\star)$. In such a situation, player i is also indifferent to any *affine* combination of a_i and b_i with respect to $\sigma^\star$, i.e. for any $p_1, p_2 \in (0, 1)$ such that $p_1+p_2 = 1$: $u_i(\sigma^\star_{-i}(p_1\cdot\eta_{a_i}+p_2\cdot\eta_{b_i})) = u_i(\sigma^\star)$. In case of indifference between two or more actions, the SWNE values are always captured by a pure strategy assignment for the indifferent player.

For nonzero-sum CSGs, with an objective X_i for each player i, we will use NE, which can be defined as for NFGs (see Definition 5.1.2). In line with the definition of zero-sum optimality (and because the model checking algorithms we will later introduce are based on backwards induction [181, 205]), we restrict our attention to *subgame-perfect* NE [184, 169], which are NE in *every state* of the CSG.

Definition 5.1.5 (Subgame-perfect NE)**.** For CSG G, a strategy profile $\sigma^\star$ is a *subgame-perfect Nash equilibrium* for objectives $\langle X_i\rangle_{i\in N}$ if and only if $\mathbb{E}^{\sigma^\star}_{\mathsf{G},s}(X_i) \geqslant \sup_{\sigma_i\in\Sigma_i} \mathbb{E}^{\sigma^\star_{-i}[\sigma_i]}_{\mathsf{G},s}(X_i)$ for all $i \in N$ and $s \in S$.

Furthermore, for infinite-horizon objectives, where the existence of NE is an open problem [29], we will in some cases use ε-NE, which do exist for any $\varepsilon > 0$ for all the properties we consider [52, 41].

Definition 5.1.6 (Subgame-perfect ε-NE)**.** For CSG G and $\varepsilon > 0$, a strategy profile $\sigma^\star$ is a *subgame-perfect ε-Nash equilibrium* for objectives $\langle X_i \rangle_{i \in N}$ if and only if $\mathbb{E}^{\sigma^\star}_{\mathsf{G},s}(X_i) \geqslant \sup_{\sigma_i \in \Sigma_i} \mathbb{E}^{\sigma^\star_{-i}[\sigma_i]}_{\mathsf{G},s}(X_i) - \varepsilon$ for all $i \in N$ and $s \in S$.

Finding Nash equilibria in bimatrix games is in the class of *linear complementarity* problems (LCPs). More precisely, a profile (σ_1, σ_2) is a Nash equilibrium of the bimatrix game $\mathsf{Z}_1, \mathsf{Z}_2 \in \mathbb{Q}^{l \times m}$ where $A_1{=}\{a_1, \ldots, a_l\}$, $A_2{=}\{b_1, \ldots, b_m\}$ if and only if there exists $u, v \in \mathbb{Q}$ such that, for the column vectors $x \in \mathbb{Q}^l$ and $y \in \mathbb{Q}^m$ where $x_i{=}\sigma_1(a_i)$ and $y_j{=}\sigma_2(b_j)$ for $1 \leqslant i \leqslant l$ and $1 \leqslant j \leqslant m$, we have:

$$\begin{aligned} x^T(\mathbf{1}u - \mathsf{Z}_1 y) &= 0 \\ y^T(\mathbf{1}v - \mathsf{Z}_2^T x) &= 0 \\ \mathbf{1}u - \mathsf{Z}_1 y &\geqslant \mathbf{0} \\ \mathbf{1}v - \mathsf{Z}_2^T x &\geqslant \mathbf{0} \end{aligned}$$

and $\mathbf{0}$ and $\mathbf{1}$ are vectors or matrices with all components 0 and 1, respectively.

As discussed in Chapter 2, there are several methods that can be used for computing Nash equilibria of bimatrix games. One of the most famous is the Lemke-Howson algorithm [145], which can be applied for finding NE and is based on the method of *labelled polytopes* [166]. Other well known methods include those based on *support enumeration* [174] and *regret minimisation* [179]. Similarly to the Lemke-Howson algorithm, we make use of labelled polytopes to characterise and find NE values, a method we elaborate in the next section.

5.1.1 Computing Values of Bimatrix Games

We first explain how we compute SWNE values in bimatrix games, then subgame-perfect SWNE values for finite-horizon objectives and lastly approximate subgame-perfect ε-SWNE values for infinite-horizon objectives. We also discuss how to synthesise SWNE profiles. Lemma 1 can be used to reduce the computation of SCNE profiles and values to those of SWNE profiles and values (or vice versa). This is achieved by negating all utilities in the NFG or bimatrix game, computing an SWNE profile and corresponding SWNE values, and then negating the SWNE values to obtain an SCNE profile and corresponding SCNE values for the original NFG or bimatrix game.

SWNE via Labelled Polytopes. Given a bimatrix game $\mathsf{Z}_1, \mathsf{Z}_2 \in \mathbb{Q}^{l \times m}$, we denote the sets of deterministic strategies of players 1 and 2 by $I{=}\{1, \ldots, l\}$ and $M{=}\{1, \ldots, m\}$ and define $J{=}\{l{+}1, \ldots, l{+}m\}$ by mapping $j \in M$ to $l{+}j \in J$. A *label*

is then defined as an element of $I \cup J$. The sets of strategies for players 1 and 2 can be represented by:

$$X = \{x \in \mathbb{Q}^l \mid \mathbf{1}x = 1 \ \wedge \ x \geqslant \mathbf{0}\} \quad \text{and} \quad Y = \{y \in \mathbb{Q}^m \mid \mathbf{1}y = 1 \ \wedge \ y \geqslant \mathbf{0}\}.$$

The strategy set Y is then divided into regions $Y(i)$ and $Y(j)$ (polytopes) for $i \in I$ and $j \in J$ such that $Y(i)$ contains strategies for which the deterministic strategy i of player 1 is a best response and $Y(j)$ contains strategies which choose action j with probability zero:

$$Y(i) = \{y \in Y \mid \forall k \in I.\, \mathsf{Z}_1(i,:)y \geqslant \mathsf{Z}_1(k,:)y\} \ \text{ and } \ Y(j) = \{y \in Y \mid y_{j-l} = 0\}$$

where $\mathsf{Z}_1(i,:)$ is the ith row vector of Z_1. A vector y is then said to have label k if $y \in Y(k)$, for $k \in I \cup J$. The strategy set X is divided analogously into regions $X(j)$ and $X(i)$ for $j \in J$ and $i \in I$ and a vector x has label k if $x \in X(k)$, for $k \in I \cup J$. A pair of vectors $(x, y) \in X \times Y$ is *completely labelled* if the union of the labels of x and y equals $I \cup J$. The Nash equilibria of the game equal the vector pairs that are completely labelled [145, 186].

We now show how to find equilibria values through SMT encoding of the sets of best response regions X and Y and subsequent search for assignments of strategy vector pairs that are completely labelled. The output are assignments to each vector component x_i, y_j that comprise the vector pairs. Each player's payoff can then be computed by simple vector-matrix multiplications. In the following, z^i_{jk} denotes the element in the jth row and kth column of matrix Z_i and by abuse of notation we let $X(r)$ and $Y(r)$ be a Boolean variable that represents whether the condition established by the inequality defining region $r \in [1, l+m]$ of strategy sets X and Y is satisfied, respectively. An equilibrium is then an assignment for vectors x and y such that:

$$\bigwedge_{r=1}^{l+m} (X(r) \vee Y(r))$$

and

$$\begin{aligned}
\forall r, s \in [1,l]: \quad & Y(r) \leftrightarrow \textstyle\bigwedge_{r \neq s} \left(\sum_{i=1}^{m} y_i z^1_{ri} \geqslant \sum_{j=1}^{m} y_j z^1_{sj} \right) \\
\forall r \in [l+1, l+m]: \quad & Y(r) \leftrightarrow (y_{r-l} = 0) \\
\forall r, s \in [l+1, l+m]: \quad & X(r) \leftrightarrow \textstyle\bigwedge_{r \neq s} \left(\sum_{i=1}^{l} z^2_{ir} \geqslant \sum_{j=1}^{l} x_j z^2_{js} \right) \\
\forall r \in [1,l]: \quad & X(r) \leftrightarrow (x_r = 0) \\
& \textstyle\sum_{i=1}^{l} x_i = 1 \\
& \textstyle\sum_{j=1}^{m} y_j = 1
\end{aligned}$$

Example 15. Let us consider again the problem discussed in Example 4 of Section 4.1.1. It referred to a coalitional instance of a stag hunt game described by the matrices:

$$\mathsf{Z}_1 = \begin{array}{c} \\ a_0 \\ a_1 \end{array} \begin{array}{c} \begin{array}{ccc} b_0 & b_1 & b_2 \end{array} \\ \begin{pmatrix} 2 & 2 & 2 \\ 0 & 4 & 6 \end{pmatrix} \end{array} \qquad \mathsf{Z}_2 = \begin{array}{c} \\ a_0 \\ a_1 \end{array} \begin{array}{c} \begin{array}{ccc} b_0 & b_1 & b_2 \end{array} \\ \begin{pmatrix} 4 & 2 & 0 \\ 4 & 6 & 9 \end{pmatrix} \end{array}$$

Following the labelled polytopes method, we would have a total of $l+m = 5$ labels for each player. The graphical representation of the regions of best response can be seen in Figure 5.1, and can be described by the following inequalities:

Y labels:
① : $Y(1) : 2y_0 + 2y_1 + 2y_2 \geqslant 4y_1 + 6y_2$
② : $Y(2) : 4y_1 + 6y_2 \geqslant 2y_0 + 2y_1 + 2y_2$
③ : $Y(3) : y_0 = 0$
④ : $Y(4) : y_1 = 0$ ✓
⑤ : $Y(5) : y_2 = 0$

X labels:
① : $X(1) : x_0 = 0$
② : $X(2) : x_1 = 0$
③ : $X(3) : 2x_0 \geqslant 2x_1 \wedge 4x_0 \geqslant 5x_1$
④ : $X(4) : 2x_1 \geqslant 2x_0 \wedge 2x_0 \geqslant 3x_1$ ✗
⑤ : $X(5) : 5x_1 \geqslant 4x_0 \wedge 3x_1 \geqslant 2x_0$

Figure 5.1 shows the strategy set Y on the right, where each subset $Y(r)$ is indicated by its label r drawn as a circled number. A similar representation is given for the strategy space X, which is shown on the left. The labels ① and ② are the pure strategies of the row player and are marked along the x_1 and x_0 axis, respectively. Similarly, the labels ③, ④ and ⑤ identify the regions in which b_0, b_1 and b_2 are respectively played with zero probability. Labels representing best response regions in mixed strategies for the row and column players are identified by the numbers ③ and ⑤ and ① and ②, respectively. We recall that, as both vectors x and y are strategy profiles, the individual sum of their elements has to be equal to 1.

Figure 5.1: Graphical representation of the regions of best response for the coalitional stag hunt game.

By examining the inequalities that describe each region, it is easy to see that the fourth X label ($X(4)$) cannot be satisfied for any values of x_0 and x_1. As all

bimatrix games have an equilibrium in mixed strategies, that means that any pair of equilibria strategies will have to satisfy the fourth Y label ($Y(4)$) to comply with the rule that, in order to be an equilibrium, a pair has to be completely labelled. This restriction simplifies the spatial representation of the regions of best response considerably, effectively reducing it from a tetrahedron, as we only have to focus on the y_0 and y_2 axes given that the value of y_1 has already been established to be zero. This restriction already gives us an insight about the problem itself, that is, there could be no equilibria in which only 1 hunter of the coalition represented by the column player decides to cooperate, as that would mean playing the action b_1 with positive probability. It is possible then to rewrite the inequalities assuming that ④ has to be satisfied as follows:

Y labels:	X labels:
① : $Y(1) : 2y_0 \geqslant 4y_2$	① : $X(1) : x_0 = 0$
② : $Y(2) : 4y_2 \geqslant 2y_0$	② : $X(2) : x_1 = 0$
③ : $Y(3) : y_0 = 0$	③ : $X(3) : 2x_0 \geqslant 2x_1 \wedge 4x_0 \geqslant 5x_1$
⑤ : $Y(5) : y_2 = 0$	⑤ : $X(5) : 5x_1 \geqslant 4x_0 \wedge 3x_1 \geqslant 2x_0$

Given that $x_0 + x_1 = 1$ and $y_0 + y_2 = 1$, it is easy to see that it is not possible to satisfy ① and ③, ② and ⑤, ① and ③ or ② and ⑤ at the same time. Likewise, it is possible to see that ① and ⑤ would always be satisfied together as would ② and ③, ① and ⑤ and ② and ③. We can then conclude the following pairs are completely labelled and therefore constitute an equilibrium:

$$(x, y) = \begin{cases} 1 : ((1,0),(1,0,0)) & \text{by satisfying ①, ②, ③, ④ and ⑤} \\ 2 : ((\frac{5}{9},\frac{4}{9}),(\frac{2}{3},0,\frac{1}{3})) & \text{by satisfying ①, ②, ③, ④ and ⑤} \\ 3 : ((0,1),(0,0,1)) & \text{by satisfying ①, ②, ③, ④ and ⑤} \end{cases}$$

Applying the strategy profiles of equilibria 1 or 2 yields a payoff of 2 for the row player and 4 for the column player. The strategies of equilibrium 3 yield a payoff of 6 for the row player and 9 for the column player and thus this is the only SWNE.

On the other hand, in order to compute SCNE, we first negate the entries in the matrix, which is equivalent to changing the direction of the inequalities that describe the regions of best response. Figure 5.2 shows the spatial representation of the regions which are described by the following sets of inequalities:

Y labels:	X labels:
① : $Y(1) : 2y_0 + 2y_1 + 2y_2 \leqslant 4y_1 + 6y_2$	① : $X(1) : x_0 = 0$
② : $Y(2) : 4y_1 + 6y_2 \leqslant 2y_0 + 2y_1 + 2y_2$	② : $X(2) : x_1 = 0$
③ : $Y(3) : y_0 = 0$	③ : $X(3) : 2x_0 \leqslant 2x_1 \wedge 4x_0 \leqslant 5x_1$
④ : $Y(4) : y_1 = 0$	④ : $X(4) : 2x_1 \leqslant 2x_0 \wedge 2x_0 \leqslant 3x_1$
⑤ : $Y(5) : y_2 = 0$	⑤ : $X(5) : 5x_1 \leqslant 4x_0 \wedge 3x_1 \leqslant 2x_0$

Differently from the case of finding regular NE, we cannot eliminate any label due to trivial unsatisfiability. We thus have a tetrahedron to represent the solution space for the strategies of the column player and the first two Y labels defined over its frontal face. Similarly to the previous case, when solving the set of inequalities we find three satisfying assignments for strategy pairs which are completely labelled:

$$(x, y) = \begin{cases} 1 : ((0,1),(1,0,0)) & \text{by satisfying } \textcircled{1}, \textcircled{2}, \textcircled{3}, \textcircled{4} \text{ and } \textcircled{5} \\ 2 : ((\frac{1}{2},\frac{1}{2}),(\frac{1}{2},\frac{1}{2},0)) & \text{by satisfying } \textcircled{1}, \textcircled{2}, \textcircled{3}, \textcircled{4} \text{ and } \textcircled{5} \\ 3 : ((1,0),(0,0,1)) & \text{by satisfying } \textcircled{1}, \textcircled{2}, \textcircled{3}, \textcircled{4} \text{ and } \textcircled{5} \end{cases}$$

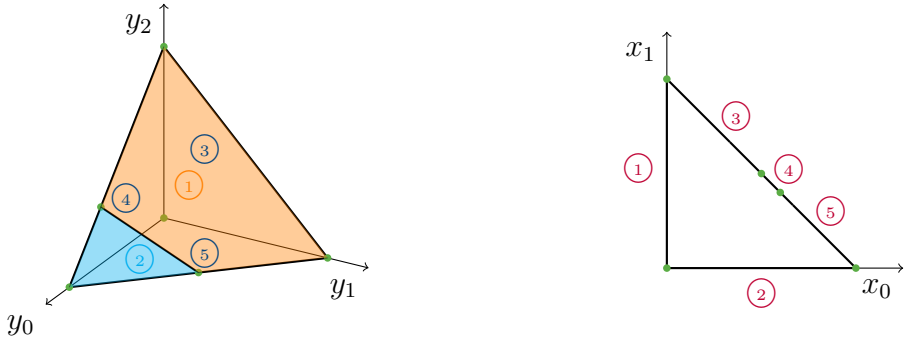


Figure 5.2: Graphical representation of the regions of best response for the negated coalitional stag hunt game.

Applying the strategy profile of equilibrium 1 yields a payoff of 0 for the row player and 4 for the column player. The strategies of equilibrium 2 yield a payoff of 2 for the row player and 4 for the column player. Finally, applying the strategy profile of the third equilibrium results on a payoff of 2 for the row player and 0 for the column player and thus this is the only SCNE.

■

5.2 Property Specification

We now extend the logic rPATL to allow the analysis of equilibria properties. Since we are limited to considering ε-SWNE for infinite-horizon properties, we assume some ε has been fixed in advance when considering such properties.

Definition 5.2.1 (Extended rPATL syntax for two-coalitional equilibria properties)**.**

The syntax of our extended version of rPATL is given by the grammar:

$$
\begin{array}{rcl}
\phi & \coloneqq & \mathtt{true} \mid \mathtt{a} \mid \neg\phi \mid \phi \wedge \phi \mid \langle\!\langle C \rangle\!\rangle \mathtt{P}_{\sim q}[\,\psi\,] \mid \langle\!\langle C \rangle\!\rangle \mathtt{R}^{r}_{\sim x}[\,\rho\,] \mid \langle\!\langle C{:}C' \rangle\!\rangle_{\mathrm{opt}\sim x}(\theta) \\
\theta & \coloneqq & \mathtt{P}[\,\psi\,]{+}\mathtt{P}[\,\psi\,] \mid \mathtt{R}^{r}[\,\rho\,]{+}\mathtt{R}^{r}[\,\rho\,] \\
\psi & \coloneqq & \mathtt{X}\,\phi \mid \phi\ \mathtt{U}^{\leqslant k}\ \phi \mid \phi\ \mathtt{U}\ \phi \\
\rho & \coloneqq & \mathtt{I}^{=k} \mid \mathtt{C}^{\leqslant k} \mid \mathtt{F}\ \phi
\end{array}
$$

where $\mathtt{a}$ is an atomic proposition, C and C' are coalitions of players such that $C'{=}N\backslash C$, $\mathrm{opt} \in \{\min, \max\}$, $\sim \in \{<, \leqslant, \geqslant, >\}$, $q \in [0,1]$, $x \in \mathbb{Q}$, r is a reward structure and $k \in \mathbb{N}$.

We extend rPATL with the ability to reason about *equilibria* through *Nash formulae* of the form $\langle\!\langle C{:}C' \rangle\!\rangle_{\mathrm{opt}\sim x}(\theta)$. In addition to the usual state (ϕ), path (ψ) and reward (ρ) formulae, we distinguish *non-zero sum* formulae (θ), which comprise a sum of probability or reward objectives. The formula $\langle\!\langle C{:}C' \rangle\!\rangle_{\max\sim x}(\theta)$ is satisfied if there exists a subgame-perfect SWNE strategy profile between coalitions C and $C'(=N\backslash C)$ under which the *sum* of the two objectives in θ is $\sim x$. Similarly, a formula $\langle\!\langle C{:}C' \rangle\!\rangle_{\min\sim x}(\theta)$ is satisfied if there exists a subgame-perfect SCNE strategy profile between coalitions C and $C'(=N\backslash C)$ under which the *sum* of the two objectives in θ is $\sim x$.

For probabilistic objectives ($\theta{=}\mathtt{P}[\,\psi^1\,]{+}\mathtt{P}[\,\psi^2\,]$), each ψ^i can be a *next* ($\mathtt{X}$), *bounded until* ($\mathtt{U}^{\leqslant k}$) or *until* ($\mathtt{U}$) operator, with the usual equivalences such as $\mathtt{F}\ \phi \equiv \mathtt{true}\ \mathtt{U}\ \phi$. For reward objectives ($\theta{=}\mathtt{R}^{r_1}[\,\rho^1\,]{+}\mathtt{R}^{r_2}[\,\rho^2\,]$), each ρ^i refers to the expected reward with respect to reward structure r_i: the instantaneous reward after k steps (*instantaneous rewards* $\mathtt{I}^{=k}$); the reward accumulated over k steps (*bounded cumulative rewards* $\mathtt{C}^{\leqslant k}$); or the reward accumulated until a state satisfying ϕ is reached (*expected reachability* $\mathtt{F}\ \phi$). As is common for probabilistic temporal logics, we allow numerical queries of the form $\langle\!\langle C{:}C' \rangle\!\rangle_{\max=?}[\theta]$ and $\langle\!\langle C{:}C' \rangle\!\rangle_{\min=?}[\theta]$, which return the approximate sum of SWNE and SCNE values respectively.

The rPATL syntax for nonzero-sum formulae presented in Definition 5.2.1 differs from that for zero-sum formulae presented in Definition 4.4 by dropping the reward operators for *expected reachability* ($\mathtt{F}^c$) and *total rewards* ($\mathtt{C}$) as currently no algorithms exist for these variants. In Example 16, we present a few properties to illustrate the use and applicability of rPATL nonzero-sum formulae. We highlight that the use of the term "collaborate" in this context should be understood as playing their equilibria strategies followed by full cooperation when either coalition satisfies its goal. In that regard, when referring to nonzero-sum or equilibria properties, we

assume a weaker notion of collaboration than one would have if the players in the game just synchronised their actions. In fact, we mean the players would act rationally but in a way to minimise or maximise the overall sum of the random variables associated to their objectives (consequently maximising social welfare or minimising social cost) and that, considering they would be following an equilibrium strategy, they would not have an incentive to unilaterally deviate.

Example 16. Recall the robot coordination problem from Example 8. The following are properties are examples of equilibria formulae one could express with rPATL:

- $\langle\!\langle robot_1{:}robot_2 \rangle\!\rangle_{\max \geqslant 2}(\mathtt{P}[\,\neg \mathsf{crash}\ \mathtt{U}\ \mathsf{goal}_1\,] + \mathtt{P}[\,\neg \mathsf{crash}\ \mathtt{U}\ \mathsf{goal}_2\,])$ expresses that the robots can collaborate so that both reach their goal with probability 1 without crashing;
- $\langle\!\langle robot_1{:}robot_2 \rangle\!\rangle_{\min=?}(\mathtt{R}^{r_{steps}}[\,\mathtt{F}\ \mathsf{goal}_1\,] + \mathtt{R}^{r_{steps}}[\,\mathtt{F}\ \mathsf{goal}_2\,])$ asks what is the sum of expected values when the robots collaborate and each wants to minimise their expected steps to reach their goal (r_{steps} is a reward structure that assigns 1 to all state action tuple pairs);
- $\langle\!\langle robot_1{:}robot_2 \rangle\!\rangle_{\max=?}(\mathtt{P}[\,\mathtt{F}\ \mathsf{goal}_1\,] + \mathtt{P}[\,\neg \mathsf{crash}\ \mathtt{U}^{\leqslant 10}\mathsf{goal}_2\,])$ asks what is the sum of probabilities when the robots collaborate to reach their individual goals with the additional condition the second has to reach its goal within 10 steps without crashing.

■

Our logic includes both *finite-horizon* ($\mathtt{X}$, $\mathtt{U}^{\leqslant k}$, $\mathtt{I}^{=k}$, $\mathtt{C}^{\leqslant k}$) and *infinite-horizon* ($\mathtt{U}$, $\mathtt{F}$) temporal operators. For the latter, the existence of SWNE or SCNE profiles is an open problem [29], but we can check for ε-SWNE or ε-SCNE profiles for any ε [52, 41]. Hence, we define the semantics of the logic in the context of a particular ε.

Definition 5.2.2 (Extended rPATL semantics for two-coalitional equilibria properties)**.** The satisfaction relation $\models$ of our rPATL extension is defined inductively on the structure of the formula. The propositional logic fragment ($\mathtt{true}$, a, $\neg$, $\wedge$) is defined in the usual way. The zero-sum formulae $\langle\!\langle C \rangle\!\rangle \mathtt{P}_{\sim q}[\,\psi\,]$ and $\langle\!\langle C \rangle\!\rangle \mathtt{R}^{r}_{\sim x}[\,\rho\,]$ are defined as in Chapter 4. For temporal operators and a state $s \in S$ in CSG $\mathsf{G}^{\mathcal{C}}$, we have:

$$s \models \langle\!\langle C{:}C' \rangle\!\rangle_{\mathrm{opt} \sim x}(\theta) \quad \Leftrightarrow \quad \exists \sigma_1^\star \in \Sigma^1, \sigma_2^\star \in \Sigma^2.\ \left(\mathbb{E}^{\sigma_1^\star, \sigma_2^\star}_{\mathsf{G}^C, s}(X_1^\theta) + \mathbb{E}^{\sigma_1^\star, \sigma_2^\star}_{\mathsf{G}^C, s}(X_2^\theta) \right) \sim x$$

where $(\sigma_1^\star, \sigma_2^\star)$ is a subgame-perfect ε-SWNE if $\mathrm{opt} = \max$ and a subgame-perfect ε-SCNE if $\mathrm{opt} = \min$ for the objectives (X_1^θ, X_2^θ) in $\mathsf{G}^\mathcal{C}$ where, for $1 \leqslant i \leqslant 2$ and $\pi \in IPaths_{\mathsf{G}^\mathcal{C},s}^{\sigma_1,\sigma_2}$:

$$\begin{aligned} X_i^{\mathtt{P}[\,\psi^1\,]+\mathtt{P}[\,\psi^2\,]}(\pi) &= 1 \text{ if } \pi \models \psi^i \text{ and 0 otherwise} \\ X_i^{\mathtt{R}^{r_1}[\,\rho^1\,]+\mathtt{R}^{r_2}[\,\rho^2\,]}(\pi) &= rew(r_i, \rho^i)(\pi) \end{aligned}$$

and $k_\phi = \min\{k-1 \mid \pi(k) \models \phi\}$.

As in the case of zero-sum properties, formulae we can represent negated path formulae by inverting the probability threshold, e.g.: $\langle\!\langle C{:}C' \rangle\!\rangle_{\max \geqslant q}(\mathtt{P}[\,\psi_1\,]+\mathtt{P}[\,\psi_2\,]) \equiv \langle\!\langle C{:}C' \rangle\!\rangle_{\min \leqslant 2-q}(\mathtt{P}[\,\neg\psi_1\,]+\mathtt{P}[\,\neg\psi_2\,])$, notably allowing the 'globally' operator to be specified for probabilistic formulae: $\mathtt{G}\ \phi \equiv \neg(\mathtt{F}\ \neg\phi)$. This theoretical result relies on Definition 5.2.2 and Lemma 2, which shows the equivalence for normal form games.

Lemma 2. For NFG $\mathsf{N} = (N, A, u)$, a profile $\sigma^\star$ of N is a SCNE of N if and only if $\sigma^\star$ is a SWNE of $\mathsf{N}^{1-} = (N, A, u^{1-})$ where $u_i^{1-}(a) = 1 - u_i(a)$ for all $i \in N$ and $a \in A$.

Proof. Consider any NFG $\mathsf{N} = (N, A, u)$ and profile $\sigma^\star$ of N. If $\sigma^\star$ is a SCNE of N, then we have:

- $\sigma^\star$ is an NE of $\mathsf{N}^- = (N, A, -u)$;
- $u_1(\sigma^\star)+\cdots+u_n(\sigma^\star) \leqslant u_1(\sigma)+\cdots+u_n(\sigma)$ for all NE σ of N^-.

Furthermore, if $\sigma^\star$ is a SWNE of N^{1-}, then we have:

- $\sigma^\star$ is an NE of $\mathsf{N}^{1-} = (N, A, u^{1-})$;
- $u_1^{1-}(\sigma^\star)+\cdots+u_n^{1-}(\sigma^\star) \geqslant u_1^{1-}(\sigma)+\cdots+u_n^{1-}(\sigma)$ for all NE σ of N^{1-}.

Now considering any strategy profiles σ and σ' and player $i \in N$:

$$\begin{aligned} u_i^{1-}(\sigma) \geqslant u_i^{1-}(\sigma') &\Leftrightarrow 1 - u_i(\sigma) \geqslant 1 - u_i(\sigma') && \text{by definition of } u_i^{1-} \\ &\Leftrightarrow -u_i(\sigma) \geqslant -u_i(\sigma') && \text{rearranging} \end{aligned}$$

and hence, using Definitions 5.1.1 and 5.1.2, we have that σ is an NE of N^- if and only if σ is an NE of N^{1-}. Furthermore for any strategy profiles σ and σ':

$$
\begin{aligned}
& u_1^{1-}(\sigma)+\cdots+u_n^{1-}(\sigma) \geqslant u_1^{1-}(\sigma')+\cdots+u_n^{1-}(\sigma') \\
\Leftrightarrow\ & (1-u_1(\sigma))+\cdots+(1-u_n(\sigma)) \geqslant (1-u_1(\sigma'))+\cdots+(1-u_n(\sigma')) \\
& \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{by definition of } u^{1-} \\
\Leftrightarrow\ & -(1-u_1(\sigma))+\cdots+(1-u_n(\sigma)) \leqslant -(1-u_1(\sigma')+\cdots+(1-u_n(\sigma')) \\
& \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{negating both sides} \\
\Leftrightarrow\ & u_1(\sigma)+\cdots+u_n(\sigma)-n \leqslant u_1(\sigma')+\cdots+u_n(\sigma')-n \qquad\qquad \text{rearranging} \\
\Leftrightarrow\ & u_1(\sigma)+\cdots+u_n(\sigma) \leqslant u_1(\sigma')+\cdots+u_n(\sigma') \qquad\qquad \text{rearranging}
\end{aligned}
$$

Using these results and Definitions 5.1.3 and 5.1.4 we have that $\sigma^\star$ is a SWNE of N^{1-} if and only if $\sigma^\star$ is a SCNE of N. □

5.3 Model Checking Two-player Nonzero-sum Properties

Computation of nonzero-sum properties is restricted to a class of CSGs that can be seen as a variant of *stopping games* [54], which was also the case for multi-objective TSGs. Compared to [54], we apply a weaker, objective-dependent assumption, which ensures that, under all profiles, with probability 1, eventually the outcome of each player's objective does not change by continuing. We formalise these conditions through the following assumptions.

Assumption 3. For nonzero-sum formulae, if $\mathtt{P}[\,\phi_1\ \mathtt{U}\ \phi_2\,]$ is a probabilistic objective, then $\mathit{Sat}(\neg\phi_1 \vee \phi_2)$ is reached with probability 1 from all states under all profiles of G.

Assumption 4. For nonzero-sum formulae, if $\mathtt{R}^r[\,\mathtt{F}\ \phi\,]$ is a reward objective, then $\mathit{Sat}(\phi)$ is reached with probability 1 from all states under all profiles of G.

Like for Assumption 2, without this restriction, algorithms for equilibria properties may not converge since values can oscillate (see Appendices C.3 and C.4). Notice that Assumption 2 is not required for nonzero-sum properties containing negative rewards since Assumption 4 is itself a stronger restriction.

In the next sections, we show how to compute subgame-perfect SWNE and SCNE values for the two objectives corresponding to a nonzero-sum formula. As for the

zero-sum case, the approach taken depends on whether the formula contains finite-horizon or infinite-horizon objectives. As rPATL also allows for a mixture of the two objectives, we now have three cases:

1. when both objectives are finite-horizon, we use backward induction [181, 205, 188] to compute (precise) subgame-perfect SWNE and SCNE values;

2. when both objectives are infinite-horizon, we use value iteration [177, 47] to approximate the values;

3. when there is a mix of the two types of objectives, we convert the problem to two infinite-horizon objectives on an augmented model.

The three cases are described separately in Sections 5.3.1, 5.3.2 and 5.3.5, respectively. In a similar style to the algorithms for zero-sum properties, in all cases the computation is an iterative process that analyses a two-player game for each state at each step. However, this now requires finding SWNE or SCNE values of a bimatrix game, rather than solving a matrix game as in the zero-sum case. We solve bimatrix games using the labelled polytopes approach presented in Section 5.1.1 (see also the more detailed discussion of its implementation in Section 7.2.2.2).

Another important aspect of our algorithms is that, for efficiency, if we reach a state where the value of one player's objective cannot change (e.g., the goal of that player is reached or can no longer be reached), then we switch to the simpler problem of solving an MDP to find the optimal value for the other player in that state. This is possible since the only SWNE profile in that state corresponds to maximising the objective of the other player. More precisely:

- the first player (whose objective cannot change) is *indifferent*, since its value will not be affected by the choices of either player;

- the second player cannot do better than the optimal value of its objective in the corresponding MDP where both players act to maximise that value;

- for any NE profile, the value of the first player is fixed and the value of the second is less than or equal to the optimal value of its objective in the MDP.

We use the notation $\mathsf{P}^{\max}_{\mathsf{G},s}(\psi)$ and $\mathsf{R}^{\max}_{\mathsf{G},s}(r,\rho)$ for the maximum probability of satisfying the path formula ψ and the maximum expected reward for the random variable $rew(r,\rho)$, respectively, when the players act to maximise the same objective in state

s. These values can be computed through standard MDP model checking [19, 73]. From rPATL semantics (Definition Definition 5.2.2), computing $Sat(\phi)$ reduces to the computation of subgame-perfect SWNE or SCNE values for the objectives (X_1^θ, X_2^θ) and a comparison of their sum to the threshold x. Again, to simplify the presentation, will use the notation $\mathtt{V}_{\mathsf{G}^\mathcal{C}}(s,\theta)$ and $\overline{\mathtt{V}}_{\mathsf{G}^\mathcal{C}}(s,\theta)$ for the SWNE and SCNE values of the objectives (X_1^θ, X_2^θ) in state s of $\mathsf{G}^\mathcal{C}$. For any state formula ϕ and state s we let $\eta_\phi(s)$ equal 1 if $s \in Sat(\phi)$ and 0 otherwise.

For the remainder of this section, we fix a CSG $\mathsf{G} = (N, S, \bar{S}, A, \Delta, \delta, AP, L)$ and coalition C of players and assume that the available actions of players 1 and 2 of the (two-player) CSG $\mathsf{G}^\mathcal{C}$ in a state s are $\{a_1, \ldots, a_l\}$ and $\{b_1, \ldots, b_m\}$, respectively. We also fix a value $\varepsilon > 0$ which, as discussed in Section 5.2, is needed to define the semantics of our logic, in particular for infinite-horizon objectives where we need to consider ε-SWNE and ε-SCNE profiles.

5.3.1 Computing SWNE Values of Finite-horizon Nonzero-sum Formulae

As for the zero-sum case, for a *finite-horizon* nonzero-sum formula θ, we compute the SWNE values $\mathtt{V}_{\mathsf{G}^\mathcal{C}}(s,\theta)$ for all states s of the coalitional game $\mathsf{G}^\mathcal{C}$ in a recursive fashion based on the step bound. We now solve bimatrix games at each step, which are defined in a similar manner to the matrix games for zero-sum properties: the actions of each bimatrix game correspond to the actions available in that state and the utilities are constructed from the transition probabilities $\delta^\mathcal{C}$ of the game $\mathsf{G}^\mathcal{C}$, the reward structure (in the case of reward formulae) and the values already computed recursively for successor states.

Below, we explain the computation for both types of finite-horizon probabilistic objectives (next and bounded until) and reward objectives (instantaneous and bounded cumulative), as well as combinations of each type.

Next. If $\theta = \mathtt{P}[\,\mathtt{X}\,\phi^1\,] + \mathtt{P}[\,\mathtt{X}\,\phi^2\,]$, then $\mathtt{V}_{\mathsf{G}^\mathcal{C}}(s,\theta)$ equals SWNE values of the bimatrix game $(\mathsf{Z}_1, \mathsf{Z}_2) \in \mathbb{Q}^{l \times m}$ where:

$$z^1_{i,j} = \textstyle\sum_{s' \in S} \delta^\mathcal{C}(s,(a_i,b_j))(s') \cdot \eta_{\phi^1}(s')$$
$$z^2_{i,j} = \textstyle\sum_{s' \in S} \delta^\mathcal{C}(s,(a_i,b_j))(s') \cdot \eta_{\phi^2}(s')\,.$$

Bounded Until. If $\theta = \mathtt{P}[\,\phi_1^1\ \mathtt{U}^{\leqslant k_1}\ \phi_2^1\,] + \mathtt{P}[\,\phi_1^2\ \mathtt{U}^{\leqslant k_2}\ \phi_2^2\,]$, we compute SWNE values for the objectives for the nonzero-sum formulae $\theta_{n+n_1,n+n_2} = \mathtt{P}[\,\phi_1^1\ \mathtt{U}^{\leqslant n+n_1}\ \phi_2^1\,] +$

$\mathtt{P}[\,\phi_1^2\ \mathtt{U}^{\leqslant n+n_2}\ \phi_2^2\,]$ for $0 \leqslant n \leqslant k$ recursively, where $k{=}\min\{k_1,k_2\}$, $n_1{=}k_1{-}k$ and $n_2 = k_2{-}k$. For any state s, if $n{=}0$, then:

$$\mathtt{V}_{\mathsf{G}^{\mathcal{C}}}(s,\theta_{n_1,n_2}) = \begin{cases} (\eta_{\phi_2^1}(s),\eta_{\phi_2^2}(s)) & \text{if } n_1{=}n_2{=}0 \\ (\eta_{\phi_2^1}(s),\mathtt{P}^{\max}_{\mathsf{G},s}(\phi_1^2\ \mathtt{U}^{\leqslant n_2}\ \phi_2^2)) & \text{else if } n_1{=}0 \\ (\mathtt{P}^{\max}_{\mathsf{G},s}(\phi_1^1\ \mathtt{U}^{\leqslant n_1}\ \phi_2^1),\eta_{\phi_2^2}(s)) & \text{otherwise.} \end{cases}$$

On the other hand, if $n > 0$, then:

$$\mathtt{V}_{\mathsf{G}^{\mathcal{C}}}(s,\theta_{n+n_1,n+n_2}) = \begin{cases} (1,1) & \text{if } s \in Sat(\phi_2^1) \cap Sat(\phi_2^2) \\ (1,\mathtt{P}^{\max}_{\mathsf{G},s}(\phi_1^2\ \mathtt{U}^{\leqslant n+n_2}\ \phi_2^2)) & \text{else if } s \in Sat(\phi_2^1) \\ (\mathtt{P}^{\max}_{\mathsf{G},s}(\phi_1^1\ \mathtt{U}^{\leqslant n+n_1}\ \phi_2^1),1) & \text{else if } s \in Sat(\phi_2^2) \\ (\mathtt{P}^{\max}_{\mathsf{G},s}(\phi_1^1\ \mathtt{U}^{\leqslant n+n_1}\ \phi_2^1),0) & \text{else if } s \in Sat(\phi_1^1) \setminus Sat(\phi_1^2) \\ (0,\mathtt{P}^{\max}_{\mathsf{G},s}(\phi_1^2\ \mathtt{U}^{\leqslant n+n_2}\ \phi_2^2)) & \text{else if } s \in Sat(\phi_1^2) \setminus Sat(\phi_1^1) \\ (0,0) & \text{else if } s \notin Sat(\phi_1^1) \cap Sat(\phi_1^2) \\ val(\mathsf{Z}_1,\mathsf{Z}_2) & \text{otherwise} \end{cases}$$

where $val(\mathsf{Z}_1,\mathsf{Z}_2)$ equals SWNE values of the bimatrix game $(\mathsf{Z}_1,\mathsf{Z}_2) \in \mathbb{Q}^{l\times m}$:

$$\begin{aligned} z^1_{i,j} &= \textstyle\sum_{s'\in S} \delta^{\mathcal{C}}(s,(a_i,b_j))(s') \cdot v^{s',1}_{(n-1)+n_1} \\ z^2_{i,j} &= \textstyle\sum_{s'\in S} \delta^{\mathcal{C}}(s,(a_i,b_j))(s') \cdot v^{s',2}_{(n-1)+n_2} \end{aligned}$$

and $(v^{s',1}_{(n-1)+n_1}, v^{s',2}_{(n-1)+n_2}) = \mathtt{V}_{\mathsf{G}^{\mathcal{C}}}(s',\theta_{(n-1)+n_1,(n-1)+n_2})$ for all $s' \in S$.

Next and Bounded Until. If $\theta = \mathtt{P}[\,\mathtt{X}\,\phi^1\,]{+}\mathtt{P}[\,\phi_1^2\ \mathtt{U}^{\leqslant k_2}\ \phi_2^2\,]$, then $\mathtt{V}_{\mathsf{G}^{\mathcal{C}}}(s,\theta)$ equals SWNE values of the bimatrix game $(\mathsf{Z}_1,\mathsf{Z}_2) \in \mathbb{Q}^{l\times m}$ where:

$$z^1_{i,j} = \textstyle\sum_{s'\in S} \delta^{\mathcal{C}}(s,(a_i,b_j))(s') \cdot \eta_{\phi^1}(s')$$

$$z^2_{i,j} = \begin{cases} 1 & \text{if } s \in Sat(\phi_2^2) \\ 0 & \text{else if } k_2 = 0 \\ \sum_{s'\in S} \delta^{\mathcal{C}}(s,(a_i,b_j))(s') \cdot \mathtt{P}^{\max}_{\mathsf{G},s}(\phi_1^2\ \mathtt{U}^{\leqslant k_2-1}\ \phi_2^2) & \text{else if } Sat(\phi_1^2) \\ 0 & \text{otherwise.} \end{cases}$$

In this case, since the value for objectives corresponding to next formulae cannot change after the first step, we can always switch to MDP verification after this step. The symmetric case is similar.

Instantaneous Rewards. If $\theta{=}\mathtt{R}^{r_1}[\,\mathtt{I}^{=k_1}\,]{+}\mathtt{R}^{r_2}[\,\mathtt{I}^{=k_2}\,]$, we compute SWNE values of the objectives for the nonzero-sum formulae $\theta_{n+n_1,n+n_2} = \mathtt{R}^{r_1}[\,\mathtt{I}^{=n+n_1}\,] + \mathtt{R}^{r_2}[\,\mathtt{I}^{=n+n_2}\,]$ for $0 \leqslant n \leqslant k$ recursively, where $k{=}\min\{k_1,k_2\}$, $n_1 = k_1{-}k$ and $n_2{=}k_2{-}k$. For any

state s, if $n{=}0$, then:

$$\mathtt{V}_{\mathsf{G}^{\mathcal{C}}}(s,\theta_{n_1,n_2}) = \begin{cases} (r_S^1(s), r_S^2(s)) & \text{if } n_1{=}n_2{=}0 \\ (r_S^1(s), \mathtt{R}^{\max}_{\mathsf{G},s}(r_2, \mathtt{I}^{=n_2})) & \text{else if } n_1{=}0 \\ (\mathtt{R}^{\max}_{\mathsf{G},s}(r_1, \mathtt{I}^{=n_1}), r_S^2(s)) & \text{otherwise} \end{cases}$$

On the other hand, if $n > 0$, then $\mathtt{V}_{\mathsf{G}^{\mathcal{C}}}(s,\theta_{n+n_1,n+n_2})$ equals SWNE values of the bimatrix game $(\mathsf{Z}_1,\mathsf{Z}_2) \in \mathbb{Q}^{l\times m}$ where:

$$\begin{aligned} z_{i,j}^1 &= \textstyle\sum_{s'\in S} \delta^{\mathcal{C}}(s,(a_i,b_j))(s') \cdot v_{(n-1)+n_1}^{s',1} \\ z_{i,j}^2 &= \textstyle\sum_{s'\in S} \delta^{\mathcal{C}}(s,(a_i,b_j))(s') \cdot v_{(n-1)+n_2}^{s',2} \end{aligned}$$

and $(v_{(n-1)+n_1}^{s',1}, v_{(n-1)+n_2}^{s',2}) = \mathtt{V}_{\mathsf{G}^{\mathcal{C}}}(s',\theta_{(n-1)+n_1,(n-1)+n_2})$ for all $s' \in S$.

Bounded Cumulative Rewards. If $\theta = \mathtt{R}^{r_1}[\,\mathtt{C}^{\leqslant k_1}\,]{+}\mathtt{R}^{r_2}[\,\mathtt{C}^{\leqslant k_2}\,]$, we compute values of the objectives for the formulae $\theta_{n+n_1,n+n_2} = \mathtt{R}^{r_1}[\,\mathtt{C}^{\leqslant n+n_1}\,] + \mathtt{R}^{r_2}[\,\mathtt{C}^{\leqslant n+n_2}\,]$ for $0 \leqslant n \leqslant k$ recursively, where $k{=}\min\{k_1,k_2\}$, $n_1{=}k_1{-}k$ and $n_2 = k_2{-}k$. For state s, if $n{=}0$:

$$\mathtt{V}_{\mathsf{G}^{\mathcal{C}}}(s,\theta_{n_1,n_2}) = \begin{cases} (0,0) & \text{if } n_1{=}n_2{=}0 \\ (0, \mathtt{R}^{\max}_{\mathsf{G},s}(r_2, \mathtt{C}^{\leqslant n_2})) & \text{else if } n_1{=}0 \\ (\mathtt{R}^{\max}_{\mathsf{G},s}(r_1, \mathtt{C}^{\leqslant n_1}), 0) & \text{otherwise} \end{cases}$$

and if $n > 0$, then $\mathtt{V}_{\mathsf{G}^{\mathcal{C}}}(s,\theta_{n+n_1,n+n_2})$ equals SWNE values of the bimatrix game $(\mathsf{Z}_1,\mathsf{Z}_2) \in \mathbb{Q}^{l\times m}$:

$$\begin{aligned} z_{i,j}^1 &= r_S^1(s) + r_A^1(s,(a_i,b_j)) + \textstyle\sum_{s'\in S} \delta^{\mathcal{C}}(s,(a_i,b_j))(s') \cdot v_{(n-1)+n_1}^{s',1} \\ z_{i,j}^2 &= r_S^2(s) + r_A^2(s,(a_i,b_j)) + \textstyle\sum_{s'\in S} \delta^{\mathcal{C}}(s,(a_i,b_j))(s') \cdot v_{(n-1)+n_2}^{s',l} \end{aligned}$$

and $(v_{(n-1)+n_1}^{s',1}, v_{(n-1)+n_2}^{s',2}) = \mathtt{V}_{\mathsf{G}^{\mathcal{C}}}(s',\theta_{(n-1)+n_1,(n-1)+n_2})$ for all $s' \in S$.

Bounded Instantaneous and Cumulative Rewards. If $\theta = \mathtt{R}^{r_1}[\,\mathtt{I}^{=k_1}\,]{+}\mathtt{R}^{r_2}[\,\mathtt{C}^{\leqslant k_2}\,]$, we compute values of the objectives for the formulae $\theta_{n+n_1,n+n_2} = \mathtt{R}^{r_1}[\,\mathtt{I}^{=n+n_1}\,] + \mathtt{R}^{r_2}[\,\mathtt{C}^{\leqslant n+n_2}\,]$ for $0 \leqslant n \leqslant k$ recursively, where $k = \min\{k_1,k_2\}$, $n_1 = k_1{-}k$ and $n_2 = k_2{-}k$. Again, here we can only switch to MDP verification when one of the step bounds equals zero. For state s, if $n = 0$:

$$\mathtt{V}_{\mathsf{G}^{\mathcal{C}}}(s,\theta_{n_1,n_2}) = \begin{cases} (r_S^1(s), 0) & \text{if } n_1 = n_2 = 0 \\ (r_S^1(s), \mathtt{R}^{\max}_{\mathsf{G},s}(r_2, \mathtt{C}^{\leqslant n_2})) & \text{else if } n_1 = 0 \\ (\mathtt{R}^{\max}_{\mathsf{G},s}(r_1, \mathtt{I}^{\leqslant n_1}), 0) & \text{otherwise} \end{cases}$$

and if $n > 0$, then $\mathtt{V}_{\mathsf{G}^{\mathcal{C}}}(s,\theta_{n+n_1,n+n_2})$ equals SWNE values of the bimatrix game $(\mathsf{Z}_1,\mathsf{Z}_2) \in \mathbb{Q}^{l\times m}$:

$$\begin{aligned} z_{i,j}^1 &= \textstyle\sum_{s'\in S} \delta^{\mathcal{C}}(s,(a_i,b_j))(s') \cdot v_{(n-1)+n_1}^{s',1} \\ z_{i,j}^2 &= r_S^2(s) + r_A^2(s,(a_i,b_j)) + \textstyle\sum_{s'\in S} \delta^{\mathcal{C}}(s,(a_i,b_j))(s') \cdot v_{(n-1)+n_2}^{s',l} \end{aligned}$$

and $(v^{s',1}_{(n-1)+n_1}, v^{s',2}_{(n-1)+n_2}) = \mathtt{V}_{\mathsf{G}^{\mathcal{C}}}(s', \theta_{(n-1)+n_1,(n-1)+n_2})$ for all $s' \in S$. The symmetric case follows similarly.

5.3.2 Computing SWNE Values of Infinite-horizon Nonzero-sum Formulae

We next show how to compute SWNE values $\mathtt{V}_{\mathsf{G}^{\mathcal{C}}}(s, \theta)$ for *infinite-horizon* nonzero-sum formulae θ in all states s of $\mathsf{G}^{\mathcal{C}}$. As for the zero-sum case, we approximate these using a value iteration approach. Each step of this computation is similar in nature to the algorithms in the previous section, where a bimatrix game is solved for each state, and a reduction to solving an MDP is used after one of the player's objective can no longer change.

A key aspect of the value iteration algorithm is that, while the SWNE (or SCNE) values take the form of a pair, with one value for each player, convergence is defined over the *sum* of the two values. This is because there is not necessarily a unique pair of such values, but the maximum (or minimum) of the sum of NE values *is* uniquely defined. Convergence of value iteration is estimated in the same way as for the zero-sum computation (see Section 4.5.2), by comparing values in successive iterations. As previously, this means that we are not able to guarantee that the computed values are within a particular error bound of the exact values.

Until. If $\theta = \mathtt{P}[\,\phi^1_1 \;\mathtt{U}\; \phi^1_2\,] + \mathtt{P}[\,\phi^2_1 \;\mathtt{U}\; \phi^2_2\,]$, values for any state s can be computed through value iteration as the limit $\mathtt{V}_{\mathsf{G}^{\mathcal{C}}}(s, \theta) = \lim_{n\to\infty} \mathtt{V}_{\mathsf{G}^{\mathcal{C}}}(s, \theta, n)$ where:

$$\mathtt{V}_{\mathsf{G}^{\mathcal{C}}}(s, \theta, n) = \begin{cases} (1,1) & \text{if } s \in Sat(\phi^1_2) \cap Sat(\phi^2_2) \\ (1, \mathtt{P}^{\max}_{\mathsf{G},s}(\phi^2_1 \;\mathtt{U}\; \phi^2_2)) & \text{else if } s \in Sat(\phi^1_2) \\ (\mathtt{P}^{\max}_{\mathsf{G},s}(\phi^1_1 \;\mathtt{U}\; \phi^1_2), 1) & \text{else if } s \in Sat(\phi^2_2) \\ (\mathtt{P}^{\max}_{\mathsf{G},s}(\phi^1_1 \;\mathtt{U}\; \phi^1_2), 0) & \text{else if } s \in Sat(\phi^1_1) \setminus Sat(\phi^2_1) \\ (0, \mathtt{P}^{\max}_{\mathsf{G},s}(\phi^2_1 \;\mathtt{U}\; \phi^2_2)) & \text{else if } s \in Sat(\phi^2_1) \setminus Sat(\phi^1_1) \\ (0,0) & \text{else if } n{=}0 \text{ or } s \notin Sat(\phi^1_1) \cap Sat(\phi^2_1) \\ val(\mathsf{Z}_1, \mathsf{Z}_2) & \text{otherwise} \end{cases}$$

where $val(\mathsf{Z}_1, \mathsf{Z}_2)$ equals SWNE values of the bimatrix game $(\mathsf{Z}_1, \mathsf{Z}_2) \in \mathbb{Q}^{l\times m}$:

$$\begin{aligned} z^1_{i,j} &= \textstyle\sum_{s'\in S} \delta^{\mathcal{C}}(s, (a_i, b_j))(s') \cdot v^{s',1}_{n-1} \\ z^2_{i,j} &= \textstyle\sum_{s'\in S} \delta^{\mathcal{C}}(s, (a_i, b_j))(s') \cdot v^{s',2}_{n-1} \end{aligned}$$

and $(v^{s',1}_{n-1}, v^{s',2}_{n-1}) = \mathtt{V}_{\mathsf{G}^{\mathcal{C}}}(s', \theta, n{-}1)$ for all $s' \in S$.

Expected Reachability. If $\theta = \mathtt{R}^{r_1}[\,\mathtt{F}\ \phi^1\,]{+}\mathtt{R}^{r_2}[\,\mathtt{F}\ \phi^2\,]$, values can be computed through value iteration as the limit $\mathtt{V}_{\mathsf{G}^{\mathcal{C}}}(s,\theta) = \lim_{n\to\infty} \mathtt{V}_{\mathsf{G}^{\mathcal{C}}}(s,\theta,n)$ where:

$$\mathtt{V}_{\mathsf{G}^{\mathcal{C}}}(s,\theta,n) = \begin{cases} (0,0) & \text{if } s \in \mathit{Sat}(\phi^1) \cap \mathit{Sat}(\phi^2) \text{ or } n{=}0 \\ (0, \mathtt{R}^{\max}_{\mathsf{G},s}(r_2, \mathtt{F}\ \phi^2)) & \text{else if } s \in \mathit{Sat}(\phi^1) \\ (\mathtt{R}^{\max}_{\mathsf{G},s}(r_1, \mathtt{F}\ \phi^1), 0) & \text{else if } s \in \mathit{Sat}(\phi^2) \\ \mathit{val}(\mathsf{Z}_1, \mathsf{Z}_2) & \text{otherwise} \end{cases}$$

where $\mathit{val}(\mathsf{Z}_1, \mathsf{Z}_2)$ equals SWNE values of the bimatrix game $(\mathsf{Z}_1, \mathsf{Z}_2) \in \mathbb{Q}^{l\times m}$:

$$\begin{aligned} z^1_{i,j} &= r^1_S(s) + r^1_A(s,(a_i,b_j)) + \textstyle\sum_{s'\in S} \delta^{\mathcal{C}}(s,(a_i,b_j))(s') \cdot v^{s',1}_{n-1} \\ z^2_{i,j} &= r^2_S(s) + r^2_A(s,(a_i,b_j)) + \textstyle\sum_{s'\in S} \delta^{\mathcal{C}}(s,(a_i,b_j))(s') \cdot v^{s',2}_{n-1} \end{aligned}$$

and $(v^{s',1}_{n-1}, v^{s',2}_{n-1}) = \mathtt{V}_{\mathsf{G}^{\mathcal{C}}}(s',\theta,n{-}1)$ for all $s' \in S$.

5.3.3 Computing SCNE Values of Finite-horizon Nonzero-sum Formulae

In this section, we show how to compute SCNE values $\overline{\mathtt{V}}_{\mathsf{G}^{\mathcal{C}}}(s,\theta)$ for *finite-horizon* nonzero-sum formula θ in all states s of $\mathsf{G}^{\mathcal{C}}$. The two differences between the computation of SWNE and SCNE values are:

- When constructing the bimatrix game for each state, we negate the values corresponding to their entries and subsequently negate the computed SWNE values;
- When switching to MDP computation, we compute the values for $\mathtt{P}^{\min}_{\mathsf{G},s}(\psi)$ and $\mathtt{R}^{\min}_{\mathsf{G},s}(r,\rho)$ for probabilistic and reward formulae, respectively.

All other arguments in Section 5.3.1 concerning finite-horizon formula also apply here. Below, we explain the computation for both types of finite-horizon probabilistic objectives (next and bounded until) and reward objectives (instantaneous and bounded cumulative), as well as combinations of each type.

Next. If $\theta = \mathtt{P}[\,\mathtt{X}\,\phi^1\,]{+}\mathtt{P}[\,\mathtt{X}\,\phi^2\,]$, then $\overline{\mathtt{V}}_{\mathsf{G}^{\mathcal{C}}}(s,\theta)$ equals the negation of the SWNE values of the bimatrix game $(\mathsf{Z}_1, \mathsf{Z}_2) \in \mathbb{Q}^{l\times m}$ where:

$$\begin{aligned} z^1_{i,j} &= -\textstyle\sum_{s'\in S} \delta^{\mathcal{C}}(s,(a_i,b_j))(s') \cdot \eta_{\phi^1}(s') \\ z^2_{i,j} &= -\textstyle\sum_{s'\in S} \delta^{\mathcal{C}}(s,(a_i,b_j))(s') \cdot \eta_{\phi^2}(s')\,. \end{aligned}$$

Bounded Until. If $\theta = \mathtt{P}[\,\phi^1_1\ \mathtt{U}^{\leqslant k_1}\ \phi^1_2\,] + \mathtt{P}[\,\phi^2_1\ \mathtt{U}^{\leqslant k_2}\ \phi^2_2\,]$, we compute SWNE values for the objectives for the nonzero-sum formulae $\theta_{n+n_1,n+n_2} = \mathtt{P}[\,\phi^1_1\ \mathtt{U}^{\leqslant n+n_1}\ \phi^1_2\,] +$

$\mathtt{P}[\,\phi_1^2\ \mathtt{U}^{\leqslant n+n_2}\ \phi_2^2\,]$ for $0 \leqslant n \leqslant k$ recursively, where $k{=}\min\{k_1,k_2\}$, $n_1{=}k_1{-}k$ and $n_2 = k_2{-}k$. For any state s, if $n{=}0$, then:

$$\overline{\mathtt{V}}_{\mathsf{G}^{\mathcal{C}}}(s,\theta_{n_1,n_2}) = \begin{cases} (\eta_{\phi_2^1}(s),\eta_{\phi_2^2}(s)) & \text{if } n_1{=}n_2{=}0 \\ (\eta_{\phi_2^1}(s),\mathtt{P}^{\min}_{\mathsf{G},s}(\phi_1^2\ \mathtt{U}^{\leqslant n_2}\ \phi_2^2)) & \text{else if } n_1{=}0 \\ (\mathtt{P}^{\min}_{\mathsf{G},s}(\phi_1^1\ \mathtt{U}^{\leqslant n_1}\ \phi_2^1),\eta_{\phi_2^2}(s)) & \text{otherwise} \end{cases}$$

On the other hand, if $n > 0$, then:

$$\overline{\mathtt{V}}_{\mathsf{G}^{\mathcal{C}}}(s,\theta_{n+n_1,n+n_2}) = \begin{cases} (1,1) & \text{if } s \in Sat(\phi_2^1) \cap Sat(\phi_2^2) \\ (1,\mathtt{P}^{\min}_{\mathsf{G},s}(\phi_1^2\ \mathtt{U}^{\leqslant n+n_2}\ \phi_2^2)) & \text{else if } s \in Sat(\phi_2^1) \\ (\mathtt{P}^{\min}_{\mathsf{G},s}(\phi_1^1\ \mathtt{U}^{\leqslant n+n_1}\ \phi_2^1),1) & \text{else if } s \in Sat(\phi_2^2) \\ (\mathtt{P}^{\min}_{\mathsf{G},s}(\phi_1^1\ \mathtt{U}^{\leqslant n+n_1}\ \phi_2^1),0) & \text{else if } s \in Sat(\phi_1^1) \setminus Sat(\phi_1^2) \\ (0,\mathtt{P}^{\min}_{\mathsf{G},s}(\phi_1^2\ \mathtt{U}^{\leqslant n+n_2}\ \phi_2^2)) & \text{else if } s \in Sat(\phi_1^2) \setminus Sat(\phi_1^1) \\ (0,0) & \text{else if } s \notin Sat(\phi_1^1) \cap Sat(\phi_1^2) \\ -val(\mathsf{Z}_1,\mathsf{Z}_2) & \text{otherwise} \end{cases}$$

where $val(\mathsf{Z}_1,\mathsf{Z}_2)$ equals the SWNE values of the bimatrix game $(\mathsf{Z}_1,\mathsf{Z}_2) \in \mathbb{Q}^{l\times m}$:

$$\begin{aligned} z^1_{i,j} &= -\textstyle\sum_{s'\in S} \delta^{\mathcal{C}}(s,(a_i,b_j))(s') \cdot v^{s',1}_{(n-1)+n_1} \\ z^2_{i,j} &= -\textstyle\sum_{s'\in S} \delta^{\mathcal{C}}(s,(a_i,b_j))(s') \cdot v^{s',2}_{(n-1)+n_2} \end{aligned}$$

and $(v^{s',1}_{(n-1)+n_1}, v^{s',2}_{(n-1)+n_2}) = \overline{\mathtt{V}}_{\mathsf{G}^{\mathcal{C}}}(s',\theta_{(n-1)+n_1,(n-1)+n_2})$ for all $s' \in S$.

Next and Bounded Until. If $\theta = \mathtt{P}[\,\mathtt{X}\,\phi^1\,]{+}\mathtt{P}[\,\phi_1^2\ \mathtt{U}^{\leqslant k_2}\ \phi_2^2\,]$, then $\overline{\mathtt{V}}_{\mathsf{G}^{\mathcal{C}}}(s,\theta)$ equals the negation of the SWNE values of the bimatrix game $(\mathsf{Z}_1,\mathsf{Z}_2) \in \mathbb{Q}^{l\times m}$ where:

$$\begin{aligned} z^1_{i,j} &= -\textstyle\sum_{s'\in S} \delta^{\mathcal{C}}(s,(a_i,b_j))(s') \cdot \eta_{\phi^1}(s') \\ z^2_{i,j} &= -1 \times \begin{cases} 1 & \text{if } s \in Sat(\phi_2^2) \\ 0 & \text{else if } k_2 = 0 \\ \sum_{s'\in S} \delta^{\mathcal{C}}(s,(a_i,b_j))(s') \cdot \mathtt{P}^{\min}_{\mathsf{G},s}(\phi_1^2\ \mathtt{U}^{\leqslant k_2-1}\ \phi_2^2) & \text{else if } Sat(\phi_1^2) \\ 0 & \text{otherwise.} \end{cases} \end{aligned}$$

In this case, since the value for objectives corresponding to next formulae cannot change after the first step, we can always switch to MDP verification after this step. The symmetric case is similar.

Instantaneous Rewards. If $\theta{=}\mathtt{R}^{r_1}[\,\mathtt{I}^{=k_1}\,]{+}\mathtt{R}^{r_2}[\,\mathtt{I}^{=k_2}\,]$, we compute SWNE values of the objectives for the nonzero-sum formulae $\theta_{n+n_1,n+n_2} = \mathtt{R}^{r_1}[\,\mathtt{I}^{=n+n_1}\,] + \mathtt{R}^{r_2}[\,\mathtt{I}^{=n+n_2}\,]$ for $0 \leqslant n \leqslant k$ recursively, where $k{=}\min\{k_1,k_2\}$, $n_1 = k_1{-}k$ and $n_2{=}k_2{-}k$. For any

state s, if n=0, then:

$$\overline{\mathtt{V}}_{\mathsf{G}^{\mathcal{C}}}(s,\theta_{n_1,n_2}) = \begin{cases} (r_S^1(s), r_S^2(s)) & \text{if } n_1{=}n_2{=}0 \\ (r_S^1(s), \mathtt{R}^{\min}_{\mathsf{G},s}(r_2, \mathtt{I}^{=n_2})) & \text{else if } n_1{=}0 \\ (\mathtt{R}^{\min}_{\mathsf{G},s}(r_1, \mathtt{I}^{=n_1}), r_S^2(s)) & \text{otherwise.} \end{cases}$$

On the other hand, if $n > 0$, then $\overline{\mathtt{V}}_{\mathsf{G}^{\mathcal{C}}}(s,\theta_{n+n_1,n+n_2})$ equals the negation of the SWNE values of the bimatrix game $(\mathsf{Z}_1, \mathsf{Z}_2) \in \mathbb{Q}^{l\times m}$ where:

$$\begin{aligned} z^1_{i,j} = & \ -\textstyle\sum_{s'\in S} \delta^{\mathcal{C}}(s,(a_i,b_j))(s') \cdot v^{s',1}_{(n-1)+n_1} \\ z^2_{i,j} = & \ -\textstyle\sum_{s'\in S} \delta^{\mathcal{C}}(s,(a_i,b_j))(s') \cdot v^{s',2}_{(n-1)+n_2} \end{aligned}$$

and $(v^{s',1}_{(n-1)+n_1}, v^{s',2}_{(n-1)+n_2}) = \overline{\mathtt{V}}_{\mathsf{G}^{\mathcal{C}}}(s', \theta_{(n-1)+n_1,(n-1)+n_2})$ for all $s' \in S$.

Bounded Cumulative Rewards. If $\theta = \mathtt{R}^{r_1}[\,\mathtt{C}^{\leqslant k_1}\,] + \mathtt{R}^{r_2}[\,\mathtt{C}^{\leqslant k_2}\,]$, we compute values of the objectives for the formulae $\theta_{n+n_1,n+n_2} = \mathtt{R}^{r_1}[\,\mathtt{C}^{\leqslant n+n_1}\,] + \mathtt{R}^{r_2}[\,\mathtt{C}^{\leqslant n+n_2}\,]$ for $0 \leqslant n \leqslant k$ recursively, where $k{=}\min\{k_1,k_2\}$, $n_1{=}k_1{-}k$ and $n_2 = k_2{-}k$. For state s, if n=0:

$$\overline{\mathtt{V}}_{\mathsf{G}^{\mathcal{C}}}(s,\theta_{n_1,n_2}) = \begin{cases} (0,0) & \text{if } n_1{=}n_2{=}0 \\ (0, \mathtt{R}^{\min}_{\mathsf{G},s}(r_2, \mathtt{C}^{\leqslant n_2})) & \text{else if } n_1{=}0 \\ (\mathtt{R}^{\min}_{\mathsf{G},s}(r_1, \mathtt{C}^{\leqslant n_1}), 0) & \text{otherwise} \end{cases}$$

and if $n > 0$, then $\overline{\mathtt{V}}_{\mathsf{G}^{\mathcal{C}}}(s,\theta_{n+n_1,n+n_2})$ equals the negation of the SWNE values of the bimatrix game $(\mathsf{Z}_1, \mathsf{Z}_2) \in \mathbb{Q}^{l\times m}$:

$$\begin{aligned} z^1_{i,j} = & \ -\left(r_S^1(s) + r_A^1(s,(a_i,b_j)) + \textstyle\sum_{s'\in S} \delta^{\mathcal{C}}(s,(a_i,b_j))(s') \cdot v^{s',1}_{(n-1)+n_1}\right) \\ z^2_{i,j} = & \ -\left(r_S^2(s) + r_A^2(s,(a_i,b_j)) + \textstyle\sum_{s'\in S} \delta^{\mathcal{C}}(s,(a_i,b_j))(s') \cdot v^{s',2}_{(n-1)+n_2}\right) \end{aligned}$$

and $(v^{s',1}_{(n-1)+n_1}, v^{s',2}_{(n-1)+n_2}) = \overline{\mathtt{V}}_{\mathsf{G}^{\mathcal{C}}}(s', \theta_{(n-1)+n_1,(n-1)+n_2})$ for all $s' \in S$.

Bounded Instantaneous and Cumulative Rewards. If $\theta = \mathtt{R}^{r_1}[\,\mathtt{I}^{=k_1}\,] + \mathtt{R}^{r_2}[\,\mathtt{C}^{\leqslant k_2}\,]$, we compute values of the objectives for the formulae $\theta_{n+n_1,n+n_2} = \mathtt{R}^{r_1}[\,\mathtt{I}^{=n+n_1}\,] + \mathtt{R}^{r_2}[\,\mathtt{C}^{\leqslant n+n_2}\,]$ for $0 \leqslant n \leqslant k$ recursively, where $k = \min\{k_1,k_2\}$, $n_1 = k_1{-}k$ and $n_2 = k_2{-}k$. Again, here we can only switch to MDP verification when one of the step bounds equals zero. For state s, if $n = 0$:

$$\overline{\mathtt{V}}_{\mathsf{G}^{\mathcal{C}}}(s,\theta_{n_1,n_2}) = \begin{cases} (r_S^1(s), 0) & \text{if } n_1 = n_2 = 0 \\ (r_S^1(s), \mathtt{R}^{\min}_{\mathsf{G},s}(r_2, \mathtt{C}^{\leqslant n_2})) & \text{else if } n_1 = 0 \\ (\mathtt{R}^{\min}_{\mathsf{G},s}(r_1, \mathtt{I}^{\leqslant n_1}), 0) & \text{otherwise} \end{cases}$$

and if $n > 0$, then $\overline{\mathsf{V}}_{\mathsf{G}^{\mathcal{C}}}(s, \theta_{n+n_1,n+n_2})$ equals the negation of the SWNE values of the bimatrix game $(\mathsf{Z}_1, \mathsf{Z}_2) \in \mathbb{Q}^{l\times m}$:

$$z^1_{i,j} = -\left(\textstyle\sum_{s'\in S} \delta^{\mathcal{C}}(s,(a_i,b_j))(s') \cdot v^{s',1}_{(n-1)+n_1}\right)$$
$$z^2_{i,j} = -\left(r^2_S(s) + r^2_A(s,(a_i,b_j)) + \textstyle\sum_{s'\in S} \delta^{\mathcal{C}}(s,(a_i,b_j))(s') \cdot v^{s',l}_{(n-1)+n_2}\right)$$

and $(v^{s',1}_{(n-1)+n_1}, v^{s',2}_{(n-1)+n_2}) = \mathsf{V}_{\mathsf{G}^{\mathcal{C}}}(s', \theta_{(n-1)+n_1,(n-1)+n_2})$ for all $s' \in S$. The symmetric case follows similarly.

5.3.4 Computing SCNE Values of Infinite-horizon Nonzero-sum Formulae

We next show how to compute SCNE values $\overline{\mathsf{V}}_{\mathsf{G}^{\mathcal{C}}}(s,\theta)$ for *finite-horizon* nonzero-sum formula θ in all states s of $\mathsf{G}^{\mathcal{C}}$. The same changes described for finite-horizon formulae in Section 5.3.3 and the arguments concerning uniqueness and convergence for the computation of infinite-horizon SWNE values described in Section 5.3.2 apply here.

Until. If $\theta = \mathtt{P}[\,\phi^1_1\ \mathtt{U}\ \phi^1_2\,]{+}\mathtt{P}[\,\phi^2_1\ \mathtt{U}\ \phi^2_2\,]$, values for any state s can be computed through value iteration as the limit $\overline{\mathsf{V}}_{\mathsf{G}^{\mathcal{C}}}(s,\theta) = \lim_{n\to\infty} \overline{\mathsf{V}}_{\mathsf{G}^{\mathcal{C}}}(s,\theta,n)$ where:

$$\overline{\mathsf{V}}_{\mathsf{G}^{\mathcal{C}}}(s,\theta,n) = \begin{cases} (1,1) & \text{if } s \in Sat(\phi^1_2) \cap Sat(\phi^2_2) \\ (1, \mathtt{P}^{\min}_{\mathsf{G},s}(\phi^2_1\ \mathtt{U}\ \phi^2_2)) & \text{else if } s \in Sat(\phi^1_2) \\ (\mathtt{P}^{\min}_{\mathsf{G},s}(\phi^1_1\ \mathtt{U}\ \phi^1_2), 1) & \text{else if } s \in Sat(\phi^2_2) \\ (\mathtt{P}^{\min}_{\mathsf{G},s}(\phi^1_1\ \mathtt{U}\ \phi^1_2), 0) & \text{else if } s \in Sat(\phi^1_1) \setminus Sat(\phi^2_1) \\ (0, \mathtt{P}^{\min}_{\mathsf{G},s}(\phi^2_1\ \mathtt{U}\ \phi^2_2)) & \text{else if } s \in Sat(\phi^2_1) \setminus Sat(\phi^1_1) \\ (0,0) & \text{else if } n{=}0 \text{ or } s \notin Sat(\phi^1_1) \cap Sat(\phi^2_1) \\ -val(\mathsf{Z}_1,\mathsf{Z}_2) & \text{otherwise} \end{cases}$$

where $val(\mathsf{Z}_1,\mathsf{Z}_2)$ equals SWNE values of the bimatrix game $(\mathsf{Z}_1,\mathsf{Z}_2) \in \mathbb{Q}^{l\times m}$:

$$z^1_{i,j} = -\textstyle\sum_{s'\in S} \delta^{\mathcal{C}}(s,(a_i,b_j))(s') \cdot v^{s',1}_{n-1}$$
$$z^2_{i,j} = -\textstyle\sum_{s'\in S} \delta^{\mathcal{C}}(s,(a_i,b_j))(s') \cdot v^{s',2}_{n-1}$$

and $(v^{s',1}_{n-1}, v^{s',2}_{n-1}) = \overline{\mathsf{V}}_{\mathsf{G}^{\mathcal{C}}}(s',\theta,n{-}1)$ for all $s' \in S$.

Expected Reachability. If $\theta = \mathtt{R}^{r_1}[\,\mathtt{F}\ \phi^1\,]{+}\mathtt{R}^{r_2}[\,\mathtt{F}\ \phi^2\,]$, values can be computed through value iteration as the limit $\overline{\mathsf{V}}_{\mathsf{G}^{\mathcal{C}}}(s,\theta) = \lim_{n\to\infty} \overline{\mathsf{V}}_{\mathsf{G}^{\mathcal{C}}}(s,\theta,n)$ where:

$$\overline{\mathsf{V}}_{\mathsf{G}^{\mathcal{C}}}(s,\theta,n) = \begin{cases} (0,0) & \text{if } s \in Sat(\phi^1) \cap Sat(\phi^2) \text{ or } n{=}0 \\ (0, \mathtt{R}^{\min}_{\mathsf{G},s}(r_2, \mathtt{F}\ \phi^2)) & \text{else if } s \in Sat(\phi^1) \\ (\mathtt{R}^{\min}_{\mathsf{G},s}(r_1, \mathtt{F}\ \phi^1), 0) & \text{else if } s \in Sat(\phi^2) \\ -val(\mathsf{Z}_1,\mathsf{Z}_2) & \text{otherwise} \end{cases}$$

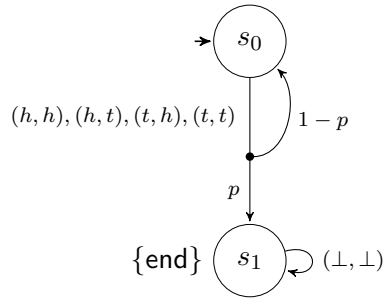
where $val(\mathsf{Z}_1, \mathsf{Z}_2)$ equals SWNE values of the bimatrix game $(\mathsf{Z}_1, \mathsf{Z}_2) \in \mathbb{Q}^{l \times m}$:

$$z^1_{i,j} = -\left(r^1_S(s) + r^1_A(s,(a_i,b_j)) + \textstyle\sum_{s' \in S} \delta^{\mathcal{C}}(s,(a_i,b_j))(s') \cdot v^{s',1}_{n-1}\right)$$
$$z^2_{i,j} = -\left(r^2_S(s) + r^2_A(s,(a_i,b_j)) + \textstyle\sum_{s' \in S} \delta^{\mathcal{C}}(s,(a_i,b_j))(s') \cdot v^{s',2}_{n-1}\right)$$

and $(v^{s',1}_{n-1}, v^{s',2}_{n-1}) = \overline{\mathsf{V}}_{\mathsf{G}^{\mathcal{C}}}(s', \theta, n-1)$ for all $s' \in S$.

In Example 17, we show the application of the model checking algorithm described in Section 5.3.2 to compute values for a infinite-horizon equilibrium property. The example shows how the bimatrix games are constructed and how SWNE values are approximated through a combination of value iteration and the computation of local (state) NE values.

Example 17. Consider the problem of computing values for the equilibria property $\phi = \langle\langle \mathit{player}_1 : \mathit{player}_2 \rangle\rangle_{\max=?}(\mathsf{R}^{r_1}[\,\mathsf{F}\,\mathsf{end}\,] + \mathsf{R}^{r_2}[\,\mathsf{F}\,\mathsf{end}\,])$ for an instance of the *matching pennies* problem shown in Figure 5.3 (left). Here we consider the problem of playing the game repeatedly, where at each round there is a probability of $p > 0$ that the game ends. Player 1 wins a round if both players play heads or tails, while the other wins if the coins are different, that is: $r^1_A(s_0,(h,h)) = r^1_A(s_0,(t,t)) = r^2_A(s_0,(h,t)) = r^2_A(s_0,(t,h)) = 1$. At each round, we need to solve the bimatrix game shown in Figure 5.3 (right) for state s_0.



(a) Matching pennies CSG.

$$\mathsf{Z}^n_1 = \begin{array}{c} \\ h \\ t \end{array} \begin{array}{c} \begin{array}{cc} \quad h & \qquad t \end{array} \\ \begin{pmatrix} 1+(1-p)\cdot v^{s_0,1}_{n-1} & (1-p)\cdot v^{s_0,1}_{n-1} \\ (1-p)\cdot v^{s_0,1}_{n-1} & 1+(1-p)\cdot v^{s_0,1}_{n-1} \end{pmatrix} \end{array}$$

$$\mathsf{Z}^n_2 = \begin{array}{c} \\ h \\ t \end{array} \begin{array}{c} \begin{array}{cc} \quad h & \qquad t \end{array} \\ \begin{pmatrix} (1-p)\cdot v^{s_0,2}_{n-1} & 1+(1-p)\cdot v^{s_0,2}_{n-1} \\ 1+(1-p)\cdot v^{s_0,2}_{n-1} & (1-p)\cdot v^{s_0,2}_{n-1} \end{pmatrix} \end{array}$$

(b) Bimatrix game for state s_0.

Figure 5.3: An instance of matching pennies in extensive and normal forms (one round).

Computing that the equilibrium strategy for both players at any round, we see that it is playing h or t with equal probability, i.e., $\sigma_1 = \sigma_2 = (\frac{1}{2}, \frac{1}{2})$. We also have that the value for each player i is equal to $v^{s_0,i}_n = \frac{1}{2} + v^{s_0,i}_{n-1} - p \cdot v^{s_0,i}_{n-1}$. For instance, if

$p = 1/2$ we have:

$$\mathsf{Z}_1^1 = \begin{matrix} \\ h \\ t \end{matrix}\!\!\begin{matrix} \begin{matrix} h & t \end{matrix} \\ \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \end{matrix}, \mathsf{Z}_1^2 = \begin{matrix} \\ h \\ t \end{matrix}\!\!\begin{matrix} \begin{matrix} h & t \end{matrix} \\ \begin{pmatrix} \frac{5}{4} & \frac{1}{4} \\ \frac{1}{4} & \frac{5}{4} \end{pmatrix} \end{matrix}, \mathsf{Z}_1^3 = \begin{matrix} \\ h \\ t \end{matrix}\!\!\begin{matrix} \begin{matrix} h & t \end{matrix} \\ \begin{pmatrix} \frac{11}{8} & \frac{3}{8} \\ \frac{3}{8} & \frac{11}{8} \end{pmatrix} \end{matrix}, \mathsf{Z}_1^4 = \begin{matrix} \\ h \\ t \end{matrix}\!\!\begin{matrix} \begin{matrix} h & t \end{matrix} \\ \begin{pmatrix} \frac{23}{16} & \frac{7}{16} \\ \frac{7}{16} & \frac{23}{16} \end{pmatrix} \end{matrix} \cdots$$

$$\mathsf{Z}_2^1 = \begin{matrix} \\ h \\ t \end{matrix}\!\!\begin{matrix} \begin{matrix} h & t \end{matrix} \\ \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \end{matrix}, \mathsf{Z}_2^2 = \begin{matrix} \\ h \\ t \end{matrix}\!\!\begin{matrix} \begin{matrix} h & t \end{matrix} \\ \begin{pmatrix} \frac{1}{4} & \frac{5}{4} \\ \frac{5}{4} & \frac{1}{4} \end{pmatrix} \end{matrix}, \mathsf{Z}_2^3 = \begin{matrix} \\ h \\ t \end{matrix}\!\!\begin{matrix} \begin{matrix} h & t \end{matrix} \\ \begin{pmatrix} \frac{3}{8} & \frac{11}{8} \\ \frac{11}{8} & \frac{3}{8} \end{pmatrix} \end{matrix}, \mathsf{Z}_2^4 = \begin{matrix} \\ h \\ t \end{matrix}\!\!\begin{matrix} \begin{matrix} h & t \end{matrix} \\ \begin{pmatrix} \frac{7}{16} & \frac{23}{16} \\ \frac{23}{16} & \frac{7}{16} \end{pmatrix} \end{matrix} \cdots$$

At the nth iteration, the sum of SWNE values can be expressed as: $v_n^{s_0,1} + v_n^{s_0,2} = 1 + v_{n-1}^{s_0,1} + v_{n-1}^{s_0,2} - p{\cdot}(v_{n-1}^{s_0,1} + v_{n-1}^{s_0,2})$. Let $v_n^{s_0}$ be equal to the sum, we then have $v_n^{s_0} = 1 + v_{n-1}^{s_0} - p{\cdot}v_{n-1}^{s_0}$. In the limit, we have $\lim_{n\to\infty} v_n^{s_0} = \frac{1}{p}$, with each player getting $\frac{1}{2p}$. ■

5.3.5 Computing SWNE Values of Mixed Nonzero-sum Formulae

In this section, we present the model transformation needed for computing values of nonzero-sum state formulae containing a *mixture* of both finite- and infinite-horizon objectives. For these cases, the problem is reduced to computing values of a formula θ' that contains only infinite-horizon objectives on a modified game G'. The approach is based on the standard construction for converting the verification of finite-horizon properties to infinite-horizon properties [175]. We fix a game $\mathsf{G}^{\mathcal{C}}$ and non-zero sum state formula $\langle\!\langle C{:}C' \rangle\!\rangle_{\text{opt}\sim x}(\theta)$. We consider the case when the first objective is finite-horizon and second infinite-horizon; the symmetric cases follow similarly. In each case, the modified game has states of the form (s, n), where s is a state of $\mathsf{G}^{\mathcal{C}}$ and $n \in \mathbb{N}$ and values for the state $(s, 0)$ of the new Nash formula correspond to values of state s for the original Nash formula.

Next and Unbounded Until. If $\theta = \mathtt{P}[\,\mathtt{X}\,\phi^1\,] + \mathtt{P}[\,\phi_1^2\ \mathtt{U}\ \phi_2^2\,]$, then we construct the game $\mathsf{G}' = (\{1, 2\}, S', \bar{S}', A^{\mathcal{C}}, \Delta', \delta', \{\mathsf{a}_{\phi^1}, \mathsf{a}_{\phi_1^2}, \mathsf{a}_{\phi_2^2}\}, L')$ where:

- $S' = \{(s, n) \mid s \in S \wedge 0 \leqslant n \leqslant 2\}$ and $\bar{S}' = \{(s, 0) \mid s \in S\}$;
- $\Delta'((s, n)) = \Delta^{\mathcal{C}}(s)$ for all $(s, n) \in S'$;
- for any $(s, n), (s', n') \in S'$ and $a \in A^{\mathcal{C}}$:

$$\delta'((s, n), a)((s', n')) = \begin{cases} \delta^{\mathcal{C}}(s, a)(s') & \text{if } 0 \leqslant n \leqslant 1 \text{ and } n' = n{+}1 \\ \delta^{\mathcal{C}}(s, a)(s') & \text{else if } n = n' = 2 \\ 0 & \text{otherwise;} \end{cases}$$

- for any $(s,n) \in S'$ and $1 \leqslant j \leqslant 2$:
  - $\mathsf{a}_{\phi^1} \in L'((s,n))$ if and only if $s \in Sat(\phi^1)$ and $n = 1$;
  - $\mathsf{a}_{\phi_j^2} \in L'((s,n))$ if and only if $s \in Sat(\phi_j^2)$.

and compute the SWNE values of $\theta' = \mathtt{P}[\,\mathtt{true}\ \mathtt{U}\ \mathsf{a}_{\phi^1}\,] + \mathtt{P}[\,\mathsf{a}_{\phi_1^2}\ \mathtt{U}\ \mathsf{a}_{\phi_2^2}\,]$ for G'.

Bounded and Unbounded Until. If $\theta = \mathtt{P}[\,\phi_1^1\ \mathtt{U}^{\leqslant k_1}\ \phi_2^1\,] + \mathtt{P}[\,\phi_1^2\ \mathtt{U}\ \phi_2^2\,]$, then we construct the game $\mathsf{G}' = (\{1,2\}, S', \bar{S}', A^{\mathcal{C}}, \Delta', \delta', \{\mathsf{a}_{\phi_1^1}, \mathsf{a}_{\phi_2^1}, \mathsf{a}_{\phi_1^2}, \mathsf{a}_{\phi_2^2}\}, L')$ where:

- $S' = \{(s,n) \mid s \in S \wedge 0 \leqslant n \leqslant k_1{+}1\}$ and $\bar{S}' = \{(s,0) \mid s \in S\}$;
- $\Delta'((s,n)) = \Delta^{\mathcal{C}}(s)$ for all $(s,n) \in S'$;
- for any $(s,n),(s',n') \in S'$ and $a \in A^{\mathcal{C}}$:

$$\delta'((s,n),a)((s',n')) = \begin{cases} \delta^{\mathcal{C}}(s,a)(s') & \text{if } 0 \leqslant n \leqslant k_1 \text{ and } n' = n{+}1 \\ \delta^{\mathcal{C}}(s,a)(s') & \text{else if } n = n' = k_1{+}1 \\ 0 & \text{otherwise;} \end{cases}$$

- for any $(s,n) \in S'$ and $1 \leqslant j \leqslant 2$:
  - $\mathsf{a}_{\phi_1^1} \in L'((s,n))$ if and only if $s \in Sat(\phi_1^1)$ and $0 \leqslant n \leqslant k_1$;
  - $\mathsf{a}_{\phi_2^1} \in L'((s,n))$ if and only if $s \in Sat(\phi_2^1)$ and $0 \leqslant n \leqslant k_1$;
  - $\mathsf{a}_{\phi_j^2} \in L'((s,n))$ if and only if $s \in Sat(\phi_j^2)$.

and compute the SWNE values of $\theta' = \mathtt{P}[\,\mathsf{a}_{\phi_1^1}\ \mathtt{U}\ \mathsf{a}_{\phi_2^1}\,] + \mathtt{P}[\,\mathsf{a}_{\phi_1^2}\ \mathtt{U}\ \mathsf{a}_{\phi_2^2}\,]$ for G'.

Bounded Instantaneous and Expected Rewards. If $\theta = \mathtt{R}^{r_1}[\,\mathtt{I}^{=k_1}\,] + \mathtt{R}^{r_2}[\,\mathtt{F}\ \phi^2\,]$, then we construct the game $\mathsf{G}' = (\{1,2\}, S', \bar{S}', A^{\mathcal{C}}, \Delta', \delta', \{\mathsf{a}_{k_1+1}, \mathsf{a}_{\phi^2}\}, L')$ and reward structures r_1' and r_2' where:

- $S' = \{(s,n) \mid s \in S \wedge 0 \leqslant n \leqslant k_1{+}1\}$ and $\bar{S}' = \{(s,0) \mid s \in S\}$;
- $\Delta'((s,n)) = \Delta^{\mathcal{C}}(s)$ for all $(s,n) \in S'$;
- for any $(s,n),(s',n') \in S'$ and $a \in A^{\mathcal{C}}$:

$$\delta'((s,n),a)((s',n')) = \begin{cases} \delta^{\mathcal{C}}(s,a)(s') & \text{if } 0 \leqslant n \leqslant k_1 \text{ and } n' = n{+}1 \\ \delta^{\mathcal{C}}(s,a)(s') & \text{else if } n = n' = k_1{+}1 \\ 0 & \text{otherwise;} \end{cases}$$

- for any $(s,n) \in S'$:

- $\mathsf{a}_{k_1+1} \in L'((s,n))$ if and only if $n = k_1{+}1$;
- $\mathsf{a}_{\phi^2} \in L'((s,n))$ if and only if $s \in Sat(\phi^2)$;

- for any $(s,n) \in S'$ and $a \in A^{\mathcal{C}}$:
  - $r_A^{1'}((s,n),a) = 0$ and $r_S^{1'}((s,n)) = r_S^{1^{\mathcal{C}}}(s)$ if $n = k_1$ and $r_A^{1'}((s,n),a) = 0$ and $r_S^{1'}((s,n)) = 0$ otherwise;
  - $r_A^{2'}((s,n),a) = r_A^{2^{\mathcal{C}}}(s)(a)$ and $r_S^{2'}((s,n)) = r_S^{2^{\mathcal{C}}}(s)$.

and compute the SWNE values of $\theta' = \mathtt{R}^{r_1'}[\,\mathtt{F}\ \mathsf{a}_{k_1+1}\,]{+}\mathtt{R}^{r_2'}[\,\mathtt{F}\ \mathsf{a}_{\phi^2}\,]$ for G'.

Bounded Cumulative and Expected Rewards. If $\theta = \mathtt{R}^{r_1}[\,\mathtt{C}^{\leqslant k_1}\,] + \mathtt{R}^{r_2}[\,\mathtt{F}\ \phi^2\,]$, then we construct the game $\mathsf{G}' = (\{1,2\}, S', \bar{S}', A^{\mathcal{C}}, \Delta', \delta', \{\mathsf{a}_{k_1}, \mathsf{a}_{\phi^2}\}, L')$ and reward structures r_1' and r_2' where:

- $S' = \{(s,n) \mid s \in S \wedge 0 \leqslant n \leqslant k_1\}$ and $\bar{S}' = \{(s,0) \mid s \in S\}$;
- $\Delta'((s,n)) = \Delta^{\mathcal{C}}(s)$ for all $(s,n) \in S'$;
- for any $(s,n),(s',n') \in S'$ and $a \in A^{\mathcal{C}}$:

$$\delta'((s,n),a)((s',n')) = \begin{cases} \delta^{\mathcal{C}}(s,a)(s') & \text{if } 0 \leqslant n \leqslant k_1{-}1 \text{ and } n' = n{+}1 \\ \delta^{\mathcal{C}}(s,a)(s') & \text{else if } n = n' = k_1 \\ 0 & \text{otherwise;} \end{cases}$$

- for any $(s,n) \in S'$:
  - $\mathsf{a}_{k_1} \in L'((s,n))$ if and only if $n = k_1$;
  - $\mathsf{a}_{\phi^2} \in L'((s,n))$ if and only if $s \in Sat(\phi^2)$;

- for any $(s,n) \in S'$ and $a \in A^{\mathcal{C}}$:
  - $r_A^{1'}((s,n),a) = r_A^{1^{\mathcal{C}}}(s)$ if $0 \leqslant n \leqslant k_1{-}1$ and equals 0 otherwise;
  - $r_S^{1'}((s,n)) = r_S^{1^{\mathcal{C}}}(s)$ if $0 \leqslant n \leqslant k_1{-}1$ and equals 0 otherwise;
  - $r_A^{2'}((s,n),a) = r_A^{2^{\mathcal{C}}}(s)(a)$ and $r_S^{2'}((s,n)) = r_S^{2^{\mathcal{C}}}(s)$.

and compute the SWNE values of $\theta' = \mathtt{R}^{r_1'}[\,\mathtt{F}\ \mathsf{a}_{k_1}\,]{+}\mathtt{R}^{r_2'}[\,\mathtt{F}\ \mathsf{a}_{\phi^2}\,]$ for G'.

5.4 Strategy Synthesis

We now discuss strategy synthesis for equilibria properties. For each nonzero-sum formula $\langle\langle C{:}C' \rangle\rangle_{\text{opt}\sim x}(\theta)$ appearing as a sub-formula, this is a subgame-perfect SWNE or SCNE profile for the objectives (X_1^θ, X_2^θ) in the coalition game $\mathsf{G}^{\mathcal{C}}$. We can perform strategy synthesis by adapting the model checking algorithms described in the previous sections which computes SWNE or SCNE values of nonzero-sum objectives. Again, as discussed in Sections 4.5.2 and 5.3.2, for infinite-horizon objectives our use of value iteration means we cannot guarantee that the values computed are within a particular error bound of the actual values; so, the same will be true of the optimal strategy that we synthesise for such a formula.

In the case of a nonzero-sum state formula $\langle\langle C{:}C' \rangle\rangle_{\text{opt}\sim x}(\theta)$, we can return a subgame-perfect SWNE or SCNE for the objectives (X_1^θ, X_2^θ). This is achieved by:

- keeping track of the SWNE/SCNE profile for the bimatrix game solved in each state, comprising the distributions over actions for each player of $\mathsf{G}^{\mathcal{C}}$ in that state;
- generating an optimal strategy when solving the MDPs when solving them [141], which is equivalent to a strategy profile for $\mathsf{G}^{\mathcal{C}}$ (randomisation is not needed for this part).

We can then combine these generated profiles to yield a subgame-perfect SWNE or SCNE, switching from the strategy generated from the bimatrix games to the MDP strategies if we reach a state where the value of one player's objective cannot change. The synthesised strategies require randomisation and memory. Memory is needed since choices change after a path formulae becomes true or a target reached, and is also required for finite-horizon properties as separate player choices are stored for each state and each time step. For infinite-horizon properties, the use of value iteration means only approximate ε-NE profiles are synthesised. We illustrate the need for memory when synthesising equilibria strategies in Example 18.

Example 18. Let us consider the CSG illustrated in Figure 5.4 and the property $\langle\langle p_1{:}p_2 \rangle\rangle_{\max=?}(\mathtt{P}[\,\mathtt{F}\ \mathsf{goal}_1\,] + \mathtt{P}[\,\mathtt{F}\ \mathsf{goal}_2\,])$. It is easy to see that memory is required in order to synthesise a strategy that maximises the SWNE value. The shortest path that visits both s_2 and s_3 is $\pi = s_0 \xrightarrow{(a_1,\bot)} s_1 \xrightarrow{(b_1,b_2)} s_2 \xrightarrow{(c_1,\bot)} s_1 \xrightarrow{(b_1,a_2)} s_3$, for which the query would evaluate to the maximum possible value of $2-p$. One can see though that in order for such a path to be constructed, one needs to visit s_1 at least twice

and, after player p_1 has reached his goal by visiting state s_2, player p_2 has to change its choice from b_2 to a_2 in order for s_3 to be reached. If in state s_1 player p_2 decided to maximise for itself instead of considering the collective rewards, it would choose a_2 which would prevent player p_1 from ever reaching its goal and the query would evaluate to 1. On the other hand, if player 2 always picked b_2 in s_2, he would never reach his goal given that at some point s_4 would be reached and the game would end. In the latter case, the sum of probability values would also be 1.

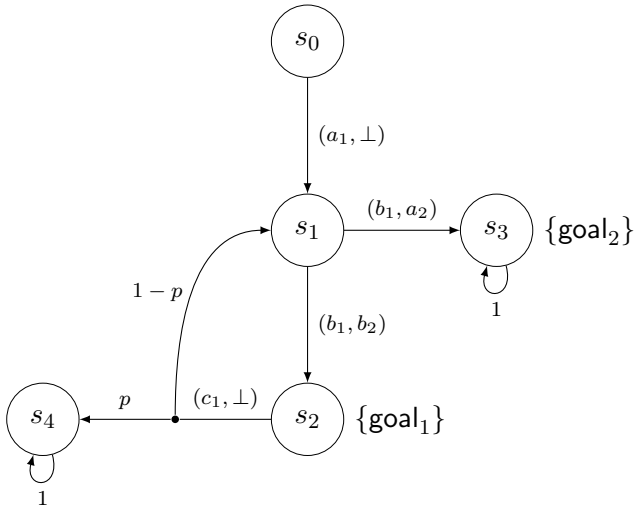


Figure 5.4: CSG example where memory may be required for synthesising an optimal strategy ($0 < p < 1.$).

■

5.5 Correctness and Complexity

The proof of correctness of our approach is given in Appendix A and shows that the values computed during value iteration correspond to subgame-perfect SWNE values of finite game trees, and the values of these game trees converge uniformly and are bounded from below and above by the finite approximations of $\mathsf{G}^{\mathcal{C}}$ and actual values of $\mathsf{G}^{\mathcal{C}}$, respectively. A limitation of our approach, as for standard value iteration [101, 127], is that convergence of the values does not give guarantees on the precision. Complexity is linear in the size of the formula. Value iteration requires solving an LCP problem of size $|A|$ for each state at every iteration, with the number of iterations depending on the convergence criterion. Papadimitriou established the complexity of solving the class of LCPs we encounter to be in PPAD (*polynomial parity argument in a directed graph*) [171] and, to the best of our knowledge, there is still no polynomial

algorithm for solving such problems. More closely related to finding all solutions, it has been shown that determining if there exists an equilibrium in a bimatrix game for which each player obtains a utility of a given bound is NP-complete [95]. Also, it is demonstrated in [9] that bimatrix games may have a number of NE that is exponential with respect to the size of the game, and thus any method that relies on finding all NE in the worst case cannot be expected to perform in a running time that is polynomial with respect to the size of the game. We present experimental results and discuss the efficiency of our approach in Chapters 7 and 8.

5.6 Summary

In this chapter, after reviewing classic game-theoretic concepts such as best response, Nash equilibrium, subgame-perfect equilibrium and social welfare equilibrium, we introduced the notions of least response and social cost equilibrium. We considered a method to compute equilibria values of bimatrix games and developed a procedure for achieving that by encoding it as an SMT problem. Subsequently, we extended rPATL to accommodate equilibria properties for two-player (coalitional) queries and introduced novel algorithms to compute/approximate SWNE and SCNE values for these properties over CSGs. rPATL includes the possibility of specifying properties that combine bounded and unbounded temporal operators, for which we also present a model transformation necessary to verify such properties.

We also considered the problem of synthesising strategies for equilibria properties, which require memory and randomisation, and discussed how to construct them by combining state-local SWNE or SWNE strategies and those resulting from MDP model checking. We concluded the chapter with a discussion of correctness and complexity, and the limitations of our approach.

CHAPTER 6

Multi-player Equilibria Properties

Contents

In this chapter we detail our approach for model checking and strategy synthesis of *multi-player* or *multi-coalitional* equilibria properties. Computation involving equilibria for multi-player games is a considerably more complex task than for two-player games, and different methods have to be employed both for solving the individual games at each state and to compute/approximate the values for the overall graph. A clear limitation of the approach described in Chapter 5 is the assumption that agents can, or would be willing to, cooperate, synchronise and form two distinct coalitions. Incorporating multi-coalitional properties into rPATL then allows for a wider scope of problems to be analysed. Naturally, the methods described in this chapter would then also cover two-player verification but are presented separately for a number of reasons:

- finding SWNE/SCNE values for games locally is no longer a linear problem;
- it is not possible to switch to MDP model checking when one player's objective cannot change;

- model checking algorithms are markedly more complex.

We start by formally presenting a well known result from game theory that we use to characterise equilibria strategy profiles for normal form games. Subsequently, we present our approach for computing NE values using nonlinear programming and *support enumeration*. We then extend rPATL to accommodate *multi-coalitional properties* and describe the algorithms used for verification and synthesis. A discussion of correctness and complexity concludes the chapter. The material presented in this chapter in based on [138].

6.1 Computing NE Values of n-Player Games

We start by introducing a lemma that is central to our approach for computing equilibria values for multi-player games. The lemma states that a profile is an NE if and only if any player switching to a single action in the support of the profile yields the same utility for the player and switching to an action outside the support cannot increase its utility. In the following, we refer to the support of a profile as the product of the supports of the individual strategies. Furthermore, a profile is said to have full support if it includes all available action tuples.

Lemma 3 ([174])**.** A strategy profile $\sigma{=}(\sigma_1, \ldots, \sigma_n)$ of an NFG N is an NE if and only if the following conditions are satisfied:

$$\forall i \in N. \forall a_i \in A_i. \quad \sigma_i(a_i) > 0 \rightarrow u_i(\sigma_{-i}(\eta_{a_i})) = u_i(\sigma) \tag{6.1}$$

$$\forall i \in N. \forall a_i \in A_i. \quad \sigma_i(a_i) = 0 \rightarrow u_i(\sigma_{-i}(\eta_{a_i})) \leqslant u_i(\sigma)\,. \tag{6.2}$$

Example 19. In order to illustrate how the conditions described in Lemma 3 characterise an NE, let us consider again the coalitional stag hunt game example originally shown in Examples 4 and 15 as the following bimatrix game:

$$\mathsf{Z}_1 = \begin{array}{c} \\ a_0 \\ a_1 \end{array} \begin{array}{c} \begin{array}{ccc} b_0 & b_1 & b_2 \end{array} \\ \begin{pmatrix} 2 & 2 & 2 \\ 0 & 4 & 6 \end{pmatrix} \end{array} \qquad \mathsf{Z}_2 = \begin{array}{c} \\ a_0 \\ a_1 \end{array} \begin{array}{c} \begin{array}{ccc} b_0 & b_1 & b_2 \end{array} \\ \begin{pmatrix} 4 & 2 & 0 \\ 4 & 6 & 9 \end{pmatrix} \end{array}$$

For a strategy profile σ we have the following payoffs for the players when they switch to deterministic strategies:

$$\begin{array}{rcl} u_1(\sigma_{-1}(\eta_{a_0})) & = & 2\sigma_2(b_0) + 2\sigma_2(b_1) + 2\sigma_2(b_2) \\ u_1(\sigma_{-1}(\eta_{a_1})) & = & 4\sigma_2(b_1) + 6\sigma_2(b_2) \\ u_2(\sigma_{-2}(\eta_{b_0})) & = & 4\sigma_1(a_0) + 4\sigma_1(a_1) \\ u_2(\sigma_{-2}(\eta_{b_1})) & = & 2\sigma_2(a_0) + 6\sigma_1(a_1) \\ u_2(\sigma_{-2}(\eta_{b_2})) & = & 9\sigma_1(a_2) \end{array}$$

If we consider the strategy profile $\sigma^{a_1,b_2} = (\delta_{a_1}, \delta_{b_2})$, then the payoff vector is $v = (6, 9)$ and the conditions of Lemma 3 are satisfied:

$$\begin{aligned} 2\sigma_2^{a_1,b_2}(b_0) + 2\sigma_2^{a_1,b_2}(b_1) + 2\sigma_2^{a_1,b_2}(b_2) &= 2 \leqslant v_1 \\ 4\sigma_2^{a_1,b_2}(b_1) + 6\sigma_2^{a_1,b_2}(b_2) &= 6 = v_1 \\ 4\sigma_1^{a_1,b_2}(a_0) + 4\sigma_1^{a_1,b_2}(a_1) &= 4 \leqslant v_2 \\ 2\sigma_1^{a_1,b_2}(a_0) + 6\sigma_1^{a_1,b_2}(a_1) &= 6 \leqslant v_2 \\ 9\sigma_1^{a_1,b_2}(a_1) &= 9 = v_2 \end{aligned}$$

On the other hand, if we consider the strategy profile $\sigma^{a_0,b_0} = (\delta_{a_0}, \delta_{b_0})$, then the payoff vector is $v = (2, 4)$ and the conditions of Lemma 3 are satisfied:

$$\begin{aligned} 2\sigma_2^{a_0,b_0}(b_0) + 2\sigma_2^{a_0,b_0}(b_1) + 2\sigma_2^{a_0,b_0}(b_2) &= 2 = v_1 \\ 4\sigma_2^{a_0,b_0}(b_1) + 6\sigma_2^{a_0,b_0}(b_2) &= 0 \leqslant v_1 \\ 4\sigma_1^{a_0,b_0}(a_0) + 4\sigma_1^{a_0,b_0}(a_1) &= 4 = v_2 \\ 2\sigma_1^{a_0,b_0}(a_0) + 6\sigma_1^{a_0,b_0}(a_1) &= 2 \leqslant v_2 \\ 9\sigma_1^{a_0,b_0}(a_1) &= 0 \leqslant v_2 \end{aligned}$$

Since both profiles satisfy the conditions of Lemma 3 it therefore follows that both are NE. It is also straightforward to check that $\sigma = (\sigma_1, \sigma_2) = ((5/9{\cdot}\eta_{a_0}{+}4/9{\cdot}\eta_{a_1}), (2/3{\cdot}\eta_{b_0}{+}1/3{\cdot}\eta_{b_2}))$ is also a NE with payoff vector $v = (2, 4)$, which then gives us the same set of strategy profiles that had been previously found in Example 15.

■

Computing SWNE Values of NFGs. Computing equilibria values for an n-player game is a complex task given that, differently from the two-player case, finding whether a given strategy profile is an equilibrium cannot be encoded as a linear programming problem. Computing SWNE values makes the problem even harder as it involves finding a strategy profile that maximises the sum of utilities. The algorithm for the two-player case presented in Section 5.1.1, based on labelled polytopes, starts by considering all the regions of the strategy profile space and then iteratively reduces the search space as positive probability assignments are found and added as restrictions on the space. The efficiency of this approach is reduced when analysing games with a large number of actions and when one or more players are indifferent, as the possible assignments resulting from action permutations need to be exhausted in order to compute SWNE values.

Going in the opposite direction, support enumeration [174] is a method for computing NE that exhaustively examines all sub-regions, i.e. supports, of the strategy profile space, one at a time, checking whether that sub-region contains an equilibrium. The computation of SWNE values for n-player games through support enumeration

can thus only be computed efficiently for games with limited number of actions, as the number of supports to be checked grows exponentially with respect to the number of actions. More precisely, the number of supports equals $\prod_{i=1}^{n}(2^{|A_i|}-1)$, out of which $\prod_{i=1}^{n}|A_i|$ correspond to pure strategy profiles. Therefore, computing SWNE values through support enumeration will only be efficient for games with a small number of actions.

We now show how, for a given support, computing SWNE profiles can be encoded as a *nonlinear programming problem*. Formally, given the support $B = B_1 \times \cdots \times B_n$ where $B_i \subseteq A_i$ and $B_i \neq \varnothing$ for all $i \in N$, we choose unique *pivot* actions[1] $b_i^p \in B_i$ for $i \in N$, the nonlinear programming problem is to minimise:

$$-\sum_{i\in N}\left(\sum_{b\in B} u_i(b)\cdot\left(\prod_{j\in N} p_{j,b_j}\right)\right) \tag{6.3}$$

subject to:

$$\sum_{c\in B_{-i}(b_i^p)} u_i(c)\cdot\left(\prod_{j\in N_{-i}} p_{j,c_j}\right) - \sum_{c\in B_{-i}(b_i)} u_i(c)\cdot\left(\prod_{j\in N_{-i}} p_{j,c_j}\right) = 0 \tag{6.4}$$

$$\sum_{c\in B_{-i}(b_i^p)} u_i(c)\cdot\left(\prod_{j\in N_{-i}} p_{j,c_j}\right) - \sum_{c\in B_{-i}(a_i)} u_i(c)\cdot\left(\prod_{j\in N_{-i}} p_{j,c_j}\right) \geqslant 0 \tag{6.5}$$

$$\sum_{b_i\in B_i} p_{i,b_i} = 1 \tag{6.6}$$

$$p_{i,b_i} > 0 \tag{6.7}$$

for all $i \in N$, $b_i \in B_i \backslash \{b_i^p\}$ and $a_i \in A_i \backslash B_i$ where $B_{-i}(c_i) = B_1 \times \cdots \times B_{i-1} \times \{c_i\} \times B_{i+1} \times \cdots \times B_n$ and $N_{-i} = N \backslash \{i\}$.

The variables $V_B = \{p_{i,b_i} \mid i \in N \wedge b_i \in B_i\}$ in the above program represent the probabilities players choose different actions, i.e. p_{i,b_i} is the probability of player $i \in N$ selecting action $b_i \in B_i$. The constraints (6.6) and (6.7) ensure that the probabilities of each player selecting actions sum to one and that the support of the corresponding profile equals B. The term in Equation (6.3) corresponds to the sum of the individual utilities of the players when they play according to the profile corresponding to the values assigned to the variables V_B. By minimising its negated value, we require the solution is social welfare optimal. The constraints (6.4) and (6.5) require that the solution corresponds to a NE as these constraints encode the constraints (6.1) and (6.2), respectively, of Lemma 3 for the corresponding profile. There is a difference from the constraints in Lemma 3 since in constraints (6.4) and (6.5) only pivot actions are compared with other actions in B_i and $A_i \backslash B_i$, respectively. However, this is sufficient as (6.4) requires all actions in the support to yield the same utility.

[1]For each $i \in N$ this can be any action in B_i.

a	u_1	u_2	u_3	a	u_1	u_2	u_3
(c_1, c_2, c_3)	7	7	7	(c_1, d_2, c_3)	3	9	3
(c_1, c_2, d_3)	3	3	9	(c_1, d_2, d_3)	0	5	5
a	u_1	u_2	u_3	a	u_1	u_2	u_3
(d_1, c_2, c_3)	9	3	3	(d_1, d_2, c_3)	5	5	0
(d_1, c_2, d_3)	5	0	5	(d_1, d_2, d_3)	1	1	1

Table 6.1: Utilities for an instance of a three-player prisoner's dilemma.

SMT solvers with nonlinear modules can be used for computing equilibria values of n-player NFGs, although they can be inefficient. Alternative approaches for solving such problems include *barrier* or *interior-point* methods [167].

Example 20. Let us consider an instance of three prisoners dilemma with payoffs described in Table 6.1. If we consider different supports $B_{-i}(a_i)$, $B_j = A_j$ for $i \neq j$, the payoff for player i when playing action $a_i \in A_i$ can be computed as:

$$
\begin{aligned}
u_1(B_{-1}(c_1)) &= 7p_{2,c_2}p_{3,c_3} + 3p_{2,c_2}p_{3,d_3} + 3p_{2,d_2}p_{3,c_3} \\
u_1(B_{-1}(d_1)) &= 9p_{2,c_2}p_{3,c_3} + 5p_{2,c_2}p_{3,d_3} + 5p_{2,d_2}p_{3,c_3} + p_{2,d_2}p_{3,d_3} \\
u_2(B_{-2}(c_2)) &= 7p_{1,c_1}p_{3,c_3} + 3p_{1,c_1}p_{3,d_3} + 3p_{1,d_1}p_{3,c_3} \\
u_2(B_{-2}(d_2)) &= 9p_{1,c_1}p_{3,c_3} + 5p_{1,c_1}p_{3,d_3} + 5p_{1,d_1}p_{3,c_3} + p_{1,d_1}p_{3,d_3} \\
u_3(B_{-3}(c_3)) &= 7p_{1,c_1}p_{2,c_2} + 3p_{1,c_1}p_{2,d_2} + 3p_{1,d_1}p_{2,c_2} \\
u_3(B_{-3}(d_3)) &= 9p_{1,c_1}p_{2,c_2} + 5p_{1,c_1}p_{2,d_2} + 5p_{1,d_1}p_{2,c_2} + p_{1,d_1}p_{2,d_2}
\end{aligned}
$$

For a *full* support B^{fs}, that is, one in which all players play all their actions with positive probability, if we let pivot actions b_i^p equal c_i for $i \in N, c_i \in A_i$, we would have the following nonlinear program:
Minimise:

$$-(u_1(B^{fs}) + u_2(B^{fs}) + u_3(B^{fs}))$$

subject to:

$$
\begin{aligned}
&u_1(B_{-1}^{fs}(c_1)) - u_1(B_{-1}^{fs}(d_1)) = 0 \quad && p_{1,c_1} + p_{1,d_1} = 1 \quad && p_{1,c_1} > 0, p_{1,d_1} > 0 \\
&u_2(B_{-2}^{fs}(c_2)) - u_2(B_{-2}^{fs}(d_2)) = 0 \quad && p_{2,c_2} + p_{2,d_2} = 1 \quad && p_{2,c_2} > 0, p_{2,d_2} > 0 \\
&u_3(B_{-3}^{fs}(c_3)) - u_3(B_{-3}^{fs}(d_3)) = 0 \quad && p_{3,c_3} + p_{3,d_3} = 1 \quad && p_{3,c_3} > 0, p_{3,d_3} > 0
\end{aligned}
$$

Finally, if we consider a *partial* support B^{ps}, wehre each player $i \in N$ only plays action d_i with positive probability, and thus d_i is the only possible pivot action for each player, we would have the following nonlinear program:
Minimise:

$$-(u_1(B^{ps}) + u_2(B^{ps}) + u_3(B^{ps}))$$

subject to:

$$\begin{array}{lll} u_1(B^{ps}_{-1}(d_1)) \geqslant 0 & p_{1,d_1} = 1 & p_{1,d_1} > 0 \\ u_2(B^{ps}_{-2}(d_2)) \geqslant 0 & p_{2,d_2} = 1 & p_{2,d_2} > 0 \\ u_3(B^{ps}_{-3}(d_3)) \geqslant 0 & p_{3,d_3} = 1 & p_{3,d_3} > 0 \end{array}$$

which reduces to:

$$\begin{array}{lll} p_{2,d_2}p_{3,d_3} \geqslant 0 & p_{1,d_1} = 1 & p_{1,d_1} > 0 \\ p_{1,d_1}p_{3,d_3} \geqslant 0 & p_{2,d_2} = 1 & p_{2,d_2} > 0 \\ p_{1,d_1}p_{2,d_2} \geqslant 0 & p_{3,d_3} = 1 & p_{3,d_3} > 0 \end{array}$$

It is then straightforward to verify that the support B^{ps} is indeed a satisfying assignment, whereas there can be no equilibria with full support, as the constraints could be reduced to $p_{3,c_3}(p_{2,d_2} + 1) = -1$, which cannot be satisfied. As with the two-player case, defection dominates cooperation for all players, which leads to the only equilibrium being the one where all players defect.

■

6.2 Property Specification

We now consider the logic rPATL with Nash formulae and enhance it with equilibria properties that separate players into more than two coalitions. Since we are limited to considering ε-SWNE and ε-SCNE for infinite-horizon properties, we assume some ε has been fixed in advance.

Definition 6.2.1 (Extended rPATL syntax for multi-coalitional properties)**.** The syntax of our extended version of rPATL is given by the grammar:

$$\begin{array}{lcl} \phi & \coloneqq & \mathtt{true} \mid \mathtt{a} \mid \neg\phi \mid \phi \wedge \phi \mid \langle\!\langle C \rangle\!\rangle \mathtt{P}_{\sim q}[\,\psi\,] \mid \langle\!\langle C \rangle\!\rangle \mathtt{R}^{r}_{\sim x}[\,\rho\,] \mid \langle\!\langle C_1 {:}\cdots{:} C_m \rangle\!\rangle_{\mathrm{opt}\sim x}(\theta) \\ \theta & \coloneqq & \mathtt{P}[\,\psi\,]{+}\cdots{+}\mathtt{P}[\,\psi\,] \;\mid\; \mathtt{R}^{r}[\,\rho\,]{+}\cdots{+}\mathtt{R}^{r}[\,\rho\,] \\ \psi & \coloneqq & \mathtt{X}\,\phi \;\mid\; \phi\ \mathtt{U}^{\leqslant k}\ \phi \;\mid\; \phi\ \mathtt{U}\ \phi \\ \rho & \coloneqq & \mathtt{I}^{=k} \;\mid\; \mathtt{C}^{\leqslant k} \;\mid\; \mathtt{F}\ \phi \end{array}$$

where $\mathtt{a}$ is an atomic proposition, C and $C_1, \ldots, C_m$ are coalitions of players such that $C_i \cap C_j = \varnothing$ for all $1 \leqslant i \neq j \leqslant m$ and $\cup_{i=1}^{m} C_i = N$, $\mathrm{opt} \in \{\min, \max\}$, $\sim \in \{<, \leqslant, \geqslant, >\}$, $q \in \mathbb{Q} \cap [0,1]$, $x \in \mathbb{Q}$, r is a reward structure and $k \in \mathbb{N}$.

Our addition to the logic is *Nash formulae* of the form $\langle\!\langle C_1 {:}\cdots{:} C_m \rangle\!\rangle_{\mathrm{opt}\sim x}(\theta)$, where the *nonzero sum* formula θ comprises a sum of m probability or reward objectives. The formula $\langle\!\langle C_1 {:}\cdots{:} C_m \rangle\!\rangle_{\max\sim x}(\mathtt{P}[\,\psi_1\,]{+}\cdots{+}\mathtt{P}[\,\psi_m\,])$ holds in a state if, when the players form the coalitions $C_1, \ldots, C_m$, there is a subgame-perfect SWNE for which the *sum* of the values of the objectives $\mathtt{P}[\,\psi_1\,], \ldots, \mathtt{P}[\,\psi_m\,]$ for the coalitions $C_1, \ldots, C_m$

satisfies $\sim x$. The case for reward objectives is similar and, for formulae of the form $\langle\langle C_1{:}\cdots{:}C_m\rangle\rangle_{\min\sim x}(\theta)$, we require the existence of an SCNE rather than an SWNE. We also allow *numerical* queries of the form $\langle\langle C_1{:}\cdots{:}C_m\rangle\rangle_{\text{opt}=?}(\theta)$, which return the sum of the SWNE or SCNE values. Similarly to the two-player case, the rPATL syntax for multi-player nonzero-sum formulae presented here differs from that for zero-sum formulae presented in Definition 4.4 by dropping the reward operators for *expected reachability* ($\mathtt{F}^c$) and *total rewards* ($\mathtt{C}$) as currently no algorithms are known for these variants. As for two-player equilibria formulae, multi-coalitional rPATL includes *infinite-horizon* ($\mathtt{U}$, $\mathtt{F}$) temporal operators for which the existence of SWNE and SCNE is open. However, ε-SWNE and ε-SCNE *do* exist for any $\varepsilon > 0$. Hence, we define the semantics of the logic in the context of a particular ε.

Definition 6.2.2 (Extended rPATL semantics for Multi-coalitional Properties)**.** The satisfaction relation $\models$ of our rPATL extension is defined inductively on the structure of the formula. The propositional logic fragment ($\mathtt{true}$, $\mathtt{a}$, $\neg$, $\wedge$) is defined in the usual way. The zero-sum formulae $\langle\langle C\rangle\rangle\mathtt{P}_{\sim q}[\,\psi\,]$ and $\langle\langle C\rangle\rangle\mathtt{R}^r_{\sim x}[\,\rho\,]$ are defined as in Chapter 4. For a Nash formula and state $s \in S$ in CSG G, we have:

$$s \models \langle\langle C_1{:}\cdots{:}C_m\rangle\rangle_{\text{opt}\sim x}(\theta) \;\Leftrightarrow\; \exists\sigma^\star \in \Sigma_{\mathsf{G}^\mathcal{C}}.\ \left(\mathbb{E}^{\sigma^\star}_{\mathsf{G}^\mathcal{C},s}(X^\theta_1) + \cdots + \mathbb{E}^{\sigma^\star}_{\mathsf{G}^\mathcal{C},s}(X^\theta_m)\right) \sim x$$

and $\sigma^\star = (\sigma^\star_1, \ldots, \sigma^\star_m)$ is a subgame-perfect ε-SWNE if $\text{opt} = \max$ and a subgame-perfect ε-SCNE if $\text{opt} = \min$ for the objectives $(X^\theta_1, \ldots, X^\theta_m)$ in game $\mathsf{G}^\mathcal{C}$, where $\mathcal{C} = \{C_1, \ldots, C_m\}$ and for $1 \leqslant i \leqslant m$ and $\pi \in \mathit{IPaths}^{\sigma^\star}_{\mathsf{G}^\mathcal{C},s}$:

$$\begin{aligned} X_i^{\mathtt{P}[\psi^1]+\cdots+\mathtt{P}[\psi^m]}(\pi) &= 1 \text{ if } \pi \models \psi^i \text{ and } 0 \text{ otherwise} \\ X_i^{\mathtt{R}^{r_1}[\rho^1]+\cdots+\mathtt{R}^{r_m}[\rho^m]}(\pi) &= \mathit{rew}(r_i, \rho^i)(\pi) \end{aligned}$$

Example 21. Let us consider again the robot coordination problem from Example 8. The following are properties are examples of multi-player equilibria formulae that could be expressed with rPATL (where $\mathtt{g}_i$ and $\mathtt{c}$ are atomic propositions indicating the goal position of robot_1 and a collision, respectively):

- $\langle\langle \mathit{robot}_1{:}\mathit{robot}_2{:}\mathit{robot}_3\rangle\rangle_{\max\geqslant 3}(\mathtt{P}[\,\neg\mathtt{c}\ \mathtt{U}\ \mathtt{g}_1\,]+\mathtt{P}[\,\neg\mathtt{c}\ \mathtt{U}\ \mathtt{g}_2\,]+\mathtt{P}[\,\neg\mathtt{c}\ \mathtt{U}\ \mathtt{g}_3\,])$ could be used to check whether the robots can collaborate so that all reach their goal with probability 1 without crashing;

- $\langle\langle \mathit{robot}_1{:}\mathit{robot}_2{:}\mathit{robot}_3\rangle\rangle_{\min=?}(\mathtt{R}^{r_{steps}}[\,\mathtt{F}\ \mathtt{g}_1\,]+\mathtt{R}^{r_{steps}}[\,\mathtt{F}\ \mathtt{g}_2\,]+\mathtt{R}^{r_{steps}}[\,\mathtt{F}\ \mathtt{g}_3\,])$ asks what is the sum of expected values when the robots collaborate and each wants to minimise their expected steps to reach their goal (r_{steps} is a reward structure that assigns 1 to all state action tuple pairs);

- $\langle\langle robot_1{:}robot_2{:}robot_3\rangle\rangle_{\max=?}(\mathtt{P}[\,\neg\mathtt{c}\ \mathtt{U}^{\leqslant k}\ \mathtt{g}_1\,]{+}\mathtt{P}[\,\neg\mathtt{c}\ \mathtt{U}^{\leqslant k}\ \mathtt{g}_2\,]{+}\mathtt{P}[\,\neg\mathtt{c}\ \mathtt{U}^{\leqslant k}\ \mathtt{g}_3\,])$ can be used to compute the sum of social welfare probability values so that the robots reach their goal without crashing within k steps.

■

6.3 Model Checking Multi-player Nonzero-sum Properties

The logic rPATL is branching-time and the model checking algorithm works by recursively computing the set $Sat(\phi)$ of states satisfying formula ϕ over the structure of ϕ. Therefore, to extend the existing approach of Chapters 4 and 5, we need only consider formulae of the form $\langle\langle C_1{:}\cdots{:}C_m\rangle\rangle_{\mathrm{opt}\sim x}(\theta)$. From Definition 6.2.2, this requires the computation of subgame-perfect SWNE or SCNE values of the objectives $(X_1^\theta,\ldots,X_m^\theta)$ and a comparison of their sum to the threshold x.

As for the two-player case reported in Chapter 5, in order to check nonzero-sum properties on CSGs, we have to work with a restricted class of games. As previously done, we use an objective-dependent assumption, which ensures that, under all profiles, with probability 1, eventually the outcome of each player's objective does not change by continuing. This can be checked using graph algorithms [72]. We require the following assumptions on CSGs for the different classes of multi-coalitional equilibria formulae.

Assumption 5. For subformulae $\mathtt{P}[\,\phi_1^i\ \mathtt{U}\ \phi_2^i\,]$, there are no end components of $\mathtt{G}$ that are subsets of $Sat(\neg\phi_1^i \vee \neg\phi_2^i)$.

Assumption 6. For subformulae $\mathtt{R}^r[\,\mathtt{F}\ \phi^i\,]$, the set $Sat(\phi^i)$ is reached with probability 1 from all states of $\mathtt{G}$ under all profiles.

If all the objectives in the nonzero sum formula θ are finite-horizon, backward induction can be applied to compute (precise) subgame-perfect SWNE values. On the other hand, when all the objectives are infinite-horizon, we extend the techniques of Chapter 5 for two-player games and use value iteration to approximate subgame-perfect SWNE values. The extension is non-trivial, as in the two-player case, when one player reaches their goal, we can apply MDP verification techniques, while this is no longer possible when there is more than one player that has not reached their goal. In addition, in such cases we cannot reduce the analysis from an n-player to a $(n{-}1)$-player game, as the choices of the player that has reached its goal can still

influence the outcomes of the remaining players and assigning its choices to one of the other players can exploit these choices to its own advantage. Instead, we need to keep track of the set of players that have reached their goal (denoted D) and can no longer reach their goal in the case of until formulae (denoted E), and define the values at each iteration using these sets. In cases when there is a combination of finite- and infinite-horizon objectives, we can extend the techniques of Section 5.3.5 and make all objectives infinite-horizon by modifying the game in a standard manner.

We now show how to compute the SWNE and SNCE values of a multi-coalitional formula $\langle\langle C{:}\cdots{:}C_m\rangle\rangle_{\mathrm{opt}\sim x}(\theta)$. For the remainder of this section we fix a NFG N and CSG G. Again, we use the notations $\mathtt{V}_{\mathsf{G}^{\mathcal{C}}}(s,\theta)$, $\mathtt{V}_{\mathsf{G}^{\mathcal{C}}}(s,\theta,n)$ and $\overline{\mathtt{V}}_{\mathsf{G}^{\mathcal{C}}}(s,\theta)$,$\overline{\mathtt{V}}_{\mathsf{G}^{\mathcal{C}}}(s,\theta,n)$ for the vector of SWNE and SCNE values of the objectives $(X_1^\theta, X_2^\theta, \ldots, X_N^\theta)$ in state s of $\mathsf{G}^{\mathcal{C}}$. We also use $\mathbf{1}_N$ and $\mathbf{0}_N$ to denote a vector of size N whose entries all equal to 1 or 0, respectively. For any set of states S' and state s we let $\eta_{S'}(s)$ equal 1 if $s \in S'$ and 0 otherwise. Furthermore, to simplify the presentation the step bounds appearing in path and reward formulae can take negative values. We refrain from showing any combination of objectives for multi-player equilibria properties for a number or reasons: (i) the added intricacy of keeping track of players who cannot satisfy their objectives due to an elapsed time bound; (ii) accounting for the possible permutations of operators inside any given formula; (iii) the reasoning behind developing the algorithms for dealing with these cases is not dissimilar to that applied for the two-player case (see Sections 5.3.1 and 5.3.5), and the problem could be solved by applying appropriate model transformations.

6.3.1 Computing SWNE Values of Finite-horizon Nonzero-sum Formulae

As for the two-player case, for a *finite-horizon* formula θ, we compute the SWNE values $\mathtt{V}_{\mathsf{G}^{\mathcal{C}}}(s,\theta)$ for all states s of the coalitional game $\mathsf{G}^{\mathcal{C}}$ in a recursive fashion based on the step bound. We now solve normal form games at each step, which are defined in a similar manner to the bimatrix games for two-player nonzero-sum properties: the actions of each normal form game correspond to the actions available in that state and the utilities are constructed from the transition probabilities $\delta^{\mathcal{C}}$ of the game $\mathsf{G}^{\mathcal{C}}$, the reward structure (in the case of reward formulae) and the values already computed recursively for successor states. Below, we explain the computation for both types of finite-horizon probabilistic objectives (next and bounded until) and reward objectives.

Next. If $\theta = \mathtt{P}[\,\mathtt{X}\,\phi^1\,]+\cdots+\mathtt{P}[\,\mathtt{X}\,\phi^m\,]$, then $\mathtt{V}_{\mathsf{G}^{\mathcal{C}}}(s,\theta)$ equals SWNE values of the game $\mathsf{N} = (M, A^{\mathcal{C}}, u)$ in which for any $1 \leqslant l \leqslant m$ and $a \in A^{\mathcal{C}}$:

$$u_l(a) = \sum_{s' \in S} \delta^{\mathcal{C}}(s,a)(s') \cdot \eta_{Sat(\phi^l)}(s')$$

Bounded Until. If $\theta = \mathtt{P}[\,\phi_1^1\ \mathtt{U}^{k_1}\ \phi_2^1\,]+\cdots+\mathtt{P}[\,\phi_1^m\ \mathtt{U}^{\leqslant k_m}\ \phi_2^{\leqslant m}\,]$, we compute SWNE values of the objectives recursively for $0 \leqslant n \leqslant k$ of the nonzero-sum formulae $\theta_n = \mathtt{P}[\,\phi_1^1\ \mathtt{U}^{\leqslant k_1 - n}\ \phi_2^1\,]+\cdots+\mathtt{P}[\,\phi_1^m\ \mathtt{U}^{\leqslant k_m - n}\ \phi_2^m\,]$, where $k = \max\{k_1,\ldots,k_l\}$ and $\mathtt{V}_{\mathsf{G}^{\mathcal{C}}}(s,\theta) = \mathtt{V}_{\mathsf{G}^{\mathcal{C}}}(s,\varnothing,\varnothing,\theta_0)$. For any state s and $0 \leqslant n \leqslant k$, $D, E \subseteq M$ such that $D \cap E = \varnothing$:

$$\mathtt{V}_{\mathsf{G}^{\mathcal{C}}}(s,D,E,\theta_n) = \begin{cases} (\eta_D(1),\ldots,\eta_D(m)) & \text{if } D \cup E = M \\ \mathtt{V}_{\mathsf{G}^{\mathcal{C}}}(s, D \cup D', E, \theta_n) & \text{else if } D' \neq \varnothing \\ \mathtt{V}_{\mathsf{G}^{\mathcal{C}}}(s, D, E \cup E', \theta_n) & \text{else if } E' \neq \varnothing \\ val(\mathsf{N}) & \text{otherwise} \end{cases}$$

where $D' = \{l \in M \backslash (D \cup E) \mid s \in Sat(\phi_2^l)\}$, $E' = \{l \in M \backslash (D \cup E) \mid s \in Sat(\neg\phi_1^l \wedge \neg\phi_2^l)\}$ and $val(\mathsf{N})$ equals SWNE values of the game $\mathsf{N} = (M, A^{\mathcal{C}}, u)$ in which for any $1 \leqslant l \leqslant m$ and $a \in A^{\mathcal{C}}$:

$$u_l(a) = \begin{cases} 1 & \text{if } l \in D \\ 0 & \text{else if } l \in E \\ 0 & \text{else if } n_l - n \leqslant 0 \\ \sum_{s' \in S} \delta^{\mathcal{C}}(s,a)(s') \cdot v_{n-1}^{s',l} & \text{otherwise} \end{cases}$$

and $(v_{n-1}^{s',1}, v_{n-1}^{s',2}, \ldots, v_{n-1}^{s',m}) = \mathtt{V}_{\mathsf{G}^{\mathcal{C}}}(s', D, E, \theta_{n-1})$ for all $s' \in S$.

Instantaneous Rewards. If $\theta = \mathtt{R}^{r_1}[\,\mathtt{I}^{=k_1}\,]+\cdots+\mathtt{R}^{r_m}[\,\mathtt{I}^{=k_m}\,]$, we compute SWNE values of the objectives for the nonzero-sum formulae $\theta_n = \mathtt{R}^{r_1}[\,\mathtt{I}^{=n_1-n}\,]+\cdots+\mathtt{R}^{r_m}[\,\mathtt{I}^{=n_l-n}\,]$ for $0 \leqslant n \leqslant k$ recursively, where $k = \max\{k_1,\ldots,k_l\}$ and $\mathtt{V}_{\mathsf{G}^{\mathcal{C}}}(s,\theta) = \mathtt{V}_{\mathsf{G}^{\mathcal{C}}}(s,\theta_0)$. For any state s and $0 \leqslant n \leqslant k$, $\mathtt{V}_{\mathsf{G}^{\mathcal{C}}}(s,\theta_n)$ equals SWNE values of the game $\mathsf{N} = (M, A^{\mathcal{C}}, u)$ in which for any $1 \leqslant l \leqslant m$ and $a \in A^{\mathcal{C}}$:

$$u_l(a) = \begin{cases} 0 & \text{if } n_l - n < 0 \\ \sum_{s' \in S} \delta^{\mathcal{C}}(s,a)(s') \cdot r_S^l(s') & \text{else if } n_l - n = 0 \\ \sum_{s' \in S} \delta^{\mathcal{C}}(s,a)(s') \cdot v_{n+1}^{s',l} & \text{otherwise} \end{cases}$$

and $(v_{n+1}^{s',1}, \ldots, v_{n+1}^{s',m}) = \mathtt{V}_{\mathsf{G}^{\mathcal{C}}}(s', \theta_{n+1})$ for all $s' \in S$.

Bounded Cumulative Rewards. If $\theta = \mathtt{R}^{r_1}[\,\mathtt{C}^{\leqslant k_1}\,] + \cdots + \mathtt{R}^{r_m}[\,\mathtt{C}^{\leqslant k_m}\,]$, we compute SWNE values of the objectives for the nonzero-sum formulae $\theta_n = \mathtt{R}^{r_1}[\,\mathtt{C}^{\leqslant n_1 - n}\,] + \cdots + \mathtt{R}^{r_l}[\,\mathtt{C}^{\leqslant n_m - n}\,]$ for $0 \leqslant n \leqslant k$ recursively, where $k = \max\{k_1,\ldots,k_l\}$ and $\mathtt{V}_{\mathsf{G}^{\mathcal{C}}}(s,\theta)$

$= \mathtt{V}_{\mathsf{G}^{\mathcal{C}}}(s, \theta_0)$. For any state s and $0 \leqslant n \leqslant k$, $\mathtt{V}_{\mathsf{G}^{\mathcal{C}}}(s, \theta_n)$ equals SWNE values of the game $\mathsf{N} = (M, A^{\mathcal{C}}, u)$ in which for any $1 \leqslant l \leqslant m$ and $a \in A^{\mathcal{C}}$:

$$u_l(a) = \begin{cases} 0 & \text{if } n_l - n \leqslant 0 \\ r_S^l(s) + r_A^l(s,a) + \sum_{s' \in S} \delta^{\mathcal{C}}(s,a)(s') \cdot v_{n+1}^{s',l} & \text{otherwise} \end{cases}$$

and $(v_{n+1}^{s',1}, \ldots, v_{n+1}^{s',m}) = \mathtt{V}_{\mathsf{G}^{\mathcal{C}}}(s', \theta_{n+1})$ for all $s' \in S$.

6.3.2 Computing SWNE Values of Infinite-horizon Nonzero-sum Formulae

We next show how to compute SWNE values $\mathtt{V}_{\mathsf{G}^{\mathcal{C}}}(s, \theta)$ for *infinite-horizon* multi-player formulae θ in all states s of $\mathsf{G}^{\mathcal{C}}$. As for the two-player case, we approximate these using a value iteration approach. Each step of this computation is similar in nature to the algorithms in the previous section, where a normal form is solved for each state, and a different game is played in states where a given set of players have satisfied or can no longer satisfy their goals.

Again, a key aspect of the value iteration algorithm is based on the fact that convergence is defined over the *sum* of equilibria values for each player. Convergence of value iteration is estimated in the same way as for the zero-sum computation (see Section 5.3.2), by comparing values in successive iterations. As previously, this means that we are not able to guarantee that the computed values are within a particular error bound of the exact values.

Until. For formulae of the type $\theta = \mathtt{P}[\,\phi_1^1 \;\mathtt{U}\; \phi_2^1\,] + \cdots + \mathtt{P}[\,\phi_1^m \;\mathtt{U}\; \phi_2^m\,]$, values can be computed through value iteration as the limit $\mathtt{V}_{\mathsf{G}^{\mathcal{C}}}(s, \theta) = \lim_{n \to \infty} \mathtt{V}_{\mathsf{G}^{\mathcal{C}}}(s, \theta, n)$ where $\mathtt{V}_{\mathsf{G}^{\mathcal{C}}}(s, \theta, n) = \mathtt{V}_{\mathsf{G}^{\mathcal{C}}}(s, \varnothing, \varnothing, \theta, n)$ and for any $D, E \subseteq M$ such that $D \cap E = \varnothing$:

$$\mathtt{V}_{\mathsf{G}^{\mathcal{C}}}(s, D, E, \theta, n) = \begin{cases} (\eta_D(1), \ldots, \eta_D(m)) & \text{if } D \cup E = M \\ (\eta_{Sat(\phi_2^1)}(s), \ldots, \eta_{Sat(\phi_2^m)}(s)) & \text{else if } n = 0 \\ \mathtt{V}_{\mathsf{G}^{\mathcal{C}}}(s, D \cup D', E, \theta, n) & \text{else if } D' \neq \varnothing \\ \mathtt{V}_{\mathsf{G}^{\mathcal{C}}}(s, D, E \cup E', \theta, n) & \text{else if } E' \neq \varnothing \\ val(\mathsf{N}) & \text{otherwise} \end{cases}$$

where $D' = \{l \in M \backslash (D \cup E) \mid s \in Sat(\phi_2^l)\}$, $E' = \{l \in M \backslash (D \cup E) \mid s \in Sat(\neg\phi_1^l \wedge \neg\phi_2^l)\}$ and $val(\mathsf{N})$ equals SWNE values of the game $\mathsf{N} = (M, A^{\mathcal{C}}, u)$ in which for any $1 \leqslant l \leqslant m$ and $a \in A^{\mathcal{C}}$:

$$u_l(a) = \begin{cases} 1 & \text{if } l \in D \\ 0 & \text{else if } l \in E \\ \sum_{s' \in S} \delta^{\mathcal{C}}(s,a)(s') \cdot v_{n-1}^{s',l} & \text{otherwise} \end{cases}$$

and $(v_{n-1}^{s',1}, v_{n-1}^{s',2}, \ldots, v_{n-1}^{s',m}) = \mathtt{V}_{\mathsf{G}^{\mathcal{C}}}(s', D, E, \theta, n-1)$ for all $s' \in S$.

Expected Reachability. If $\theta = \mathtt{R}^{r_1}[\,\mathtt{F}\ \phi^1\,] + \cdots + \mathtt{R}^{r_m}[\,\mathtt{F}\ \phi^m\,]$, values can be computed through value iteration as the limit $\mathtt{V}_{\mathsf{G}^{\mathcal{C}}}(s,\theta) = \lim_{n\to\infty} \mathtt{V}_{\mathsf{G}^{\mathcal{C}}}(s,\theta,n)$ where $\mathtt{V}_{\mathsf{G}^{\mathcal{C}}}(s,\theta,n) = \mathtt{V}_{\mathsf{G}^{\mathcal{C}}}(s,\varnothing,\theta,n)$ and for any $D \subseteq M$:

$$\mathtt{V}_{\mathsf{G}^{\mathcal{C}}}(s,D,\theta,n) = \begin{cases} \mathbf{0}_m & \text{if } D = M \\ \mathbf{0}_m & \text{else if } n = 0 \\ \mathtt{V}_{\mathsf{G}^{\mathcal{C}}}(s, D \cup D', \theta, n) & \text{else if } D' \neq \varnothing \\ val(\mathsf{N}) & \text{otherwise} \end{cases}$$

$D' = \{l \in M \backslash D \mid s \in Sat(\phi^l)\}$ and $val(\mathsf{N})$ equals SWNE values of the game $\mathsf{N} = (M, A^{\mathcal{C}}, u)$ in which for any $1 \leqslant l \leqslant m$ and $a \in A^{\mathcal{C}}$:

$$u_l(a) = \begin{cases} 0 & \text{if } l \in D \\ r_S^l(s) + r_A^l(s,a) + \sum_{s' \in S} \delta^{\mathcal{C}}(s,a)(s') \cdot v_{n-1}^{s',l} & \text{otherwise} \end{cases}$$

and $(v_{n-1}^{s',1}, v_{n-1}^{s',2}, \ldots, v_{n-1}^{s',m}) = \mathtt{V}_{\mathsf{G}^{\mathcal{C}}}(s', D, \theta, n-1)$ for all $s' \in S$.

6.3.3 Computing SCNE Values of Finite-horizon Nonzero-sum Formulae

In this section, we show how to compute SCNE values $\overline{\mathtt{V}}_{\mathsf{G}^{\mathcal{C}}}(s,\theta)$ for *finite-horizon* multi-player formula θ in all states s of $\mathsf{G}^{\mathcal{C}}$. Similarly the algorithms of Section 5.3.4, this is achieved by negating the utility function values of the normal form games representing each state, and subsequently negating the SWNE values computed.

Next. If $\theta = \mathtt{P}[\,\mathtt{X}\,\phi^1\,] + \cdots + \mathtt{P}[\,\mathtt{X}\,\phi^m\,]$, then $\overline{\mathtt{V}}_{\mathsf{G}^{\mathcal{C}}}(s,\theta)$ equals the negation of the SWNE values of the game $\mathsf{N} = (M, A^{\mathcal{C}}, u)$ in which for any $1 \leqslant l \leqslant m$ and $a \in A^{\mathcal{C}}$:

$$u_l(a) = -\sum_{s' \in S} \delta^{\mathcal{C}}(s,a)(s') \cdot \eta_{Sat(\phi^l)}(s')$$

Bounded Until. If $\theta = \mathtt{P}[\,\phi_1^1\ \mathtt{U}^{k_1}\ \phi_2^1\,] + \cdots + \mathtt{P}[\,\phi_1^m\ \mathtt{U}^{\leqslant k_m}\ \phi_2^{\leqslant m}\,]$, we compute SCNE values of the objectives recursively for $0 \leqslant n \leqslant k$ of the nonzero-sum formulae $\theta_n = \mathtt{P}[\,\phi_1^1\ \mathtt{U}^{\leqslant k_1 - n}\ \phi_2^1\,] + \cdots + \mathtt{P}[\,\phi_1^m\ \mathtt{U}^{\leqslant k_m - n}\ \phi_2^m\,]$, where $k = \max\{k_1, \ldots, k_l\}$ and $\overline{\mathtt{V}}_{\mathsf{G}^{\mathcal{C}}}(s,\theta) = \overline{\mathtt{V}}_{\mathsf{G}^{\mathcal{C}}}(s,\varnothing,\varnothing,\theta_0)$. For any state s and $0 \leqslant n \leqslant k$, $D, E \subseteq M$ such that $D \cap E = \varnothing$:

$$\overline{\mathtt{V}}_{\mathsf{G}^{\mathcal{C}}}(s,D,E,\theta_n) = \begin{cases} (\eta_D(1), \ldots, \eta_D(m)) & \text{if } D \cup E = M \\ \overline{\mathtt{V}}_{\mathsf{G}^{\mathcal{C}}}(s, D \cup D', E, \theta_n) & \text{else if } D' \neq \varnothing \\ \overline{\mathtt{V}}_{\mathsf{G}^{\mathcal{C}}}(s, D, E \cup E', \theta_n) & \text{else if } E' \neq \varnothing \\ -val(\mathsf{N}) & \text{otherwise} \end{cases}$$

where $D' = \{l \in M \backslash (D \cup E) \mid s \in Sat(\phi_2^l)\}$, $E' = \{l \in M \backslash (D \cup E) \mid s \in Sat(\neg\phi_1^l \wedge \neg\phi_2^l)\}$ and $val(\mathsf{N})$ equals the SWNE values of the game $\mathsf{N} = (M, A^{\mathcal{C}}, u)$ in which for any $1 \leqslant l \leqslant m$ and $a \in A^{\mathcal{C}}$:

$$u_l(a) = -1 \times \begin{cases} 1 & \text{if } l \in D \\ 0 & \text{else if } l \in E \\ 0 & \text{else if } n_l - n \leqslant 0 \\ \sum_{s' \in S} \delta^{\mathcal{C}}(s,a)(s') \cdot v_{n-1}^{s',l} & \text{otherwise} \end{cases}$$

and $(v_{n-1}^{s',1}, v_{n-1}^{s',2}, \ldots, v_{n-1}^{s',m}) = \overline{\mathtt{V}}_{\mathsf{G}^{\mathcal{C}}}(s', D, E, \theta_{n-1})$ for all $s' \in S$.

Instantaneous Rewards. If $\theta = \mathtt{R}^{r_1}[\,\mathtt{I}^{=k_1}\,] + \cdots + \mathtt{R}^{r_m}[\,\mathtt{I}^{=k_m}\,]$, we compute SCNE values of the objectives for the nonzero-sum formulae $\theta_n = \mathtt{R}^{r_1}[\,\mathtt{I}^{=n_1-n}\,] + \cdots + \mathtt{R}^{r_m}[\,\mathtt{I}^{=n_l-n}\,]$ for $0 \leqslant n \leqslant k$ recursively, where $k = \max\{k_1, \ldots, k_l\}$ and $\overline{\mathtt{V}}_{\mathsf{G}^{\mathcal{C}}}(s, \theta) = \overline{\mathtt{V}}_{\mathsf{G}^{\mathcal{C}}}(s, \theta_0)$. For any state s and $0 \leqslant n \leqslant k$, $\overline{\mathtt{V}}_{\mathsf{G}^{\mathcal{C}}}(s, \theta_n)$ equals the negation of the SWNE values of the game $\mathsf{N} = (M, A^{\mathcal{C}}, u)$ in which for any $1 \leqslant l \leqslant m$ and $a \in A^{\mathcal{C}}$:

$$u_l(a) = -1 \times \begin{cases} 0 & \text{if } n_l - n < 0 \\ \sum_{s' \in S} \delta^{\mathcal{C}}(s,a)(s') \cdot r_S^l(s') & \text{else if } n_l - n = 0 \\ \sum_{s' \in S} \delta^{\mathcal{C}}(s,a)(s') \cdot v_{n+1}^{s',l} & \text{otherwise} \end{cases}$$

and $(v_{n+1}^{s',1}, \ldots, v_{n+1}^{s',m}) = \overline{\mathtt{V}}_{\mathsf{G}^{\mathcal{C}}}(s', \theta_{n+1})$ for all $s' \in S$.

Bounded Cumulative Rewards. If $\theta = \mathtt{R}^{r_1}[\,\mathtt{C}^{\leqslant k_1}\,] + \cdots + \mathtt{R}^{r_m}[\,\mathtt{C}^{\leqslant k_m}\,]$, we compute SCNE values of the objectives for the nonzero-sum formulae $\theta_n = \mathtt{R}^{r_1}[\,\mathtt{C}^{\leqslant n_1-n}\,] + \cdots + \mathtt{R}^{r_l}[\,\mathtt{C}^{\leqslant n_m-n}\,]$ for $0 \leqslant n \leqslant k$ recursively, where $k = \max\{k_1, \ldots, k_l\}$ and $\overline{\mathtt{V}}_{\mathsf{G}^{\mathcal{C}}}(s, \theta) = \overline{\mathtt{V}}_{\mathsf{G}^{\mathcal{C}}}(s, \theta_0)$. For any state s and $0 \leqslant n \leqslant k$, $\overline{\mathtt{V}}_{\mathsf{G}^{\mathcal{C}}}(s, \theta_n)$ equals the negation of the SWNE values of the game $\mathsf{N} = (M, A^{\mathcal{C}}, u)$ in which for any $1 \leqslant l \leqslant m$ and $a \in A^{\mathcal{C}}$:

$$u_l(a) = -1 \times \begin{cases} 0 & \text{if } n_l - n \leqslant 0 \\ r_S^l(s) + r_A^l(s,a) + \sum_{s' \in S} \delta^{\mathcal{C}}(s,a)(s') \cdot v_{n+1}^{s',l} & \text{otherwise} \end{cases}$$

and $(v_{n+1}^{s',1}, \ldots, v_{n+1}^{s',m}) = \overline{\mathtt{V}}_{\mathsf{G}^{\mathcal{C}}}(s', \theta_{n+1})$ for all $s' \in S$.

6.3.4 Computing SCNE Values of Infinite-horizon Nonzero-sum Formulae

We next show how to compute SCNE values $\overline{\mathtt{V}}_{\mathsf{G}^{\mathcal{C}}}(s, \theta)$ for *finite-horizon* multi-player formula θ in all states s of $\mathsf{G}^{\mathcal{C}}$. The same changes described for finite-horizon formulae in Section 6.3.3 and the arguments concerning uniqueness and convergence for the computation of infinite-horizon SWNE values described in Section 6.3.2 apply here.

Until. For formulae of the type $\theta = \mathtt{P}[\,\phi_1^1\ \mathtt{U}\ \phi_2^1\,]+\cdots+\mathtt{P}[\,\phi_1^m\ \mathtt{U}\ \phi_2^m\,]$, values can be computed through value iteration as the limit $\overline{\mathtt{V}}_{\mathsf{G}^\mathcal{C}}(s,\theta) = \lim_{n\to\infty}\overline{\mathtt{V}}_{\mathsf{G}^\mathcal{C}}(s,\theta,n)$ where $\overline{\mathtt{V}}_{\mathsf{G}^\mathcal{C}}(s,\theta,n) = \overline{\mathtt{V}}_{\mathsf{G}^\mathcal{C}}(s,\varnothing,\varnothing,\theta,n)$ and for any $D, E \subseteq M$ such that $D \cap E = \varnothing$:

$$\overline{\mathtt{V}}_{\mathsf{G}^\mathcal{C}}(s,D,E,\theta,n) = \begin{cases} (\eta_D(1),\ldots,\eta_D(m)) & \text{if } D\cup E = M \\ (\eta_{Sat(\phi_2^1)}(s),\ldots,\eta_{Sat(\phi_2^m)}(s)) & \text{else if } n = 0 \\ \overline{\mathtt{V}}_{\mathsf{G}^\mathcal{C}}(s, D\cup D', E, \theta, n) & \text{else if } D' \neq \varnothing \\ \overline{\mathtt{V}}_{\mathsf{G}^\mathcal{C}}(s, D, E\cup E', \theta, n) & \text{else if } E' \neq \varnothing \\ -val(\mathsf{N}) & \text{otherwise} \end{cases}$$

where $D' = \{l \in M\backslash(D\cup E) \mid s \in Sat(\phi_2^l)\}$, $E' = \{l \in M\backslash(D\cup E) \mid s \in Sat(\neg\phi_1^l \wedge \neg\phi_2^l)\}$ and $val(\mathsf{N})$ equals SWNE values of the game $\mathsf{N} = (M, A^\mathcal{C}, u)$ in which for any $1 \leqslant l \leqslant m$ and $a \in A^\mathcal{C}$:

$$u_l(a) = -1 \times \begin{cases} 1 & \text{if } l \in D \\ 0 & \text{else if } l \in E \\ \sum_{s'\in S}\delta^\mathcal{C}(s,a)(s')\cdot v_{n-1}^{s',l} & \text{otherwise} \end{cases}$$

and $(v_{n-1}^{s',1}, v_{n-1}^{s',2}, \ldots, v_{n-1}^{s',m}) = \overline{\mathtt{V}}_{\mathsf{G}^\mathcal{C}}(s', D, E, \theta, n-1)$ for all $s' \in S$.

Expected Reachability. If $\theta = \mathtt{R}^{r_1}[\,\mathtt{F}\ \phi^1\,]+\cdots+\mathtt{R}^{r_m}[\,\mathtt{F}\ \phi^m\,]$, values can be computed through value iteration as the limit $\overline{\mathtt{V}}_{\mathsf{G}^\mathcal{C}}(s,\theta) = \lim_{n\to\infty}\overline{\mathtt{V}}_{\mathsf{G}^\mathcal{C}}(s,\theta,n)$ where $\overline{\mathtt{V}}_{\mathsf{G}^\mathcal{C}}(s,\theta,n) = \overline{\mathtt{V}}_{\mathsf{G}^\mathcal{C}}(s,\varnothing,\theta,n)$ and for any $D \subseteq M$:

$$\overline{\mathtt{V}}_{\mathsf{G}^\mathcal{C}}(s,D,\theta,n) = \begin{cases} \mathbf{0}_m & \text{if } D = M \\ \mathbf{0}_m & \text{else if } n = 0 \\ \overline{\mathtt{V}}_{\mathsf{G}^\mathcal{C}}(s, D\cup D', \theta, n) & \text{else if } D' \neq \varnothing \\ -val(\mathsf{N}) & \text{otherwise} \end{cases}$$

$D' = \{l \in M\backslash D \mid s \in Sat(\phi^l)\}$ and $val(\mathsf{N})$ equals SWNE values of the game $\mathsf{N} = (M, A^\mathcal{C}, u)$ in which for any $1 \leqslant l \leqslant m$ and $a \in A^\mathcal{C}$:

$$u_l(a) = -1 \times \begin{cases} 0 & \text{if } l \in D \\ r_S^l(s) + r_A^l(s,a) + \sum_{s'\in S}\delta^\mathcal{C}(s,a)(s')\cdot v_{n-1}^{s',l} & \text{otherwise} \end{cases}$$

and $(v_{n-1}^{s',1}, v_{n-1}^{s',2}, \ldots, v_{n-1}^{s',m}) = \overline{\mathtt{V}}_{\mathsf{G}^\mathcal{C}}(s', D, \theta, n-1)$ for all $s' \in S$.

6.4 Strategy Synthesis

When verifying a Nash formula $\langle\!\langle C_1{:}C_2{:}\cdots{:}C_m\rangle\!\rangle_{\mathrm{opt}\sim x}(\theta)$, we can also build a strategy that is a subgame-perfect SWNE or SCNE for the objectives $(X_1^\theta, \ldots, X_N^\theta)$. This

is achieved by keeping track of a SWNE for the NFG solved in each state. The synthesised strategies require randomisation and memory. Randomisation is needed for NE of NFGs. Memory is required for finite-horizon properties and given that choices change after a path formula becomes true or a target is reached.

For infinite-horizon properties only approximate ε-NE profiles are synthesised and memory is also needed to keep track of players that have satisfied their objectives as illustrated by Example 18 for the two-player case. As for strategy synthesis of infinite-horizon zero-sum and two-player properties discussed in Sections 4.6 and 5.4, given that we use value iteration, we cannot guarantee that the values computed are within a particular error bound of the actual values. Given we cannot switch to MDP model checking in games with more than two players, the memory elements are used to keep track of the sets D and E, which record the players who have reached and cannot reach their goals, respectively, as described in Section 6.3. Alternatively, an extended game can be built with additional labels and states to keep track of that information, as detailed in Chapter 7.

6.5 Correctness and Complexity

The proof of correctness follows similarly to the two-player case (see Appendix A, Section A.1.2 for the proof). In the case of finite-horizon nonzero-sum formulae the correctness of the model checking algorithm follows from the fact that we use backward induction [181, 205, 188]. For infinite-horizon nonzero-sum formulae the proof is based on showing that the values of the players computed during value iteration correspond to subgame-perfect SWNE or SCNE values of finite game trees, and the values of these game trees converge uniformly to the actual values of $\mathsf{G}^{\mathcal{C}}$. The only difference from the two-player case is that in the game trees when some of the players' objectives are met we need to keep track of this in the subsequent states of the game tree, whereas in the two-player case such situations reduce to MDP model checking. The complexity of the algorithm is linear in the formula size, and finding subgame-perfect NE for reachability objectives in n-player games is PSPACE [35]. Value iteration requires finding all NE for a NFG in each state of the model, and computing NE of an NFG with three (or more) players is PPAD-complete [71].

6.6 Summary

In this chapter, we reviewed how to compute Nash equilibria for CSGs through support enumeration and presented an encoding of this problem as a nonlinear program. Subsequently, we presented the syntax and semantics of a multi-coalitional extension of rPATL, which allows for property specification considering any given number of coalitions. We developed algorithms for computing/approximating SWNE or SCNE values of multi-coalitional properties, which are also used for strategy synthesis. Multi-coalitional properties add considerably to the complexity of model checking for equilibria properties, both in terms of solving games at a state level and over the overall graph. For this type of properties, we can no longer reduce part of the computation to MDP model checking and need to keep track of sets of players that have satisfied their objective at a given point. We conclude the chapter with a discussion of correctness and complexity.

CHAPTER 7

Tool Implementation

Contents

In this chapter we describe the implementation for the model checking and strategy synthesis methods for concurrent stochastic games based on the algorithms described in Chapters 4, 5 and 6, which account for the main new features of PRISM-games 3.0. We build on the existing implementation of PRISM-games 2.0 [144, 143] that already supported verification for single-objective and multi-objective zero-sum properties of turn-based stochastic games. PRISM-games is a mature state-of-art tool for verifying stochastic games, originally built as an extension of the probabilistic model checker PRISM [135]. Although the tool already supported TSGs, which were built as the result of simple synchronisation among different components or through a compositional approach guided by an *assume-guarantee* framework [211], both the modelling and property specification languages revealed themselves to be unsuitable for concurrent stochastic games. Additionally, it was not possible to verify equilibria properties over TSGs, which can now be done by modelling them as CSGs. We start by describing the modelling language extension, illustrating its main new features through an example. Subsequently, we discuss the choices and challenges behind

the implementation of the model checking procedures. A brief tool demonstration concludes this chapter.

PRISM-games 3.0 is available at https://www.prismmodelchecker.org/games/ and is an open source tool under the GPL licence, accompanied by a tool paper [139] presented at CAV 2020. Earlier prototype versions that supported CSGs were developed and used in [136, 137, 138, 140], and can also be found online.

7.1 Modelling and Property Specification Language

Prior to the work reported in this thesis, PRISM-games already allowed for the specification of action-labelled turn-based stochastic games using an extension of the native PRISM modelling language [143], which is itself based on the reactive modules formalism of [3]. In this section, we describe the changes necessary for supporting CSGs, together with the new features added to the modelling and property specification languages.

7.1.1 Overview

Model building in PRISM-games is done through *module composition*. *Modules* are used to represent smaller, self-contained parts of a system that may interact with other modules by reading their *variables* and synchronising through *action-labelled guarded commands*. When building a model, the tool makes a product among all synchronising commands whose *guards* are satisfied in a given state in order to determine the transitions the system may take. The next state is then computed by considering the *updates* of each command and the product of each probability distribution. CSGs could not be naturally modelled with this approach for several reasons: (i) players need to be able to concurrently choose between multiple commands with different action labels; (ii) the update performed by one player may be different depending on the action chosen by another player; (iii) when multiple players execute, variables may need to be updated according to an arbitrary probability distribution, rather than being limited to the product of separate distributions specified locally by individual modules.

The modelling language was adapted to accommodate the specification of CSGs by only allowing modules to be associated to players and, in any state, letting each player choose between enabled commands of the corresponding modules (if no command is enabled, the player idles). In order to allow the updates of variables to depend on the choices of other players, we extended the language by allowing commands to be

labelled with lists of actions $[act_1, \ldots, act_n]$, and thus represent behaviour dependent on other players' choices. Reward structures were extended similarly so that an individual player's rewards can depend on the choices taken by multiple players. Moreover, we also allow variables to be updated dependent on the updated values of other variables, provided there are no cycles of dependency. This allows variables of different players to be updated according to a joint probability distribution. Another feature added to this new version of the language is the ability to specify *independent* modules. Independent modules are not associated to any player and thus cannot define distinct action labels, being only able to synchronise with other modules. These modules are a useful addition and make it easier to model parallel components such as counters and observers. They also make it possible to have a clear definition of players as those in charge of how the system progresses, by linking players to modules that specify actions and nondeterministic choices.

Figure 7.1 shows an example of the PRISM-games 3.0 modelling language, which we use to illustrate some of its new features. It models a probabilistic version of the *medium access control* problem (MAC), previously described in [33]. Two users share a communication channel. At each time step, user `maci` (`i` = 1,2) can choose between transmitting a message (`ti`) or waiting (`wi`). Variable `si` tracks whether a user successfully sent its message in the last time step and `ei` represents its energy level: transmissions can only occur when energy is positive. A third component is the channel `channel`, modelled by Boolean variable `c` denoting whether a collision occurred on the last transmission attempt. See for example the update `(s1'=c'?0:1)` on line 22, which updates `s1` depending on whether there was a channel collision (reflected in `c'`, the updated value of `c`). We use this mechanism to model interference on the channel: module `channel` specifies a joint probability distribution which is used to update variables `s1` and `s2` simultaneously. The model in Figure 7.1 will be used as the running example in this chapter and is the one we will be referring to, unless it is explicitly stated otherwise.

7.1.2 Modelling CSGs in PRISM-games

Modules. Both in the original modelling language for PRISM and in that for PRISM-games, all behaviour is encapsulated in modules, whose states are defined by a set of finite-range variables and whose behaviour is specified using action-labelled guarded commands. Variables are typed, may be initialised to a value using the `init` keyword, and must be defined over a *domain* (e.g. the variables `si` in the model are defined over

```
csg
// Player specification
player p1 mac1 endplayer
player p2 mac2 endplayer
// Probability qi for transmission success when i users send
const double q1;
const double q2;
// Channel (computes joint transmission probabilities)
module channel
 c : bool init false; // Did a collision occur during transmission?
 [t1,w2] true -> q1:(c'=false) + (1-q1):(c'=true);// User 1 transmits
 [w1,t2] true -> q1:(c'=false) + (1-q1):(c'=true);// User 2 transmits
 [t1,t2] true -> q2:(c'=false) + (1-q2):(c'=true);// Both transmit
endmodule
// Max energy per user
const int emax;
// User 1
module mac1
 s1 : [0..1] init 0;// Has user 1 sent?
 e1 : [0..emax] init emax;// Energy level of user 1
 [w1] true -> true;// Wait
 [t1] e1>0 -> (s1'=c'?0:1) & (e1'=e1-1);// Transmit
endmodule
// Define second user using module renaming
module mac2 = mac1 [ s1=s2, e1=e2, w1=w2, t1=t2 ] endmodule
```

```
// Reward structures
// Number of messages sent by user 1
rewards "mess1"
 s1=1 : 1;
endrewards
// Number of messages sent by user 2
rewards "mess2"
 s2=1 : 1;
endrewards
// Number of times that users 1 and 2 transmit simultaneously
rewards "send2"
 [t1,t2] true : 1;
endrewards
```

Figure 7.1: A model of the MAC problem as concurrent stochastic game.

[0..1] and initialised to 0). In a state, one or more modules can execute a command to change its state: if the guard (a predicate over variables, considering their current state values) is satisfied, the state can be modified (probabilistically) by applying the updates of the command. Multiple modules can synchronise on a given set of actions and then execute their updates simultaneously depending on how their commands are labelled. It is assumed w.l.o.g. that all action names are *distinct*. If a module is associated to a player, its commands should be specified according to the following

syntax:

$$[act_1, \ldots, act_n]\ \texttt{guard} \rightarrow p_1 : \texttt{update}_1 + p_2 : \texttt{update}_2 \cdots ;$$

The list of actions that label each command in a player's module should have at least one action. The head of the list should always be an action belonging to that player, while the remainder can be written in any given order. Naturally, the length of each list should not exceed the number of player declarations in the model and should not have more than one action belonging to the same player. A list can be partially specified, that is, it can contain actions of only a subset of the players. In such cases, the tool assumes that the command synchronises with any other action of players not represented in the list.

Modules that are not associated to players are called *independent*. They are specified in the same way and can also have its set of variables. However, given they do not represent a player in the game, they cannot introduce new actions and its commands' labels are defined as a list of player actions written in any given order. A feature restricted to independent modules is the ability of having empty labels, in which case the command synchronises with any combination of actions and is executed at every step provided its guard is satisfied. Such commands follow the syntax:

$$[]\ \texttt{guard} \rightarrow p_1 : \texttt{update}_1 + p_2 : \texttt{update}_2 \cdots ;$$

Naturally, independent modules should not introduce nondeterminism in a model and thus special attention has to paid to appropriately defining its guards when synchronising on lists of actions or specifying empty-labelled commands.

Players. Players are added to a model by means of *player* declarations defined according to the following syntax:

$$\texttt{player p}\ mod_1, \ldots, mod_n\ \texttt{endplayer}$$

A player may have an arbitrary number of modules associated to it and must have at least one. This is done by providing the list of module names $(mod_1, \ldots, mod_n)$ one wants a player to control. An action declared in one module (by being the head of a list labelling a command) can be reused in other modules provided they all belong to the same player. If in a given state no command belonging to any of the player's module has its guards satisfied, that player idles (which is the equivalent of playing $\perp$ as described in Section 4.3).

Rewards. Reward structures can be added to a model using the following syntax:

```
rewards "r"
  [act_1, ..., act_n] guard : exp;
  ⋮
endrewards
```

The reward structure specified above is named `r` and is made up of a list of reward assignments as guarded statements: the optional label $[act_1, \ldots, act_n]$ specifies a list of actions, the guard `guard` restricts the set of states to which the reward is applied, and `exp` defines the rewards as a real value or an expression over variables. Rewards defined exclusively on actions are specified by assignments of the form $[act_1, \ldots, act_n]$ `true : values`, which are trivially satisfied in every state. Similarly, assignments of the form [] `guard : exp` are valid for all actions of states that satisfy `guard`. Finally, assignments of the form `guard : true` are used to specify rewards that are applied exclusively to states. For example, in Figure 7.1, the reward structures `messi` define state rewards that assign a value of 1 to states that correspond to player i having sent a message successfully, whereas the reward structure `send2` assigns a value of 1 to each time actions `t1` and `t2` are taken simultaneously.

Specification Language. PRISM-games 3.0 also extends the language used to specify properties for verification and strategy synthesis. The previous implementation already supported zero-sum queries for TSGs using rPATL, including multi-objective formulae, which were expressed as Boolean combinations of zero-sum objectives. Support for zero-sum properties of CSGs is a straightforward extension of the implementation for TSGs. For equilibria properties, we compute values or synthesise strategies which are SWNE or SCNE, i.e., which maximise (or minimise) the sum of the values associated to the objectives for each player. We express such properties by adding to rPATL the + operator, which is then used to denote the sum of the values associated to both *bounded* and *unbounded* objectives.

When using the reward operator in equilibria properties, we can reason about *cumulative* ($\mathtt{C}^{\leqslant k}$), *instantaneous* ($\mathtt{I}^{=k}$) and *expected reachability* ($\mathtt{F}$) objectives, with *total rewards* ($\mathtt{C}$) and the variant of expected reachability ($\mathtt{F}^c$) being restricted to zero-sum. For properties with the probability operator, we support bounded and unbounded reachability using the temporal operators *next* ($\mathtt{X}$), *eventually*($\mathtt{F}$) and *until* ($\mathtt{U}$) . With respect to expressing zero-sum properties for CSGs, we have implemented all the previous temporal operators for probabilistic queries and a subset of the rPATL operators reported in [53] (with the exception being $\mathtt{F}^0$) for reward-based queries,

adding to that the instantaneous reward operator. The total reward operator had been added within the context of multi-objective properties [144].

Finally, following the style of rPATL, we separate players into *coalitions* with the syntax $\langle\langle coalition \rangle\rangle$, in order to specify the player or association of players for which we seek to maximise or minimise the values for a given zero-sum property. For equilibria properties, given that we maximise/minimise the sum, we use the same operator to separate players in different coalitions using a colon, while players in the same coalition are separated by a comma. Figure 7.2 shows two properties for the running example. Both properties listed are SWNE equilibria properties that separate the two players into different coalitions. The first computes the accumulated rewards up to instant k and the second the sum of probabilistic reachability values.

```
1 const int k; // Time bound
2
3 // Each player wants to maximise the
4 // expected number of messages sent in k steps
5 <<p1:p2>>max=? (R{"mess1"}[C<=k] + R{"mess2"}[C<=k])
6
7 // Each player wants to maximise the
8 // probability of sending a message
9 <<p1:p2>>max=? (P[F s1=1] + P[F s2=1])
```

Figure 7.2: Properties for the MAC problem model shown in Figure 7.1.

7.2 Implementation Details

In this section, we present some details concerning the implementation for model building and model checking.

7.2.1 Model Building

CSGs are built and stored in explicit-state fashion using an extension of PRISM's Java-implemented *explicit* (sparse-matrix based) engine. CSGs are stored "flat", that is, each state has a list of choices indexed by a product of actions. The reason for that is that local games (matrix, bimatrix or NFGs, which describe a state) can only be built once a coalition (or list of coalitions) has (have) been specified. Each collective choice points to a distribution over the set of successor states. Model building for CSGs is more intricate and costly than for TSGs, due to both the nature of the problem and the new features added to the language. The fact that different

commands can synchronise partially on lists of actions requires different levels of checking in order to make sure there is no nondeterminism or underspecification, for instance, that in a two-player game *there is* an outcome and *just one* outcome for every possible element of the product of action sets $A_1 \times A_2$. Due to the added feature of using updated values of variables, we also need to make sure their updates are not mutually dependent.

Figure 7.3 shows the state space representation in Dot format [93] of the medium access problem as exported by PRISM-games (when instantiating the model with `q1` = 0.9 and `q2` = 0.8). The states are represented by the diamond-shaped nodes, which contain the values for the different variables of the system in the order they are declared (in this case (`c`, `s1`, `e1`, `s2`, `e2`)). Connecting edges represent transitions which are annotated with the element of action product set they represent, along with the corresponding probability values, e.g., if `p1` chooses `t1` and `p2` chooses `w2` from the state (`false`, `0`, `1`, `0`, `1`), the system moves to state (`true`, `0`, `0`, `0`, `1`) with probability 0.1 or to state (`false`, `1`, `0`, `0`, `1`) with probability 0.9.

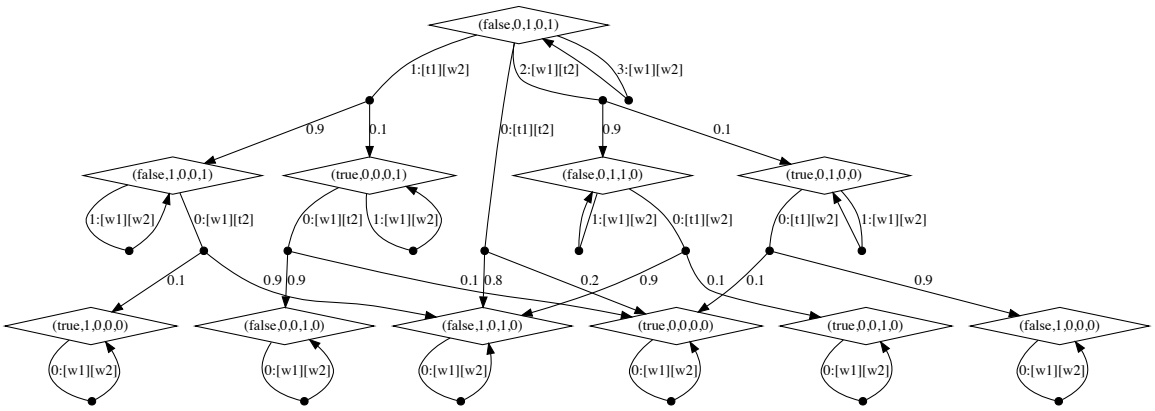


Figure 7.3: State space of the MAC problem as exported by PRISM-games 3.0.

7.2.2 Model Checking

Matrix games, bimatrix games and NFGs are built on the fly during verification. For all cases, we pre-allocate a number of variables for each player corresponding to the highest number of actions they may have at any state. We found that allocating variables either for the linear program or when using SMT solvers can be quite costly

in terms of time, and thus allocating a large enough number and resizing the LP or using the solvers' assertion stacks speeds up computation. As we have to make repeated calls to instantiate and solve a new problem for each state, the time spent on interfacing with solvers can be considerable.

7.2.2.1 Model Checking For Zero-sum Properties

When verifying zero-sum properties of CSGs, PRISM-games makes use of the model checking algorithms described in Chapter 4, which were based on the methods formulated in [76, 75]. We rely on backward induction and value iteration with classical convergence criteria to compute or approximate the values for all states of the game under study, and on solving a linear program to compute minimax values at each state. This corresponds to solving a matrix game, which is the representation of the one-shot zero-sum game for the enabled actions of each player in a given state. Prior to this numerical solution phase, we find and remove the states for which the optimal expected reward values are infinite by using the qualitative algorithms detailed in Appendix B. Our current implementation uses the LPsolve [148] library to solve the matrix games at each state. This library is based on the revised simplex and branch-and-bound methods.

Algorithm 1 General model checking routine for rPATL (for zero-sum formulae).

procedure ModelCheck($\phi, \mathsf{G}^{\mathcal{C}}$)
  $sat \leftarrow \varnothing$
  if $\phi = \top$ **then**
    $sat \leftarrow S$
  else if $\phi = l \in AP$ **then**
    $sat \leftarrow \{s \in S \mid l \in L(s)\}$
  else if $\phi = \neg\phi_1$ **then**
    $sat \leftarrow S \setminus$ ModelCheck(ϕ_1, $\mathsf{G}^{\mathcal{C}}$)
  else if $\phi = \phi_1 \wedge \phi_2$ **then**
    $sat \leftarrow$ ModelCheck(ϕ_1, $\mathsf{G}^{\mathcal{C}}$) $\cap$ ModelCheck(ϕ_2, $\mathsf{G}^{\mathcal{C}}$)
  else if $\phi = \langle\langle C \rangle\rangle \mathtt{P}_{\sim x}[\,\psi\,]$ **then**
    $sat \leftarrow$ ModelCheckProb(ϕ, $\mathsf{G}^{\mathcal{C}}$)
  else if $\phi = \langle\langle C \rangle\rangle \mathtt{R}^{r}_{\sim x}[\,\psi\,]$ **then**
    $sat \leftarrow$ ModelCheckRew(ϕ, $\mathsf{G}^{\mathcal{C}}$)
  end if
  return sat
end procedure

We present the algorithms for rPATL model checking in pseudocode starting with the general recursive procedure for zero-sum formulae described in Algorithm 1. This

routine traverses a formula's parse tree breaking it down into sub-formulae and calling the appropriate sub-routines in order to compute the overall satisfying set of states. Algorithm 2 shows the procedure for the zero-sum probabilistic operator $\langle\langle C \rangle\rangle \mathtt{P}_{\sim x}[\,\psi\,]$. Due to space restrictions, the algorithm was broken into two separate listings. It is worth noting that lines 7 and 9 from Algorithm 2 correspond to solving the matrix game for state s when minimising and maximising for $\langle\langle C \rangle\rangle$, respectively. We make use of the same notation in the different algorithms for zero-sum formulae. Model checking for the *next* operator ($\mathtt{X}$) is a simple one step procedure, which computes the maximal or minimal probability of reaching a target state in the next step. For *until* ($\mathtt{U}$) and *bounded until* ($\mathtt{U}^{\leqslant k}$) multiple iterations may be required either to reach convergence or to reach the specified time bound. When computing the satisfying set for the unbounded version, we stop as soon as the maximal difference between the values of two consecutive iterations becomes smaller than ε, which we assume to be set beforehand, according to a given convergence precision criterion to be selected by the user. For the bounded version, we stop as soon as exactly k iterations of the loop have been performed.

Algorithm 2 Probability operator model checking routine (Next).

1: **procedure** ModelCheckProb($\phi, \mathsf{G}^{\mathcal{C}}$)
2: $sat \leftarrow \varnothing$
3: **if** $\psi = \mathtt{X}\,\phi$ **then**
4: $T \leftarrow$ ModelCheck(ϕ, $\mathsf{G}^{\mathcal{C}}$)
5: **for all** $s \in S$ **do**
6: **if** opt = min **then**
7: $val \leftarrow \min\limits_{a_1 \in A_1^{\mathcal{C}}} \max\limits_{a_2 \in A_2^{\mathcal{C}}} \sum\limits_{t \in T} \delta^{\mathcal{C}}(s, a_1, a_2)(t)$
8: **else**
9: $val \leftarrow \max\limits_{a_1 \in A_1^{\mathcal{C}}} \min\limits_{a_2 \in A_2^{\mathcal{C}}} \sum\limits_{t \in T} \delta^{\mathcal{C}}(s, a_1, a_2)(t)$
10: **end if**
11: **if** $val \sim x$ **then**
12: $sat \leftarrow sat \cup \{s\}$
13: **end if**
14: **end for**

Model checking for zero-sum reward formulae $\langle\langle C \rangle\rangle \mathtt{R}^{r}_{\sim x}[\,\psi\,]$ is done in a similar way, by first determining the property type and calling the appropriate sub-routine. For reward formulae, we have separate algorithms for instantaneous ($\mathtt{I}^{=k}$), total and cumulative ($\mathtt{C}$, and $\mathtt{C}^{\leqslant k}$) and the two variants of expected reachability ($\mathtt{F}$ and $\mathtt{F}^{c}$). The pseudocode for the zero-sum reward formulae is included in Appendix B.2.

Algorithm 2 Probability operator model checking routine (Until).

15: **else if** $\psi = \phi_1 \ \mathtt{U} \ \phi_2$ **or** $\psi_1 \ \mathtt{U}^{\leqslant k} \ \phi_2$ **then**
16: $T_2 \leftarrow \textsc{ModelCheck}(\phi_2, \mathsf{G}^{\mathcal{C}})$
17: $T_1 \leftarrow \textsc{ModelCheck}(\phi_1, \mathsf{G}^{\mathcal{C}}) \setminus T_2$
18: $iter \leftarrow 0$; $X \leftarrow 0$; $X' \leftarrow 0$
19: **for all** $s \in S$ **do**
20: **if** $s \in T_2$ **then** $X[s] \leftarrow 1$ **else** $X[s] \leftarrow 0$ **end if**
21: **end for**
22: **while true do**
23: $X' \leftarrow X$
24: $iter \leftarrow iter + 1$
25: **for all** $s \in S$ **do**
26: **if** $s \in T_1$ **then**
27: **if** opt $=$ min **then**
28: $X[s] \leftarrow \min_{a_1 \in A_1^{\mathcal{C}}} \max_{a_2 \in A_2^{\mathcal{C}}} \sum_{t \in S} \delta^{\mathcal{C}}(s, a_1, a_2, t) \cdot X'[t]$
29: **else**
30: $X[s] \leftarrow \max_{a_1 \in A_1^{\mathcal{C}}} \min_{a_2 \in A_2^{\mathcal{C}}} \sum_{t \in S} \delta^{\mathcal{C}}(s, a_1, a_2, t) \cdot X'[t]$
31: **end if**
32: **end if**
33: **if** $X[s] \sim x$ **then**
34: $sat \leftarrow sat \cup \{s\}$
35: **end if**
36: **end for**
37: **if** $k \leqslant iter$ **then**
38: **break**
39: **else if** $\textsc{CheckConvergence}(\varepsilon, X, X')$ **then**
40: **break**
41: **end if**
42: **end while**
43: **end if**
44: **return** sat
45: **end procedure**

7.2.2.2 Model Checking For Two-player Equilibria Properties

For equilibria properties of CSGs, PRISM-games 3.0 implements the methods described in Chapters 5 and 6. As for the zero-sum case, we rely on value iteration and backward induction to compute/approximate values or synthesise strategies that are SWNE or SCNE. We remind the reader that we can only compute exact values for finite-horizon properties. For infinite-horizon properties, we can only compute values that are ε-Nash, and assume ε is specified beforehand. At each state, we solve a bimatrix game, which is a representation of a one-shot nonzero-sum game and is a linear complementarity problem. We solve these games via labelled polytopes finding all equilibria values through an SMT-based implementation, for which we use third-party SMT solvers Z3 [77] and Yices [79]. We make use of a pre-computation step of finding and removing *strictly dominated strategies* in order to minimise the number of calls to the solver.

Regarding the complexity of solving bimatrix games, if each player has n actions, then the number of possible assignments to the supports[1] of the strategy profiles is $(2^n-1)^2$, which therefore grows exponentially with the number of actions, surpassing 4.2 billion when each player has 16 actions. This particularly affects performance in cases which one or both players are *indifferent* with respect to a given support. More precisely, in such cases, if there is a equilibrium including pure strategies over these supports, then there are also equilibria including mixed strategies over these supports as the indifferent player would get the same utility for *any affine combination* of pure strategies.

Example 22. Consider the following bimatrix game:

$$\mathsf{Z}_1 = \begin{array}{c} \\ a_1 \\ a_2 \end{array} \begin{array}{c} \begin{array}{cc} b_1 & b_2 \end{array} \\ \begin{pmatrix} 1 & 0 \\ 1 & 0 \end{pmatrix} \end{array} \qquad \mathsf{Z}_2 = \begin{array}{c} \\ a_1 \\ a_2 \end{array} \begin{array}{c} \begin{array}{cc} b_1 & b_2 \end{array} \\ \begin{pmatrix} 2 & 2 \\ 4 & 4 \end{pmatrix} \end{array}$$

Since the entries in the rows for the utility matrix for player 1 are the same and the columns are the same for player 2, it is easy to see that both players are indifferent with respect to their actions. As can be seen in Table 7.1, all $(2^2-1)^2 = 9$ possible support assignments lead to an equilibrium as both players are indifferent.

■

For the task of computing a non-optimal NE, the large number of supports can be somewhat mitigated by eliminating *weakly dominated* strategies [166]. However,

[1]The support of a strategy profile is the set of action-tuples that are chosen with nonzero probability.

Player 1 strategy		Player 2 strategy		Utilities
prob. a_1	prob. a_2	prob. b_1	prob. b_2	(u_1, u_2)
0.0	1.0	0.0	1.0	(0.0,4.0)
0.0	1.0	1.0	0.0	(1.0,4.0)
0.0	1.0	0.5	0.5	(0.5,4.0)
1.0	0.0	0.0	1.0	(0.0,2.0)
1.0	0.0	1.0	0.0	(1.0,2.0)
1.0	0.0	0.5	0.5	(0.5,2.0)
0.5	0.5	0.0	1.0	(0.0,3.0)
0.5	0.5	1.0	0.0	(1.0,3.0)
0.5	0.5	0.5	0.5	(0.5,3.0)

Table 7.1: Possible NE strategies and utilities of Example 22.

removing such strategies is not a straightforward task when computing SWNE or SCNE, since it can lead to the elimination of SWNE or SCNE. For example, if we removed the row corresponding to action a_2 or the column corresponding to action b_1 from the matrices in Example 22 above, then we eliminate an SWNE. As the number of actions for each player increases, the number of NE also tends to increase and so does the likelihood of indifference. Naturally, the number of actions also affects the number of variables that have to be allocated, and the number and complexity of assertions passed to the SMT solver. As our method is based on the progressive elimination of support assignments that lead to NE, it takes longer to find SWNE and SCNE as the number of possible supports grows and further constraints are added each time an equilibrium is found.

Algorithm 3 shows the implementation for two-coalitional equilibria formulae with probabilistic reachability operators, that is, $\langle\langle C{:}C' \rangle\rangle_{\text{opt}\sim x}(\mathtt{P}[\,\mathtt{F}\ \psi_1\,] + \mathtt{P}[\,\mathtt{F}\ \psi_2\,])$. The code follows a similar structure as for zero-sum operators with the major difference being the computation of SCNE/SWNE values instead of solving a matrix game and the convergence check, which now has to be done on two solution vectors. Pseudocode for the other equilibria operators and formulae is shown in Appendix B.3.

7.2.2.3 Model Checking For Multi-player Equilibria Properties

The computation of equilibria values and the strategy synthesis algorithms for multi-coalitional equilibria properties relies on both SMT and non-linear solvers. Finding SWNE of NFGs, which can be reduced to solving a nonlinear programming problem (see Section 6.1), is performed using a combination of the SMT solver Z3 and the nonlinear optimisation suite IPOPT [207]. Although SMT solvers are able to find solutions to nonlinear problems, they are not guaranteed to do so and are only efficient in certain cases. These cases include when there is a small number of actions per player or finding support assignments for which an equilibrium is not possible. To mitigate the inefficiencies of the SMT solver, we use Z3 for filtering out unsatisfiable

Algorithm 3 Model checking routine for two-player infinite-horizon nonzero-sum probability formulae (Until).

procedure ModelCheckEqInfHorizonProb($\phi, \mathsf{G}^{\mathcal{C}}$)
$sat \leftarrow \varnothing; R \leftarrow \varnothing$
$T_{\phi_j^i} =$ ModelCheck(ϕ_j^i, $\mathsf{G}^{\mathcal{C}}$)
$X_{\psi_1} \leftarrow \mathtt{P}_{\text{opt}}[\, \phi_1^1 \ \mathtt{U} \ \phi_2^1 \,]$
$X_{\psi_2} \leftarrow \mathtt{P}_{\text{opt}}[\, \phi_1^2 \ \mathtt{U} \ \phi_2^2 \,]$
for all $s \in S$ **do**
  $R \leftarrow R \cup \{s\}$
  if $s \in T_{\phi_2^1} \cap T_{\phi_2^2}$ **then**
    $X_1[s] \leftarrow 1; X_2[s] \leftarrow 1$
  else if $s \in T_{\phi_2^1}$ **then**
    $X_1[s] \leftarrow 1; X_2[s] \leftarrow X_{\psi_2}[s]$
  else if $s \in T_{\phi_2^2}$ **then**
    $X_1[s] \leftarrow X_{\psi_1}[s]; X_2[s] \leftarrow 1$
  else if $s \in T_{\phi_1^1} \backslash T_{\phi_1^2}$ **then**
    $X_1[s] \leftarrow X_{\psi_1}[s]; X_2[s] \leftarrow 0$
  else if $s \in T_{\phi_1^2} \backslash T_{\phi_1^1}$ **then**
    $X_1[s] \leftarrow 0; X_2[s] \leftarrow X_{\psi_2}[s]$
  else if $s \notin T_{\phi_1^1} \cap T_{\phi_1^2}$ **then**
    $X_1[s] \leftarrow 0; X_2[s] \leftarrow 0$
  else
    $X_1[s] \leftarrow 0; X_2[s] \leftarrow 0; R \leftarrow R \backslash \{s\}$
  end if
end for
while true do
  $X_1' \leftarrow X_1; X_2' \leftarrow X_2$
  for all $s \in S \backslash R$ **do**
    if opt = min **then**
      Eq $\leftarrow$ Scne(s, X_1', X_2')
    else
      Eq $\leftarrow$ Swne(s, X_1', X_2')
    end if
    $X_1[s] \leftarrow$ Eq[1]; $X_2[s] \leftarrow$ Eq[2]
  end for
  if CheckConvergence($\varepsilon, X_1, X_1', X_2, X_2'$) **then**
    break
  end if
end while
for all $s \in S$ **do**
  if $X_1[s] + X_2[s] \sim q$ **then** $sat \leftarrow sat \cup \{s\}$ **end if**
end for
return *sat*
end procedure

support assignments with a timeout, and therefore given a support assignment Z3 returns either *unsat*, *sat* or *unknown* (if the timeout is reached). If either *sat* or *unknown* are returned, then the assignment is passed to IPOPT, which checks for satisfiability (if required) and approximates SWNE values using an interior-point filter line-search algorithm [208]. To speed up the overall computation the support assignments are analysed in parallel. We also search for and filter out *dominated strategies* as a precomputation step. The NFGs are built on the fly, as well as the gradient of the objective function defined in Equation (6.3) and the Jacobian of the constraints defined in Equations (6.4) - (6.6) of the nonlinear program described by Equations (6.3) - (6.7) (Section 6.1), which are required as an input to IPOPT.

In Section 6.3, we presented algorithms for computing SWNE and SCNE values of multi-player formulae over extensive games. The public release of PRISM-games 3.0 does not include the implementation for multi-coalitional properties, as IPOPT uses third-party libraries, including the HSL Mathematical Software Library [117] which does not allow redistribution. Furthermore, up to the moment of writing, only a subset of operators are supported. For infinite-horizon formulae, only simple probabilistic reachability ($\texttt{P}[\,\texttt{F}\ \phi\,]$) and expected reachabilty ($\texttt{R}[\,\texttt{F}\ \phi\,]$) were implemented. For finite-horizon, methods for bounded probabilistic reachability ($\texttt{P}[\,\texttt{F}^{\leqslant k}\ \phi\,]$) and instantaneous ($\texttt{R}[\,\texttt{I}^{=k}\,]$) and cumulative rewards ($\texttt{R}[\,\texttt{C}^{\leqslant k}\,]$) have been implemented, under the restriction the bounds are equal. The algorithms that are part of the prototype differ from the description of Chapter 6 in that they rely on building an extended game instead of branching when an objective has or cannot be satisfied and keeping track of multiple solution vectors. In the current implementation, for formulae of type $\langle\!\langle C_1{:}C_2{:}\cdots{:}C_m\rangle\!\rangle_{\mathrm{opt}\sim x}(\theta)$, given the set of sub-formulae $\phi^i \in \theta$, $i \in N$, we extend the original game by constructing a *subgame* G_C for each non-empty coalitions of players $C \subseteq N$. The subgame $\mathsf{G}_C = (N, S_C, \bar{S}_C, A, \Delta_C, \delta_C, AP', L_C)$ is constructed as follows:

$$
\begin{aligned}
&\bar{S}_C = \left(\textstyle\bigcap_{j\in N\setminus C} \overline{Sat(\phi^j)}\right) \cap \left(\textstyle\bigcap_{i\in C} Sat(\phi^i)\right)\\
&S_C = Reach_{\mathsf{G}}(\bar{S}_C);\\
&\Delta_C = \Delta(s) \text{ for all } s \in S_C;\\
&\delta(s,a) = \delta(s,a) \text{ for all } s \in S_C;\\
&AP' = AP \cup \{\mathtt{a}_{\phi_1}, \ldots, \mathtt{a}_{\phi_N}\};\\
&\mathtt{a}_{\phi_i} \in L_C(s) \text{ if and only if } i \in C.
\end{aligned}
$$

The set of initial states $\bar{S}_C$ of each subgame is composed of the states in the original game that satisfy ϕ^i for all $i \in C$ and do not satisfy ϕ^j for any $j \in N \setminus C$. If $\bar{S}^C = \varnothing$,

there is no need to build the subgame for C. We write $Reach_{\mathsf{G}}(\bar{S}_C)$ to refer to the set of states in G that can be reached from any state in $\bar{S}_C$. Separate reward structures also need to be set in order to properly compute the values of each subgame. Let $r^i_{A,C}$ and $r^i_{S,C}$ be the action and state rewards for player i in subgame G_C. For all $i \in C$, $j \in N \setminus C$, $n \in N$, $s \in S^C$, $a \in A$:

$$r_A^{C,i}(s,a) = 0 \text{ and } r_S^{C,i}(s) = 0;$$
$$r_A^{C,j}(s,a) = r_A^j(s,a) \text{ and } r_S^{C,j}(s) = r_S^j(s).$$

The set of values computed for each subgame G_C are the equilibria values for a game with $N - |C|$ players, which is the game played after the objectives of the players in C have been satisfied. Once the extended game has been built, we run backward induction and value iteration for a given number of iterations if verifying for a finite-horizon formula, or else until the values converge.

7.2.3 Strategy Synthesis

Strategies are also stored using the explicit (sparse matrix) engine. When synthesising strategies, for each state we store a distribution over action indices given both zero-sum and nonzero-sum properties required randomisation. For two-player equilibria strategies, the computation is split between the MDP computation, which provides the strategy once either player has achieved his goal and the equilibria computation. This is enough to reconstruct the strategy for the whole game, as we follow the CSG strategy until we reach a player's target state and the MDP from then on. Memoryless strategies guarantee optimality for reachability properties over MDPs. For the multi-player equilibria strategies, the current implementation builds an extended game as detailed in Section 7.2.2.3. Strategies are then computed over the new model, which is composed of the original game and the different sub-games that arise after any of the players have reached their goal.

We run the model checking algorithms to construct the strategy for a given property and then make the product of the strategy with the original (or extended) model to make it easier for the user to see how the system would progress if they were followed. The strategy and the product are exported in Dot format, which contains the graph corresponding to the product and the moves for each player at each state. The graph describes the execution of the system up to the point the objectives have been satisfied or the point from which a state that satisfies them cannot be reached. This is shown by self-loops which indicate the states which satisfy either condition.

In Figure 7.4, we show a strategy that was synthesised for the MAC problem, when running the property $\langle\langle p_1{:}p_2 \rangle\rangle_{\max=?}(\mathtt{P}[\,\mathtt{F}\,\mathtt{s}_1 = 1\,]{+}\mathtt{P}[\,\mathtt{F}\,\mathtt{s}_2 = 1\,])$, as exported by PRISM-games in Dot format. As in Figure 7.3, the nodes represent states which are distinguished by an index and a tuple of values for the variables in the system. The edges are annotated by an identifier denoting which algorithm generated that part of the strategy (CSG/MDP), the probability values that each action should be played and the resulting probability distribution over the next states when the players play accordingly. For instance, from state $(\mathtt{false}, 0, 1, 0, 1)$, if player `mac1` plays `t1` and `mac2` plays `w2`, both with probability 1, there is a 0.9 probability the system moves to $(\mathtt{false}, 1, 0, 0, 1)$ and a 0.1 probability it will move to $(\mathtt{true}, 0, 0, 0, 1)$. We highlight that transitions tagged with the MDP identifier mean that one of the players has already satisfied its goal, and should then cooperate with the other so as to maximise social-welfare (or minimise social-cost when applicable). Finally, we mention terminal states whose self-loops are transitions annotated by statements such as `CSG: Sat(0) - Unsat(1):1.0`, which means that the first player, `mac1`, has already satisfied its goal and that from that state it is not possible for player 2, `mac2`, to reach a state satisfying its goal.

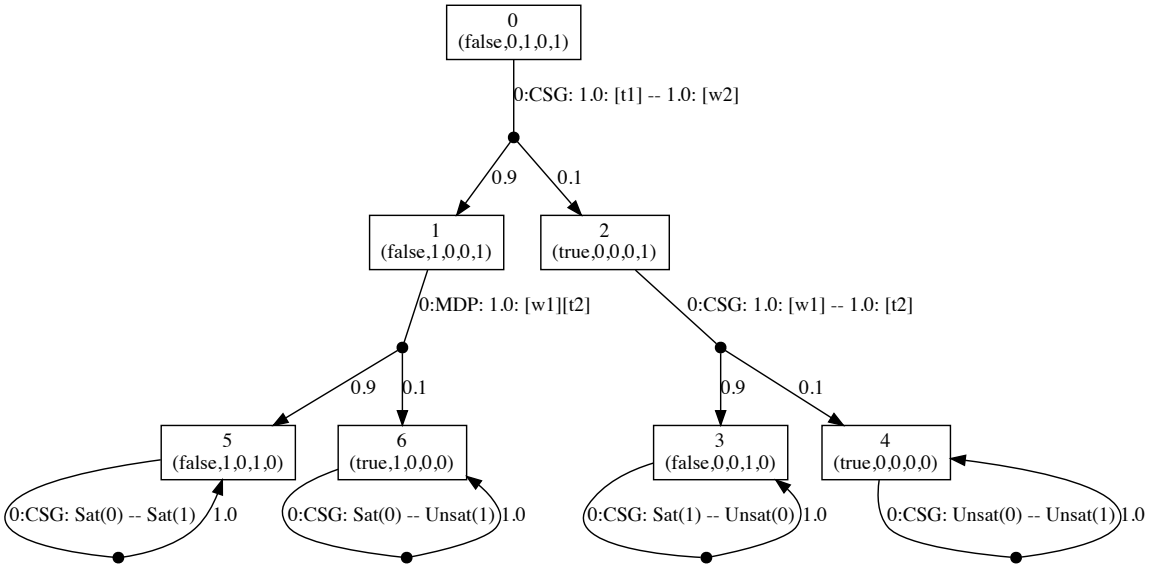


Figure 7.4: A strategy for the MAC example shown in Figure 7.1 as exported by PRISM-games 3.0.

7.3 Tool Demonstration

In this section we present a brief overview of the usage of PRISM-games 3.0. We refer the interested reader to https://www.prismmodelchecker.org/games/ for further details, download and installation instructions, and quick tutorials. We present the new functionalities of the tool focusing on added features for supporting CSGs. PRISM-games can used through a command line interface or a GUI, which can be launched by running `./bin/prism` and `./bin/xprism`, respectively, from the installation directory. By running `./bin/prism -help`, a comprehensive list of options and switches is displayed to guide the user. If running PRISM-games via the GUI, the user can access the main features of the tool through an intuitive set of menus and panels. In Figure 7.5, we show the model editor window of PRISM-games with the CSG model of the MAC problem. Models can be loaded and built by accessing the *Model* menu. If model building is successful, it triggers the tool to display basic statistics (number os states and transitions) in the bottom left corner.

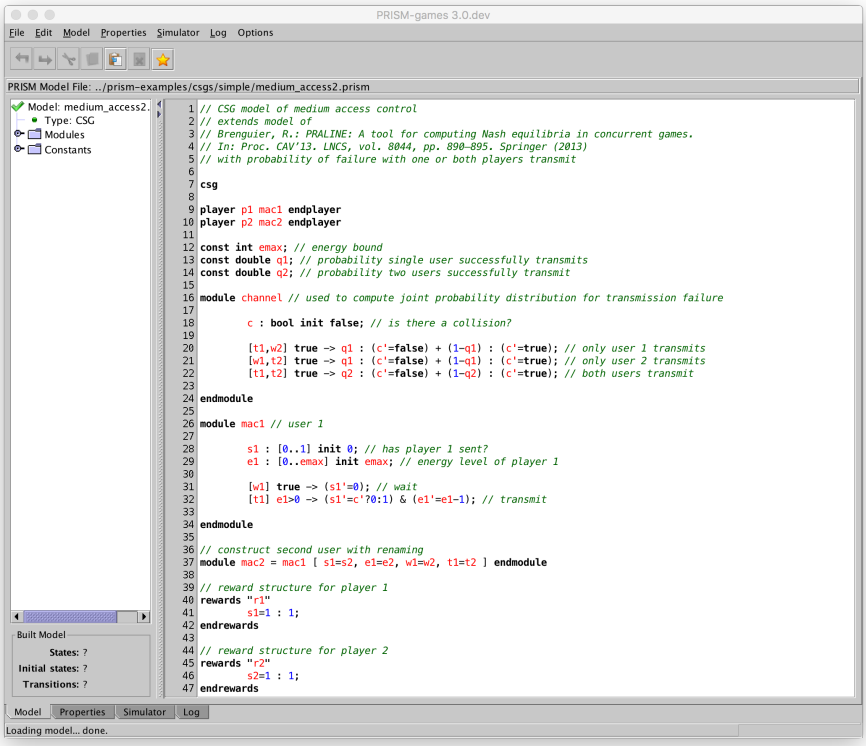


Figure 7.5: Screenshot of PRISM-games 3.0 showing a CSG model.

By accessing the *Simulator* tab, we change to the simulator window displayed

in Figure 7.6. In the simulator, each row in the table represents a state. Actions can be chosen automatically at random or manually. We show the construction of a path where, after spending all their energy units, both agents can only choose to wait (which is indicated by the arrow connecting two different rows). The possibility of constructing paths and checking the values of each variable at each step, as well the values for reward structures, is instrumental in debugging models and ensuring correct behaviour.

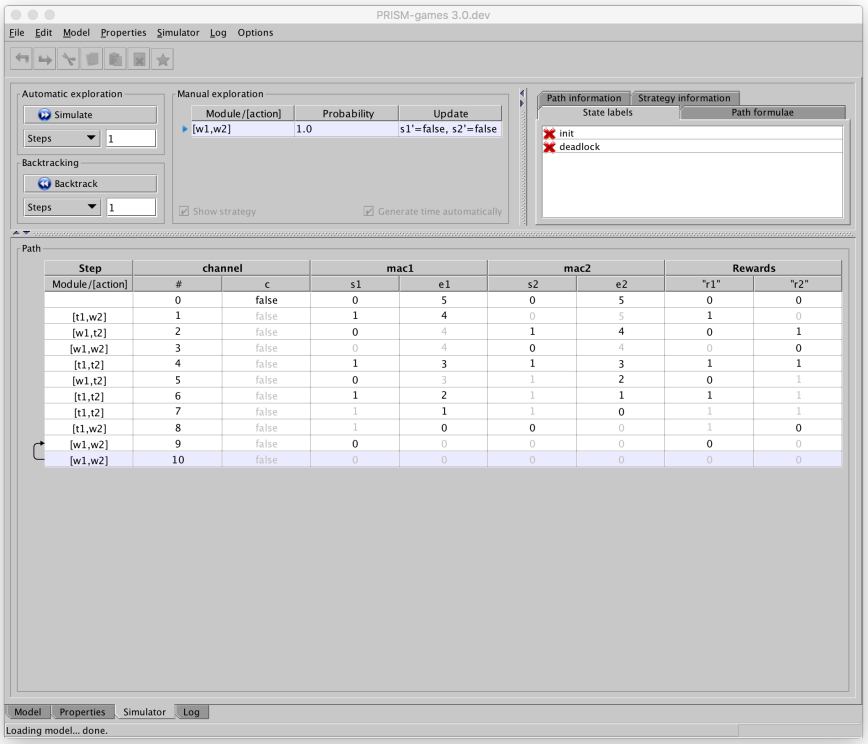


Figure 7.6: Screenshot of PRISM-games 3.0 showing a path being constructed using the simulator.

Moving to the *Properties* tab, we access the property verification window, shown in Figure 7.7. In it, we display the numerical result for the same quantitative property we synthesised the strategy displayed in Figure 7.4. We can see that the value computed for the SWNE property is 1.8, which can be easily confirmed by the strategy if we consider the sum of the individual probability values of reaching a state where either player has sent a message. Finally, in Figure 7.8 we show the plot corresponding to running an experiment for the property $\langle\langle p_1{:}p_2 \rangle\rangle_{\max=?}(\mathtt{R}^{mess_1}[\,\mathtt{C}^{\leqslant k}\,]+\mathtt{R}^{mess_2}[\,\mathtt{C}^{\leqslant k}\,])$ for $k \in [1, 12]$, which computes the sum of SWNE values of the number of messages

successfully sent in k rounds. Under the current implementation, strategy synthesis is only accessible through the command line. We plan to integrate that feature into the GUI in a future release.

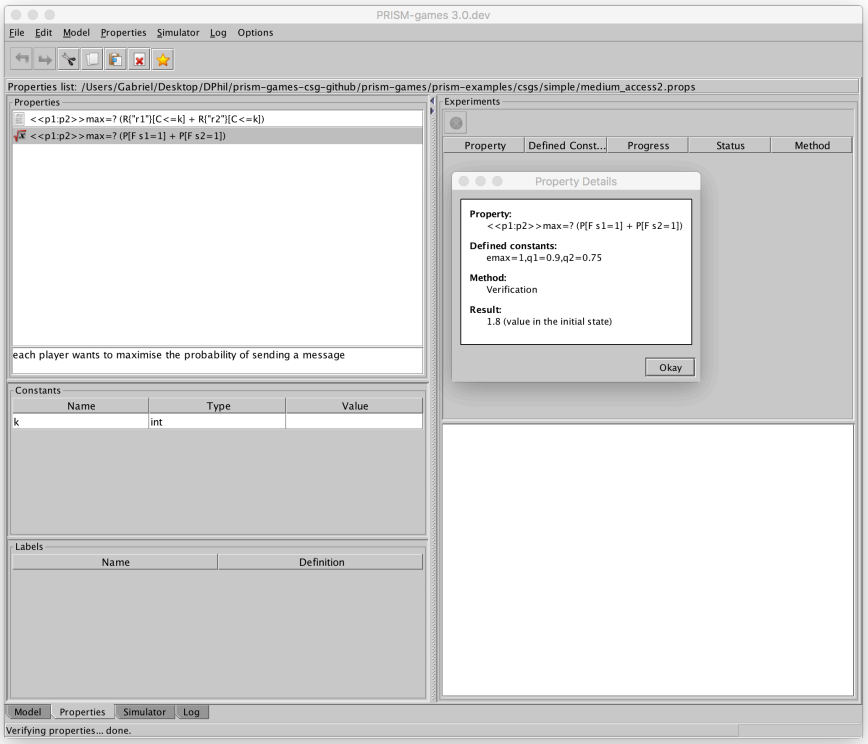


Figure 7.7: Screenshot of PRISM-games 3.0 showing a property being verified.

7.4 Summary

In this chapter we detailed the implementation of our framework for verification and strategy synthesis for zero-sum and equilibria properties of concurrent stochastic games within PRISM-games 3.0. Our tool provides an intuitive and versatile environment for modelling and analysing CSGs. We highlight the following main contributions: (i) we extended the modelling language of PRISM-games with various features to provide support for the specification of multi-player CSGs; (ii) we added new operators to rPATL to accommodate equilibria properties as well as to widen the scope of zero-sum verification; (iii) we extended the tool's model checking capabilities considerably, incorporating different third-party libraries and solvers to support solving different types of normal form games and allowing one to compute values for

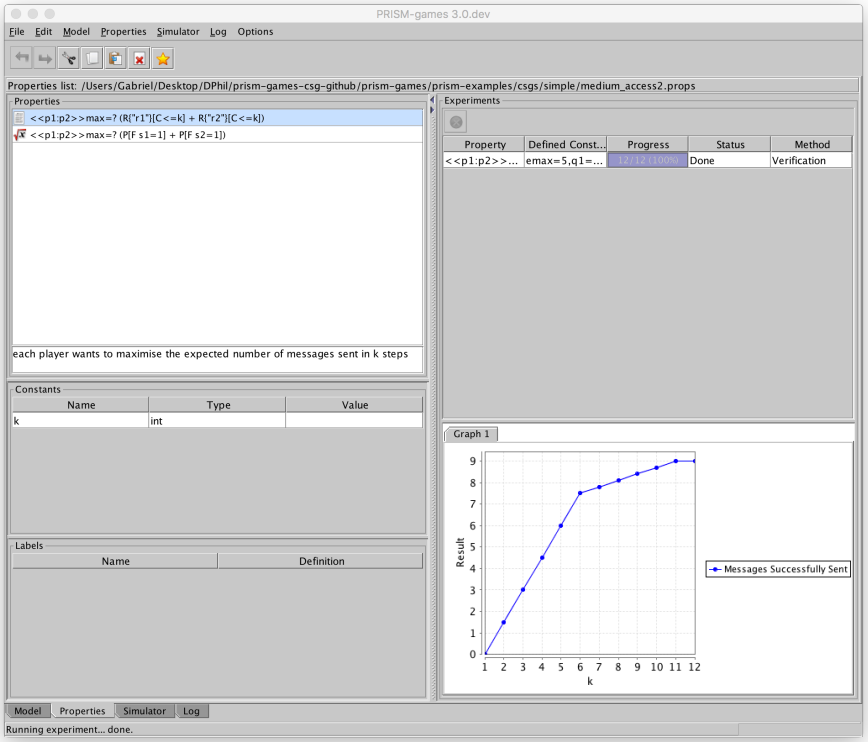


Figure 7.8: Screenshot of PRISM-games 3.0 plotting the results of an experiment when running an equilibria property.

properties with distinctive objectives, which may include a mixture of bounded and unbounded operators. To the best of our knowledge, at the time of elaboration of this document, our tool is the only of its kind to support zero-sum verification as well as the computation of SWNE and SCNE values for CSGs. Furthermore, given TSGs are a subclass of CSGs, these new features extend to turn-based games which can be modelled with the modified language. Finally, since we support multi-coalitional properties, the added extensions grant multi-player verification capabilities to Prism-games for the first time.

The current implementation could be enhanced in various ways. A key challenge is improving the performance of model checking algorithms, especially for equilibria properties. A more refined set of pre-computation techniques such as the removal of *conditionally dominated strategies* would speed up equilibria computation at the state level when using support enumeration. A similar point could be made about solving LPs, which could be sped up by incorporating methods of polynomial complexity. For two-player games, performance could be improved by considering other enumer-

ation solutions (e.g. *lexicographic reverse search*, *enumeration of extreme equilibria* [9]) as well as internal tests for indifference based on *minimal* supports. Another important improvement would be related to the internal representation of local games, probability distributions and solution vectors. PRISM-games currently relies on *doubles* to build and store the models as well as the different computation methods for strategy synthesis and model checking. Similarly to implementations developed for other probabilistic models, solution methods for CSGs can be sensitive to floating-point arithmetic issues, particularly for equilibria properties, and arbitrary precision representations should be considered to alleviate these problems. Other interesting extensions could explore different policies for selecting equilibria (for bounded properties) and add support for discounted models and multi-objective zero-sum properties for CSGs.

CHAPTER 8

Case Studies and Experimental Results

Contents

In this chapter, we present a series of case studies that were developed to showcase both the scope as well as the relevance of the verification and strategy synthesis methods detailed in this thesis. All models discussed in this chapter were written with the new modelling language described in Chapter 7 and verified with PRISM-games 3.0 against properties expressed in the extensions of rPATL described in Chapters 4-6. We discuss applications from a wide range of domains, including network protocols, attack-defence scenarios, finance, robot coordination and others. Another focus of this chapter is to demonstrate the advantages of adopting equilibria strategies in non-strictly competitive scenarios and the differences between two and muti-player games and properties.

As mentioned in Chapters 4-6, a number of assumptions are made with relation to the underlying graph structure and the specification of reward structures in order to guarantee convergence. The analysis is this chapter is then restricted to those previously established combinations of models and properties. We found that these restrictions do not significantly affect expressiveness. Preliminary versions of the case studies were reported in [136, 137, 138, 140] and can be found at https://prismmodelchecker.org/files/thesisgsantos/.

8.1 Case Studies

In some of the case studies, we compare our results with the corresponding zero-sum properties. For example, for some of the models for which we run the nonzero-sum property $\langle\langle C{:}C' \rangle\rangle_{\max=?}(\mathtt{P}[\,\mathtt{F}\ \phi_1\,]{+}\mathtt{P}[\,\mathtt{F}\ \phi_2\,])$, we compute the value and a zero-sum optimal strategy σ_C for coalition C of the formula $\langle\langle C \rangle\rangle \mathtt{P}_{\max=?}[\,\mathtt{F}\ \phi_1\,]$, and then find the value of a zero-sum optimal strategy for the coalition C' for $\mathtt{P}_{\min=?}[\,\mathtt{F}\ \phi_2\,]$ and $\mathtt{P}_{\max=?}[\,\mathtt{F}\ \phi_2\,]$ in the MDP induced by CSG when C follows σ_C. The aim is to showcase the advantages of cooperation as, in many real-world applications, agents' goals are not strictly opposed.

8.1.1 Robot Coordination

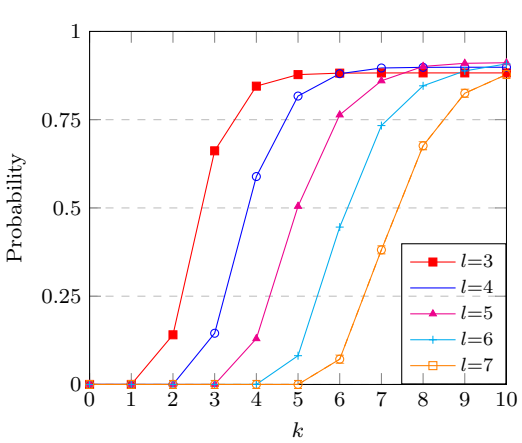
This case study expands on the scenario of Example 8, in which two robots move concurrently over a grid of size $l{\times}l$. The robots start in diagonally opposite corners and try to reach the corner from which the other starts. Recall that a robot can move either diagonally, horizontally or vertically towards its goal and when it moves there is a probability (q) that it instead moves in an adjacent direction. E.g., if a robot moves north east, then with probability $q/2$ it will move north or east. We suppose the robots try to maximise the probability of reaching their individual goals eventually and within a given number of steps (k). If there is no bound and $l \geqslant 4$, we notice that the SWNE strategies allow each robot to reach its goal with probability 1 (as time is not an issue, they can collaborate to avoid crashing).

Consider the objectives (X_1^k, X_2^k) where, for any infinite path π of G_l:

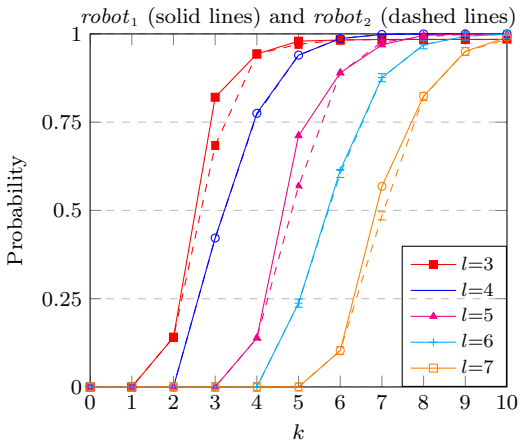
$$X_i^k(\pi) = \begin{cases} 1 & \text{if } \mathsf{goal}_i \in L(\pi(j)) \text{ for some } j \leqslant k \text{ and } \mathsf{crash} \notin L(\pi(m)) \text{ for all } m < j \\ 0 & \text{otherwise} \end{cases}$$

i.e., objective X_i^k returns 1 for paths on which $robot_i$ reaches its goal within k steps without crashing. So, the expected value of X_i^k for a given strategy profile equals the probability that $robot_i$ reaches its goal within k steps without crashing.

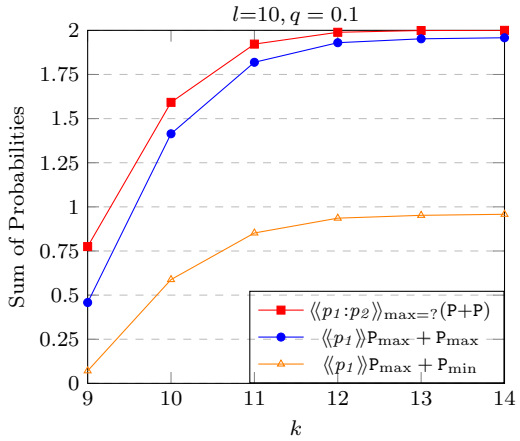
For $q{=}0.25$, we first consider the case where $k{=}\infty$. For robot 1, we find that the values in the initial state with respect to X_1^∞ converge to 1 as l increases, for example the values for G_5, G_{10} and G_{20} with respect to X_1^∞ are approximately 0.9116, 0.9392 and 0.9581, respectively. On the other hand, considering SWNE values for (X_1^∞, X_2^∞) and $l \geqslant 4$, the SWNE strategies allow each robot to reach its goal with probability 1 (since time is not an issue, they can collaborate to avoid crashing).



(a) Value of G_l with respect to X_1



(b) SWNE values of G_l for (X_1, X_2)

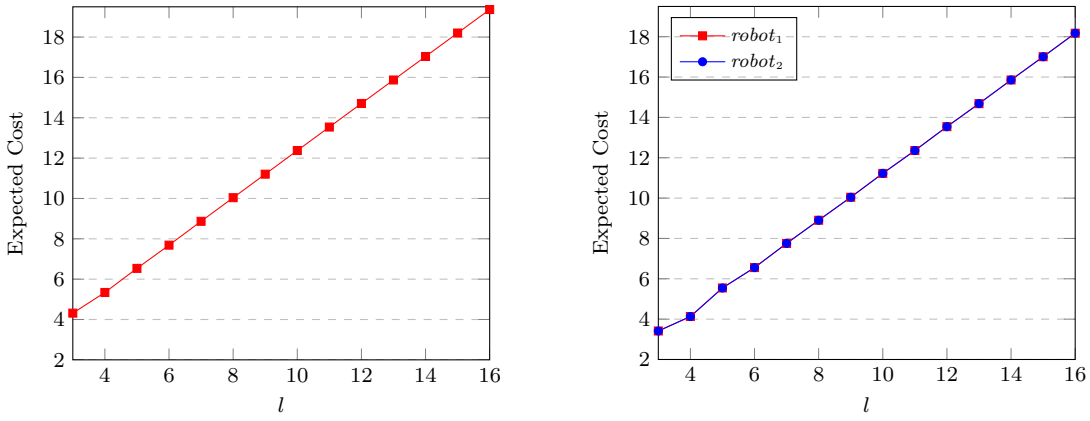


(c) Sum of SWNE and zero-sum values for finite-horizon reachability.

Figure 8.1: Robot coordination: probability of reaching the goal without crashing.

For the bounded case, Figure 8.1 shows, for a range of k and various grid sizes, both the values of G_l with respect to X_1^k (left) and SWNE values of the individual robots for (X_1^k, X_2^k) (right). We can also see in Figure 8.1 (bottom) the plot for the sum of SWNE and the comparison with the sum of zero-sum values for a grid of fixed size $l = 10$. When there is only one route to each goal within the bound (along the diagonal), i.e., when $k = l-1$, the SWNE strategies of both robots take this route. In odd grids, there is a high chance of crashing, but also a chance one will deviate and the other reaches their goal. Initially, as the bound k increases, for odd grids the SWNE values for the robots are not equal (see Figure 8.1 right). Here, it is better overall for one robot to follow the diagonal and the other to take a longer route, since if both took the diagonal route, the chance of crashing increases, decreasing the chance of reaching their goals. As expected, if we compare the optimal and SWNE

values, we see that the robots can improve their chances of reaching their goals by collaborating.



(a) Value of G_l with respect to Y_1 (b) SCNE values of G_l for (Y_1, Y_2)

Figure 8.2: Robot coordination: expected steps to reach the goal.

Next, we consider the objectives Y_1 and Y_2 where, for any infinite path π of G_l:

$$Y_i(\pi) = \begin{cases} \sum_{j=0}^{k^i_{\min}} r(\pi(i)) & \text{if } \mathsf{goal}_i \in L(\pi(j)) \text{ for some } j \\ \infty & \text{otherwise} \end{cases}$$

where $k^i_{\min} = \min\{j \in \mathbb{N} \mid \mathsf{goal}_i \in L(\pi(j))\}$ and, for any state s, we have $r(s)$=10 if $\mathsf{crash} \in L(s)$ and $r(s)$=1 otherwise. The expected value of objective Y_i for a given strategy profile equals the expected number of steps that player i requires to reach their goal, where crashing incurs a delay of 10 steps.

In Figure 8.2 we have plotted the zero-sum values of G_l with respect to Y_1 and SCNE values for the individual players for (Y_1, Y_2) as the grid size varies. The results again demonstrate that the players can gain by collaborating.

8.1.2 Secret Sharing

In this version of a *secret sharing* protocol [104], Halpern and Teague propose introducing uncertainty as a means to induce cooperation. In the first step of the protocol, all players toss a coin, which is simulated by the parameter α. If their coin lands heads, they are supposed to share their share of the secret with the other participants. If it lands tails, they do not share. In the second step, everyone reveals their coin. The game ends if everyone is able to reconstruct the secret or if someone cheats, that is, not revealing their share when supposed to. If at any stage, a player does not receive data from a player from whom it was supposed to, it also stops the

protocol. Also, in other to make the protocol fair, if exactly two players get heads, then no one learns the secret. Otherwise, if no one cheats but is not able to learn the secret, the issuer issues new secret shares to the players and a new round starts. The protocol also assumes that each player prefers learning the secret to not learning it and prefers that fewer of the others learn it. This is expressed in the model by the variables $u_{all}, u_{two}, u_{one}$ and u_{none}, which are the payoffs a player i gets if everyone, two players (including i), only i, and no one is able to reconstruct the secret, respectively.

A *rational* player in this context is one that has the choice between cheating and not cheating, that is, ignoring the coin toss in order to maximise the probability of being the only one to learn the secret. An *altruistic* player is one who follows the protocol strictly. Finally, a *byzantine* player is one who may fail by sending or computing the wrong values. The protocol's steps can be summarised as follows:

1. Each player is issued a signed share of the secret.

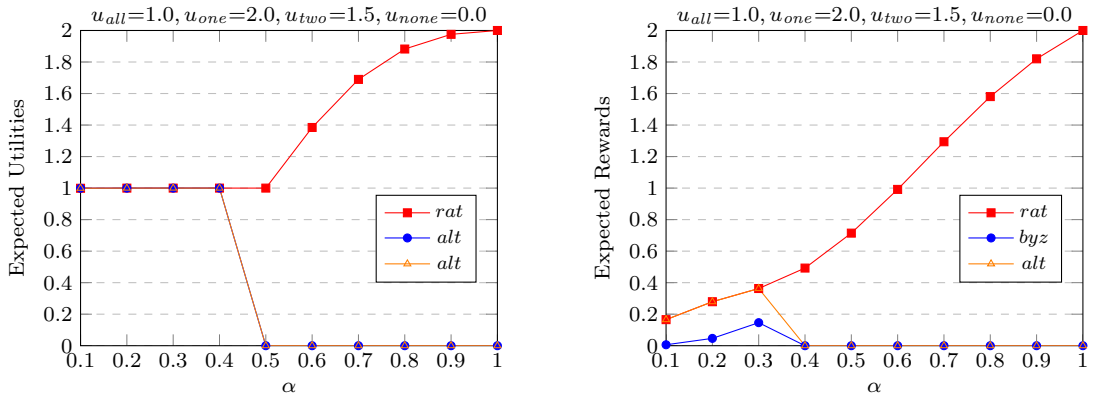
2. Each player i chooses a bit c_i that is set to 0 with probability α and 1 with probability $1-\alpha$. Another bit, $c_{(i,+)}$ is set uniformly (i.e. 0 or 1 with probability 0.5). A third bit, $c_{(i,-)}$, is computed as $c_i \oplus c_{(i,+)}$. Player i then sends $c_{(i,+)}$ to player i^+ and $c_{(i,-)}$ to i^-. This naturally means that it receives $c_{(i^+,-)}$ from i^+ and $c_{(i^-,+)}$ from i^-.

3. Each player then computes $c_{(i^+,-)} \oplus c_i$ and sends it to i^-. That way, each player receives $c_{((i^+)^+,-)} \oplus c_{i^+} = c_{(i^-,-)} \oplus c_{i^+}$ from i^+.

4. Each player calculates $p = c_{(i^-,+)} \oplus c_{(i^-,-)} \oplus c_{i^+} \oplus c_i = c_{i^-} \oplus c_{i+} \oplus c_i$. If $p = c_i = 1$ then player i sends its signed share to the others.

5. If $p = 0$ and i received no shares, or if p = 1 and player i received exactly one share (possibly from itself), the issuer is asked to restart the protocol; otherwise, i stops the protocol (given that either it has all shares or because someone must have cheated).

Figure 8.3 presents the expected utilities when there are two altruistic and one rational agent, and when there is one altruistic, one byzantine and one rational agent, as α varies. The results when there is one altruistic and two rational agents or three rational agents yield the same graph as Figure 8.3(left), where the one or two additional rational agents utilities match those of the altruistic agents. According to



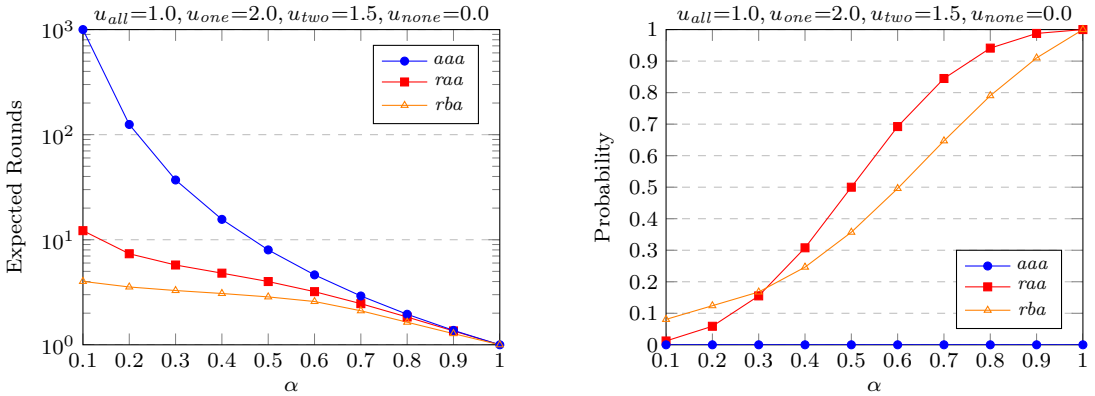
(a) 2 altruistic, 1 rational. (b) 1 altruistic, 1 byzantine, 1 rational.

Figure 8.3: $\langle\!\langle p_1{:}p_2{:}p_3 \rangle\!\rangle_{\max=?}(\mathtt{R}[\mathtt{F}\ \mathit{done}]{+}\mathtt{R}[\mathtt{F}\ \mathit{done}]{+}\mathtt{R}[\mathtt{F}\ \mathit{done}])$, ($p_{fail}$=0.2 for *rba*).

[104], for a model with one rational and two altruistic agents, the rational agent only has an incentive to cheat if:

$$\frac{u_1{\cdot}\alpha^2}{\alpha^2+(1-\alpha)^2} + \frac{u_0{\cdot}(1-\alpha)^2}{\alpha^2+(1-\alpha)^2} > u_3 \tag{8.1}$$

This is validated by Figure 8.3(left) for the given utility values; the rational agent only cheats when $\alpha \geqslant 0.5$ (for $\alpha < 0.5$ all agents receive a utility of 1 corresponding to all agents getting the secret), which corresponds to when Equation (8.1) holds for our chosen utility values. Figure 8.3 also shows that the closer α is to one then the greater the expected utility of a rational agent. It is also possible to notice in Figure 8.3(right) that, with a byzantine agent, the rational agent cheats when $\alpha \geqslant 0.4$.

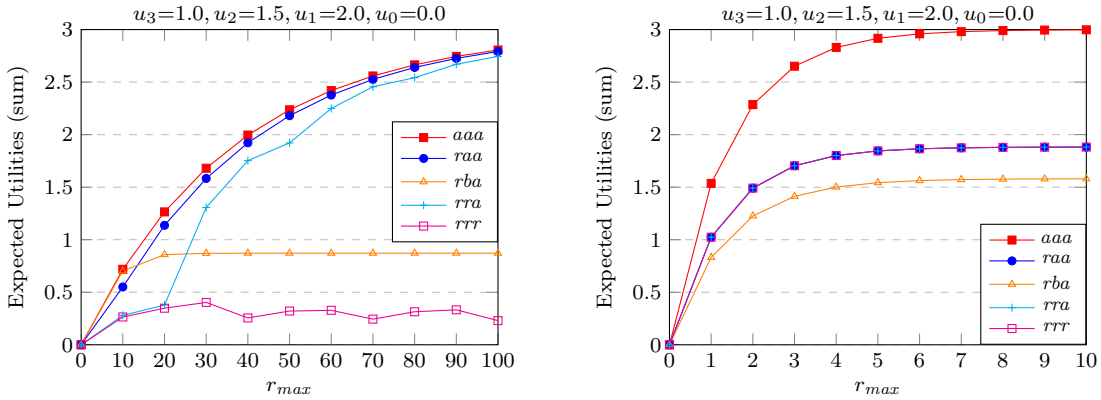


(a) Expected rounds. (b) Prob. only first player learns secrets.

Figure 8.4: Expected number of rounds (left) and probability only the first player learns the secret (right). (p_{fail}=0.2 for *rba*).

Figure 8.4 (left) demonstrates that there is a trade off between the protocol being

unfair and the minimum time that the protocol takes to complete. We point out that this does not mean all agents would have been able to reconstruct the secret, as the protocol also stops when an agent cheats, and the optimal behaviour so that all agents receive the other shares is that of an altruistic player. Indeed, that is what explain the difference between the curves, as a protocol with a rational agent takes less time to complete given the agent can choose to always cheat. Similarly, in Figure 8.4 (right) we show the maximal probability values of the first player being the only one to learn the secret for different values of α. It is possible to notice that, as expected, for a model where all players are altruistic the value is always zero. Moreover, for models where there is at least one rational agent (which is the first player), that agent can take advantage of the possibility of cheating to increase that probability. Interestingly enough, if there is a byzantine agent, the rational agent can actually benefit from the fact that the byzantine agent may end up sending its share when not supposed to, and have a higher probability of being the only to learn the secret for smaller values of α, when comparing to a model where the two other agents are altruistic.



(a) α=0.3 (b) α=0.8

Figure 8.5: Expected utilities over a bounded number of rounds (p_{fail}=0.2 for *rba*).

Figure 8.5 plots the expected utilities of the agents when the protocol stops after a maximum number of rounds (r_{max}) when α=0.3 and α=0.8. The utilities converge more slowly for α=0.3, since when α is small there is a higher chance that an agent flips tails in a round meaning, not all agents will share their secret in this round and the protocol will move into another round. Again we see that there are more incentives for a rational agent to cheat as α gets closer to 1. However, when α=0.3 and there are altruistic agents, the incentive decreases and eventually disappears as the number of rounds increases. Multi-player properties and algorithms were essential for developing this case study, as the analysis could not be done by grouping two of the players in

a coalition. Moreover, although we do run zero-sum properties to compute minimum expected number of rounds and maximal probabilities for one of the players, the point of the original paper was to investigate incentive mechanisms so that a rational player would cheat, which then asks for considering equilibria properties.

8.1.3 Future Market Investors

The futures market investor model represents the interactions between investors and a stock market. For the original TSG version of the model described in [157], in successive months, a single investor chooses whether to invest, next the market decides whether to bar the investor, with the restriction that the investor cannot be barred two months in a row or in the first month, and then the values of shares and a cap on values are updated probabilistically.

We have built and analysed several CSG variants of the model, analysing optimal strategies for investors under adversarial conditions. First, we made a single investor and market take their decisions concurrently, and verified that this yielded no additional gain for the investor. This is because the market and investor have access to the same information, i.e. the stock prices, and so the market knows when it is optimal for the investor to invest without needing to see its decision. We next modelled two competing investors who simultaneously decide whether to invest (and, as above, the market simultaneously decides which investors to bar). If the two investors cash in their shares in the same month, then their profits are reduced. We also consider several distinct profit models: 'normal market', 'later cash-ins', 'later cash-ins with fluctuation' and 'early cash-ins'. The first is from [157] and the remaining reward either postponing cashing in shares or the early cashing in of shares.

The CSG has 3 players: one for each investor and one representing the market who decides on the barring of investors. We study both the maximum profit of one investor and the maximum combined profit of both investors. For comparison, we also build a TSG model in which the investors first take turns to decide whether to invest (the ordering decided by the market) and then the market decides on whether to bar any of the investors.

Figure 8.6 shows the maximum expected value over a fixed number of months under the 'normal market' for both the profit of first investor and the combined profit of the two investors. For the former, we show results for the first investor acting alone ($\langle\langle i1 \rangle\rangle$) and when in a coalition with the second investor ($\langle\langle i1, i2 \rangle\rangle$). We plot the corresponding results from the TSG model for comparison. Figure 8.7 shows the maximum expected combined profit for the other two profit models.

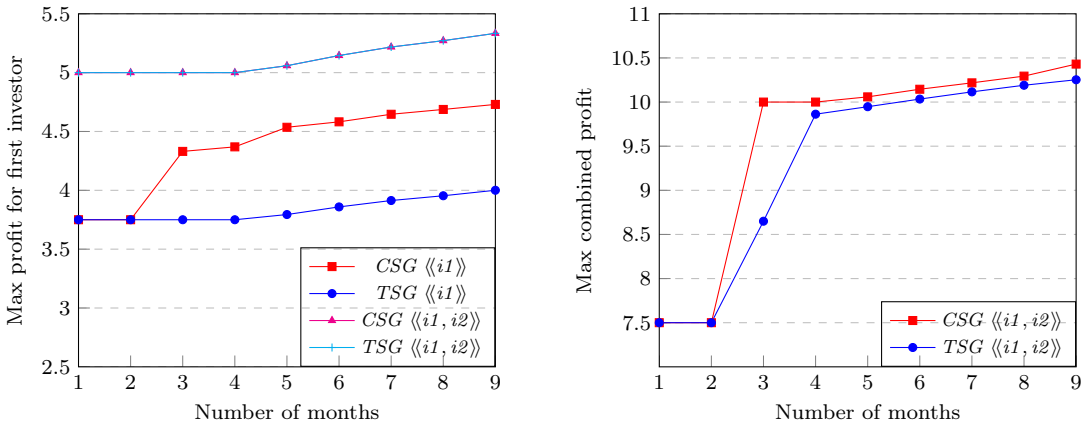


Figure 8.6: Futures market investors: normal market.

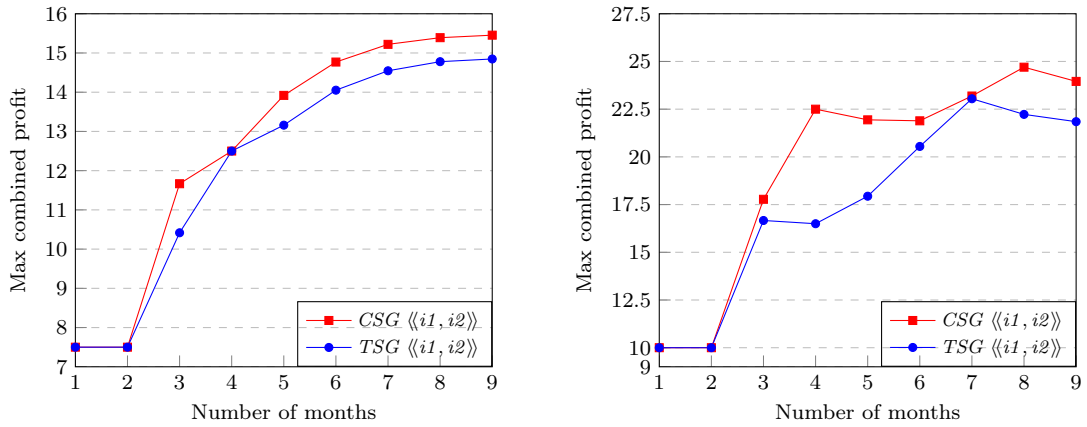


Figure 8.7: Futures market investors: later cash-ins without (left) and with (right) fluctuations.

When investors cooperate to maximise the profit of the first, results for the CSG and TSG models coincide. This follows from the discussion above since all the second investor can do is make sure it does not invest at the same time as the first. For the remaining cases and given sufficient months, there is always a strategy in the concurrent setting that outperforms all turn-based strategies. The increase in profit for a single investor in the CSG model is due to the fact that, as the investors decisions are concurrent, the second cannot ensure it invests at the same time as the first, and hence decrease the profit of the first. In the case of combined profit, the difference arises because, although the market knows when it is optimal for one investor to invest, in the CSG model the market does not know which one will, and therefore may choose the wrong investor to bar. On the other hand, in the TSG version, the market's choice is made after both investors have made their decisions and can therefore see which investor to block.

We performed strategy synthesis to study the optimal actions of investors. By way of example, consider $\langle\langle i1 \rangle\rangle \mathtt{R}^{profit1}_{\max=?}[\,\mathtt{F}^{c}\ cashed\_in_1\,]$ over three months and for a normal market (see Figure 8.6 left). The optimal TSG strategy for the first investor is to invest in the first month (which the market cannot bar) ensuring an expected profit of 3.75. The optimal (randomised) CSG strategy is to invest:

- in the first month with probability ∼0.4949;
- in the second month with probability 1, if the second investor has cashed in;
- in the second month with probability ∼0.9649, if the second investor did not cash in at the end of the first month and the shares went up;
- in the second month with probability ∼0.9540, if the second investor did not cash in at the end of the first month and the shares went down;
- in the third month with probability 1 (this is the last month to invest).

Following this strategy, the first investor ensures an expected profit of ∼4.33.

We now make the market probabilistic, where the probability the market bars an individual investor equals *pbar*, and consider nonzero-sum properties in which each investor tries to maximise their individual profit, for different reward structures. In Figure 8.8 and Figure 8.9 we have plotted the results for the investors where the profit models of the investors follow a normal profile and where the profit models of the investors differ ('later cash-ins' for the first investor and 'early cash-ins' for the second), when *pbar* equals 0.1 and 0.5 respectively. The results demonstrate that, given more time and a more predicable market, i.e., when *pbar* is lower, the players can collaborate to increase their profits.

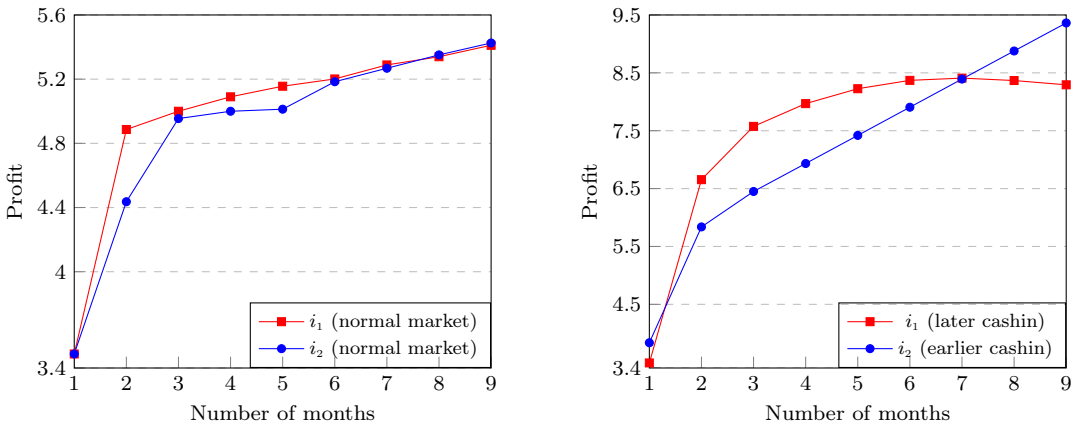


Figure 8.8: Investors: normal profiles (left) and mixed profiles (right) (*pbar*=0.1).

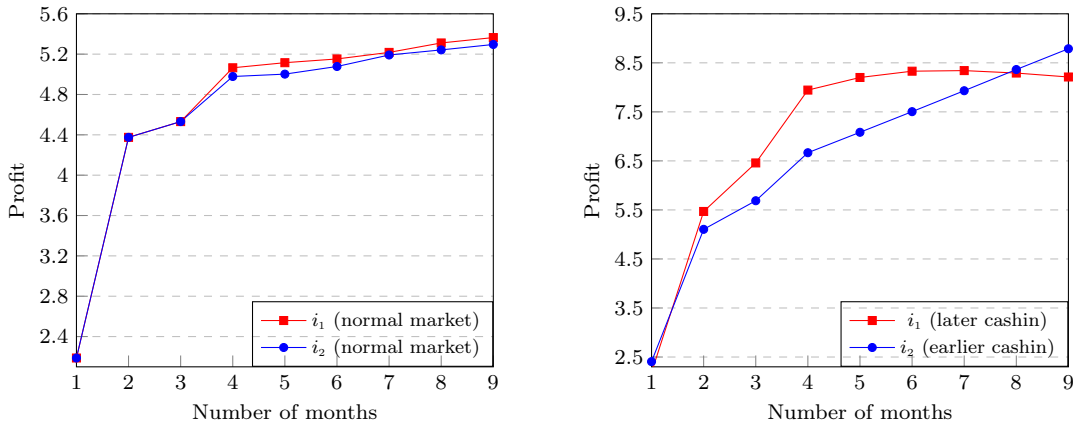


Figure 8.9: Investors: normal profiles (left) and mixed profiles (right) ($pbar$=0.5).

Performing strategy synthesis, we find that the strategies in the mixed profiles model are for the investor with an 'early cash-ins' profit model to invest as soon as possible, i.e., it tries to invest in the first month and if this fails because it is barred, it will be able to invest in the second. On the other hand, for the investor with the 'later cash-ins' profile, the investor will delay investing until the chances of the shares falling start to increase or they reach the month before last and then invest (if the investor is barred in this month, they will be able to invest in the final month).

8.1.4 Trust Models for User-centric Networks

Trust models for user-centric networks were analysed previously using TSGs in [142]. The analysis considered the impact of different parameters on the effectiveness of cooperation mechanisms between service providers. The providers share information on the measure of *trust* for users in a *reputation*-based setting. Each measure of trust is based on the service's previous interactions with the user (which previous services they paid for), and providers use this measure to block or allow the user to obtain services.

In the original TSG model, a single user can either make a request to one of three service providers or buy the service directly by paying maximum price. If the user makes a request to a service provider, then the provider decides to accept or deny the request based on the user's trust measure. If the request was accepted, the provider would next decide on the price again based on the trust measure, and the user would then decide whether to pay for the service and finally the provider would update its trust measure based on whether there was a payment. This sequence of steps would

have to take place before any other interactions occurred between the user and other providers.

Here we consider CSG models allowing the user to make requests and pay different service providers simultaneously and for the different providers to execute requests concurrently. There are 7 players: one for the user's interaction with each service provider, one for the user buying services directly and one for each of the 3 service providers. Three trust models were considered. In the first, the trust level was decremented by 1 (td=1) when the user does not pay, decremented by 2 in the second (td=2) and reset to 0 in the third (td=inf).

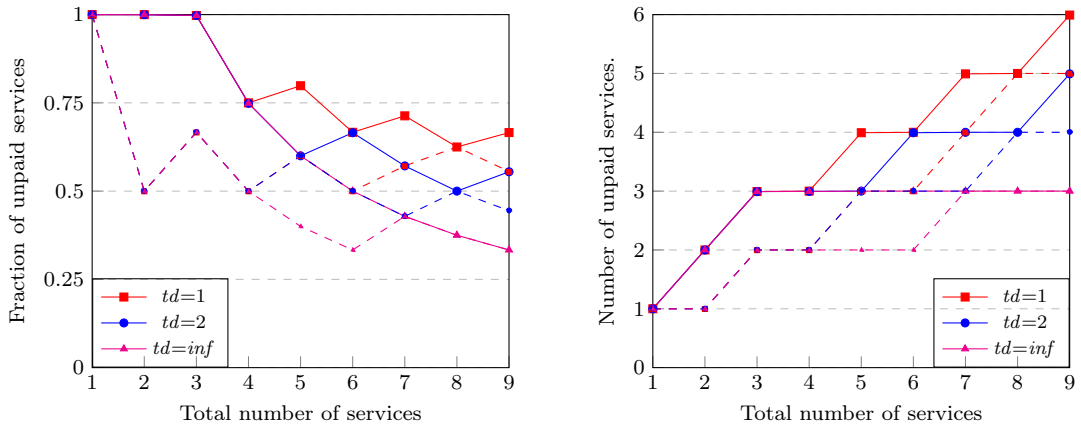


Figure 8.10: User-centric network results (CSG/TSG values as solid/dashed lines).

Figure 8.10 presents results for the maximum fraction and number of unpaid services the user can ensure for each trust model. The results for the original TSG model are included as dashed lines. They demonstrate that the user can take advantage of the fact that in the CSG model it can request multiple services at the same time, and obtain more services without paying before the different providers get a chance to inform each other about non-payment. In addition, the results show that having a more severe penalty on the trust measure for non-payment decreases the unpaid services the user can obtain.

8.1.5 Aloha Protocol

This case study concerns three users trying to send packets using the slotted ALOHA protocol [149]. In a time slot, if a single user tries to send a packet, there is a probability (q) that the packet is sent; as more users try and send, then the probability of success decreases. If sending a packet fails, the number of slots a user waits before resending is set according to an exponential backoff scheme. More precisely, each user

maintains a backoff counter which increases each time there is a failure (up to b_{max}) and, if the counter equals k, randomly chooses the slots to wait from $\{0, 1, \ldots, 2^k-1\}$.

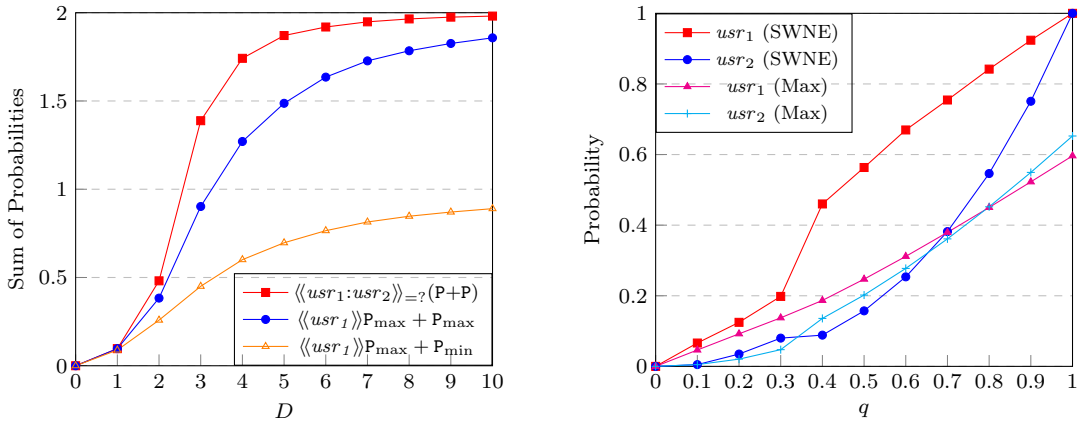


Figure 8.11: Aloha: $\langle\langle usr_1{:}\{usr_2, usr_3\}\rangle\rangle_{\max=?}(\mathtt{P}[\,\mathtt{F}\ (\mathtt{s}_1 \wedge t \leqslant D)\,] + \mathtt{P}[\,\mathtt{F}\ (\mathtt{s}_2 \wedge \mathtt{s}_3 \wedge t \leqslant D)\,])$

We suppose three users try to maximise the probability of sending packets before a deadline D, with users 2 and 3 forming a coalition. Figure 8.11 presents total values as D varies (left) and individual values as q varies (right). Through synthesis, we find the collaboration is dependent on D and q. Given more time there is a greater chance for the users to collaborate by sending in different slots, while if q is small, it is likely that players have to repeatedly try and send their packets; hence waiting for others to send in the limited slots is not advantageous. On the other hand, if q is large it is unlikely users need to repeatedly send, so again can send in different slots. As the coalition has more messages to send, their probabilities are lower. However, even for two users, the probabilities are different, since, although it is advantageous to collaborate and only one user tries first, if the transmission fails, then both users try to send as this is the best option for their individual goals.

We have also considered the case when the users try to minimise the expected time before their packets are sent, where again users 2 and 3 form a coalition. Synthesising the strategies we see that the players collaborate with the coalition of users 2 and 3, letting user 1 to try before sending their messages. However, if user 1 fails to send, then the coalition either lets user 1 try again in case the user can do so immediately, and otherwise the coalition tries to send their messages.

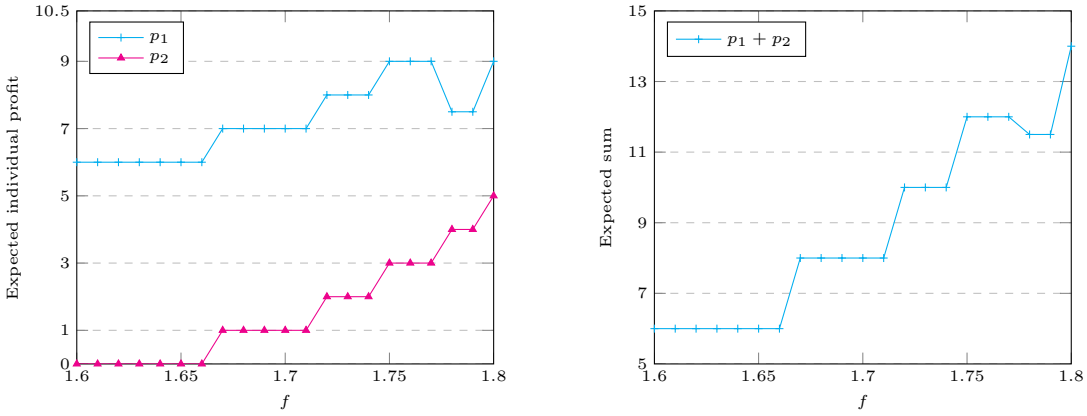
8.1.6 Public Good

We consider a *public good* game [111], in which each player receives a given amount of capital (e_{init}) and, over k rounds, the players can (partially) invest their capital in a common stock. In each round, the total invested is multiplied by a factor f and

distributed equally among the players. The aim of the players is to maximise their final capital. The capital return at the end of each round for player i is calculated as $(c_i - l_i) + f/n \cdot (\sum_{i=1}^{n} l_i)$, where:

- c_i is player i's capital at the start of the round;
- l_i is the amount invested by player i in that round;
- f is a multiplying factor;
- n is the number of players.

Figure 8.12 shows the reward values for a two-player model, where players can invest none, a quarter, half, three quarters or the entirety of their capitals. It is possible to notice the impact of small increments of f in the player's willingness to invest and their profits, illustrating the subtleties of tuning incentive mechanisms.



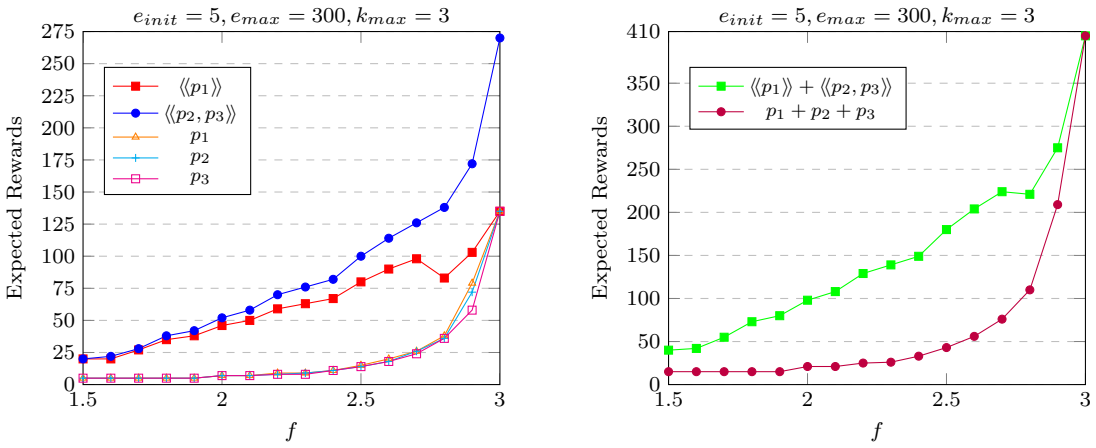
(a) Sum of rewards.

(b) Individual rewards.

Figure 8.12: Public good game: Impact of small increments in parameter f.

Figure 8.13 presents results for a 3-player public good game, in which a player can invest none, half or all of their current capital in each round. The results compare, as f varies, the expected utilities when the players act in isolation and when player 1 acts in isolation and players 2 and 3 form a coalition.

When the players act in isolation, for $f \leqslant 2$ there is no incentive for the players to invest. As f increases, the players start to invest some of their capital in some of the rounds, and when f=3 each player invests all their capital in each round. On the other hand, when players 2 and 3 act in a coalition, there is incentive to invest capital for smaller values of f, as players 2 and 3 can coordinate their investments to ensure they both profit; however player 1 also gains from these investments, and therefore has



(a) Individual rewards.

(b) Sum of rewards.

Figure 8.13: Public good game: Expected rewards after k_{max} rounds.

no incentive to invest in the final round. As f increases, there is a greater incentive for player 1 to invest and all the players' final capital increases. The drop in the capital of player 1, as f increases, is caused by players 2 and 3 coordinating against player 1 and decreasing their investments. This forces player 1 to invest to increase its investment which, as profits are shared, also increases the capital of players 2 and 3.

8.1.7 Medium Access Control

In this subsection, we extend our discussion on the model access control problem presented in Chapter 7 (Figures 7.1 and 7.3). In [33] a deterministic concurrent game is used to model medium access control. Two users with limited energy share a wireless channel and choose to *transmit* or *wait* and, if both transmit, the transmissions fail due to interference. Users can only choose to transmit they have not yet successfully transmitted all their messages and not used up their limited energy supply. Trying to transmit a packet uses up one unit of energy.

Figure 8.14 presents results for these properties as the bound k varies. For both properties, the SWNE strategies yield equal values for the players. Synthesising strategies, we see that for small values of k there is not sufficient time to collaborate (both users always try and transmit); however, as k increases there is time for the users to collaborate and try to transmit in different slots, and hence improve their values. Since the users have limited energy, Figure 8.14 shows that eventually adding steps does not increase the reward or probability. The graph of Figure 7.4 demonstrates that, in the synthesised strategy profile, the players cooperate to achieve their

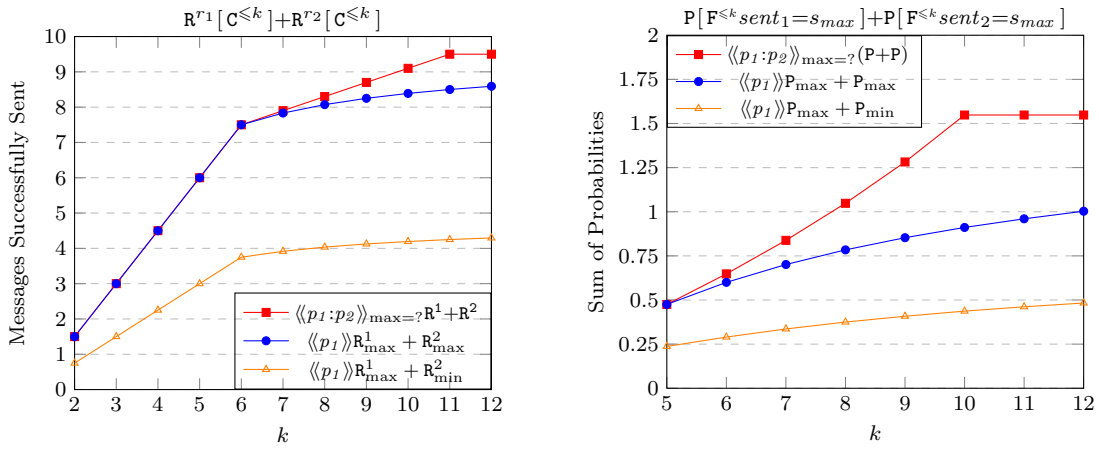


Figure 8.14: Medium access control (e_{max}=5, s_{max}=5, q_1=0.9 and q_2=0.75)

individual goals. More precisely, player 1 waits while first player 2 tries to transmit until either its goal has been reached, i.e. two successful transmissions, or it does not have sufficient energy to meet its objective, i.e. either the first transmission fails and it has only the energy to try and transmit one more time or the first succeeds and then second transmission fails after which it has run out of energy. After this player 2 no longer tries to transmit and player 1 tries to transmit until again either its goal has been reached or it does not have sufficient energy to meet its objective. This can also be seen in the SWNE values of (0.81,0.81), as 0.81 corresponds to a player trying to transmit twice and being successful each time and the other player not transmitting at these times.

8.1.8 Power Control

This case study is based on a model of power control in cellular networks from [33]. In the model, phones emit signals over a cellular network and the signals can be strengthened by increasing the power level up to a bound (pow_{max}). A stronger signal can improve transmission quality, but uses more energy and lowers the quality of other transmissions due to interference. We extend this model by adding a failure probability (q_{fail}) when a power level is increased and assume each phone has a limited battery capacity (e_{max}). Based on [33], we associate a reward structure with each phone representing transmission quality dependent both on its power level and that of other phones due to interference. Each user has a reward structure representing the transmission quality of the phone, where the quality improves if the phone's power level is increased. However, the quality decreases as the other phone's power level is

increased. If a phone has run out of energy, then the phone has no influence on the other phone's transmission quality.

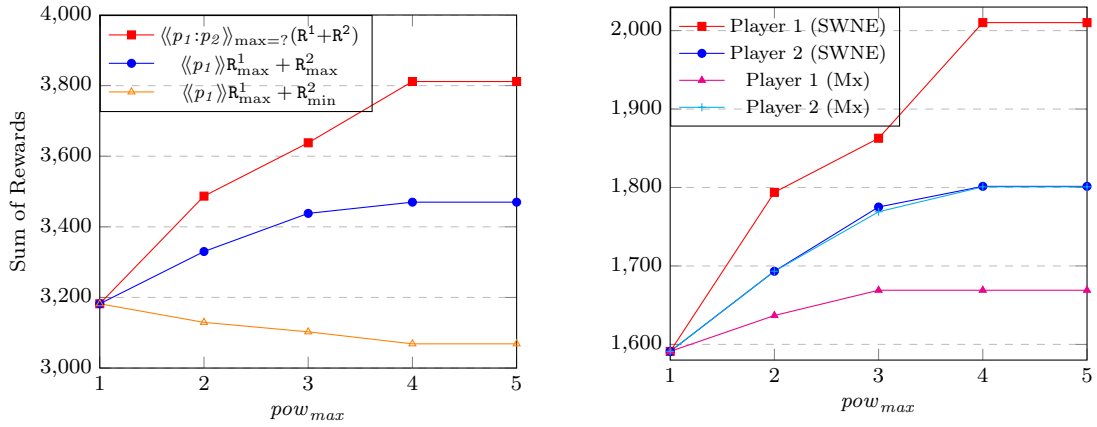


Figure 8.15: Power control: $\langle\langle p_1{:}p_2\rangle\rangle_{\max=?}(\mathtt{R}^{r_1}[\,\mathtt{F}\ e_1{=}0\,]{+}\mathtt{R}^{r_2}[\,\mathtt{F}\ e_2{=}0\,])$

We consider two players, each trying to maximise their reward before their battery is empty. Figure 8.15 presents, for e_{max}=10, the sum of the SWNE values (left) and the values of the individual players (right) as pow_{max} varies. The values of the players are different because if one increases their power level this increases the overall reward (their reward increases, while that of the other's decreases by a lesser amount due to interference), whereas if both increase their power level the overall reward decreases (both rewards decrease due to interference).

Synthesising the optimal strategies for the players we find that the values of the players are different because, if one increases their power level, this increases the overall reward (their reward increases, while the other's decreases by a lesser amount due to interference) whereas, if both increase, the overall reward decreases (both rewards decrease due to interference).

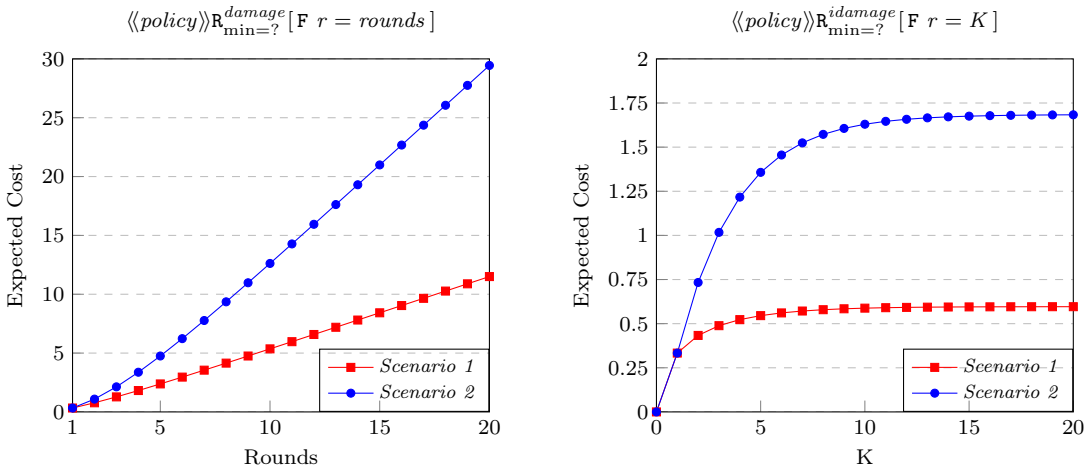
8.1.9 Intrusion Detection Policies

In [219], CSGs are used to model the interaction between an intrusion detection policy and attacker. The policy has a number of libraries it can use to detect attacks and the attacker has a number of different attacks which can incur different levels of damage if not detected. Furthermore, each library can only detect certain attacks. In the model, in each round the policy chooses a library to deploy and the attacker chooses an attack.

In the model, the system can be in 3 possible states: *healthy*, *compromised* and *failure*. Initially, the system is in the healthy state and remains in that state provided

the correct defence is chosen. If the wrong defence is chosen, it moves to a compromised with probability *attack* and otherwise remains in the healthy state. From the compromised state, if the correct defence is chosen it goes to the healthy state with probability *recover* and otherwise remains in the compromised state. If the wrong defence is chosen from the compromised state, it goes to the failure state with probability *fail* and otherwise remains in the compromised state. Finally, from the failure state, if the correct defence is chosen it goes to the compromised state with probability *repair* and otherwise remains in the failure state. If the wrong defence is chosen from the failure state, the system does not change its state.

A reward structure is specified representing the level of damage when an attack is not detected. Each time the wrong defence library is chosen, a cost is incurred. If the system is already in the compromised or failure state, the cost is higher. The goal is to find optimal intrusion detection policies which correspond to finding a strategy for the policy that minimises damage. We have constructed CSG models with 2 players (representing the policy and the attacker) for the two scenarios outlined in [219].



(a) Minimum expected cumulative cost.

(b) Minimum expected cost at K.

Figure 8.16: Intrusion detection policies: min. expected cost cumulative (left) and at a given round (right).

In scenario 1, there's a small chance an attack is successful (*attack*=0.2), a high chance of recovering if attack is prevented (*recover*=0.8), a small chance of failure from a compromised state when attack continues (*fail*=0.2) and a high chance of repairing itself, and then going from a failure state to a compromised state, when attack stops (*repair*=0.8). In scenario 2, we flip the values of attack/recover and fail/repair.

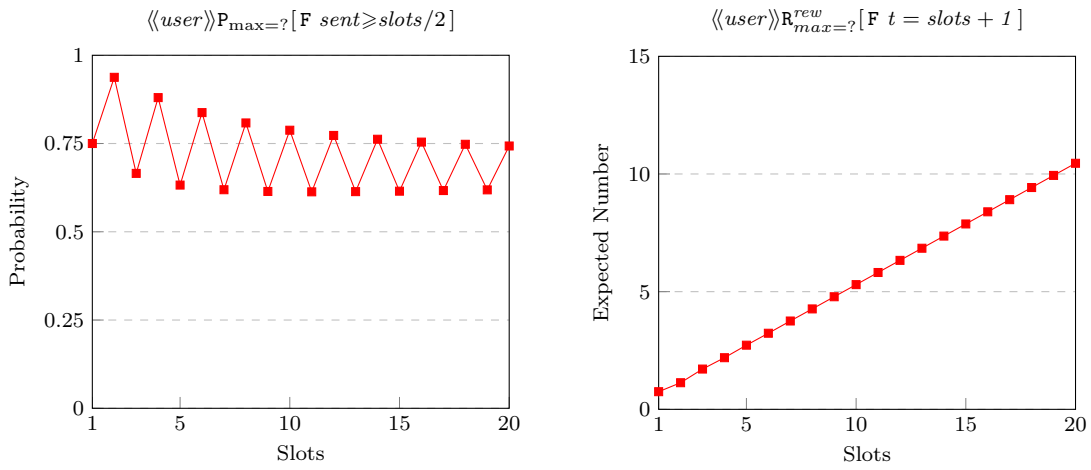
Figure 8.16 shows the plots for the minimum expected cumulative cost over a number of rounds and the minimum expected cost at a given round K, considering a measure of undetected attacks. It can be seen that the two reflect each other as the rate of the cumulative cost grows or stabilises so does the instantaneous cost. It is also possible to notice that, as expected, the associated cost in scenario 1 is considerably smaller given the smaller rates of successful attacks and failures and the higher rates of recover and repair.

We have synthesised optimal policies which ensure the minimum cumulative damage over a fixed number of rounds and damage in a specific round. In each case, the optimal strategy for the system is randomised. In addition, it favours the second defence which can be attributed to the fact that the second defence has at least partial success against both attacks, while this is not true of the first. Here, concurrency is required for the game to be meaningful, otherwise it is easy for the player whose turn follows the other player to "win". For example, if the attacker knows what library is being deployed, then it can simply choose an attack the library cannot detect.

8.1.10 Jamming Multi-channel Radio Systems

A CSG model for jamming multi-channel cognitive radio systems is presented in [220]. The system consists of a number of channels (*chans*), which can be in an *idle* or *occupied* state. A user can only send a message successfully if a channel is idle. The state of each channel remains fixed within a time slot and between slots is Markovian (i.e. the state changes randomly based only on the state of the channel in the previous slot). A secondary user has a subset of available channels and at each time-slot must decide which to use. There is a single attacker, which again has a subset of available channels and at each time slot decides to send a jamming signal over one of them. Again, the ability to model concurrence is crucial here given that, if modelled as a turn-based game, it would be easy for the player that follows the other's move to pick a winning action. For instance, if the attacker was able to see which channel would be used, you could simply choose to jam that channel. The CSG has 2 players: one representing the secondary user and one for the attacker.

We considered a model with a total of 4 channels, where the user is able to use the first three and the attacker is able to jam the last three. Figure 8.17 shows the results for the maximum probability a user can ensure at least half of the messages are sent and the maximum expected number of messages sent. At each time slot, there is a 75% probability that each channel changes its state (e.g. moving from *idle* to *occupied*) and initially al channels are occupied.



(a) Max. prob. half of the messages are sent.

(b) Max. number of messages sent.

Figure 8.17: Jamming multi-channel radio systems: maximum probability half of the messages are sent (left) and maximum number of messages sent (right).

We synthesise strategies for the secondary user, which maximise the probability of ensuring at least half the messages are sent correctly. For the case where there are only two slots, we see that the optimal behaviour of the user is to initially send on channel 1 (as there is a high chance it will be idle in the next slot and the attacker cannot jam this channel). In the next slot, the user should again send on channel 1, unless it was occupied in the previous slot and both channels 2 and 3 were idle. In this case, the user should choose between channels 2 and 3 uniformly at random. The reason is that, in this case, there is only a small chance channel 1 will be idle when the message is sent and a high chance both that channel 2 and channel 3 will be idle and the attacker cannot jam both these channels. In all other cases there is either: (i) a high chance that channel 1 will be idle and this channel cannot be jammed; (ii) a high chance only channel 2 or channel 3 will be idle and, since the attacker knows this, the likely to be idle channel will be jammed, and so it is better to send on channel 1 even though there is only a low chance it will be idle.

8.2 Efficiency and Scalability

In this section, we demonstrate the performance and scalability of our methods on the case studies in this chapter. We begin by presenting a selection of results illustrating the performance of our implementation. The experiments for zero-sum and two-player equilibria properties were run on a 2.10 GHz Intel Xeon with 16GB of JVM memory.

The precision set for convergence of value iteration for all experiments was 10^{-6}. In Table 8.3, we give the model statistics for the examples used: the number of players, states, transitions and model construction times. Due to improvements in the modelling language and the model building procedure, some of the model statistics differ from those presented in [136, 137]. This is due to fact that the earlier version of the implementation did not allow for variables of different players to be updated following a joint probability distribution, which made it necessary to introduce intermediate states in order to specify some of the behaviour. This section also includes results for additional properties compared to [138], as we work with an extended version of the logic.

Tables 8.4 and 8.6 present the model checking statistics when analysing zero-sum and two-player nonzero-sum properties respectively. In both tables, this includes the maximum and average number of actions of each coalition in the matrix/bimatrix games solved at each step of value iteration and the number of iterations performed. In the case of zero-sum properties including reward formulae of the form $\texttt{F}\ \phi$, value iteration is performed twice (see Section 4.5.2), and therefore the number of iterations for each stage are presented (and separated by a semi-colon). For zero-sum properties, the timing statistics are divided into the time for qualitative and quantitative verification (including solving matrix games). For nonzero-sum properties we divide the timing statistics into the time for CSG verification (including solving bimatrix games) and the instances of MDP verification. In the case of mixed nonzero-sum properties, i.e., properties including both finite and infinite horizon objectives, we must first build a new game (see Section 5.3.5) and the statistics for these CSGs are presented in Table 8.7.

The implementation can analyse models with over 3 million states and almost 18 million transitions; all are solved in under 2 hours and most are considerably quicker. The majority of the time is spent solving matrix or bimatrix games so, as well as the number of states, it is the number of choices of each coalition, rather than the number of players, that affects performance. For example, larger instances of the Aloha models are verified relatively quickly since the coalitions have only one choice in many states (the average number of choices is 1.00 for both coalitions). However, for models where players have choices in almost all states, only models with up to hundreds of thousands of states for zero-sum properties and tens of thousands of states for nonzero-sum properties can be verified within 2 hours.

Table 8.2 presents experimental results for the time to solve bimatrix games using the Yices and Z3 solvers, as the numbers of actions of the individual games vary. The

Bimatrix Game	Actions of each player	SWNE values			SCNE values		
		Solution time (s)		Num. of NE in N	Solution time (s)		Num. of NE in N^-
		Yices	Z3		Yices	Z3	
Covariant games	2	0.04	0.06	1	0.2	1.1	1
	4	0.04	0.1	3	0.04	0.1	3
	8	0.3	2.7	13	0.3	1.8	21
	12	8.3	77.0	15	11.0	130.2	45
	16	764.8	4,238	103	318.6	2,627	109
Dispersion games	2	0.04	0.08	3	0.04	0.08	3
	4	0.05	0.2	51	0.04	0.1	15
	8	1.8	16.4	6,051	0.2	1.1	255
	12	6,368	t/o	523,251	6.0	38.3	4,095
Majority voting games	2	0.03	0.08	5	0.03	0.07	2
	4	0.05	0.1	15	0.04	0.1	13
	8	0.5	1.0	433	0.2	0.7	186
	12	9.9	22.4	3,585	9.9	16.0	3,072
	16	532.5	2,386	61,441	465.9	2,791	49,153
Randomly generated games	2	0.15	1.9	1	0.03	0.08	1
	4	0.04	0.1	3	0.03	0.09	3
	8	0.4	2.3	13	0.3	1.8	7
	12	45.60	422.0	27	68.0	343.6	31
	16	2,370	t/o	81	1,112	t/o	69

Table 8.1: Finding SWNE/SCNE values in bimatrix games: comparing SMT solvers.

table also shows the number of NE in each game N, as found when determining the SWNE values, and also the number of NE in N^-, as found when determining the SCNE values (see Lemma 1). These games were generated using GAMUT (a suite of game generators) [168] and a time-out of 2 hours was used for the experiments. The results show Yices to be the faster implementation and that the difference in solution time grows as the number of actions increases. Therefore, in our experimental results in the next section, all verification runs use the Yices implementation. The results in Table 8.2 also demonstrate that the solution time for either solver can vary widely and depends on both the number of NE that need to be found and the structure of the game. For example, when solving the dispersion games, the differences in the solution times for SWNE and SCNE seem to correspond to the differences in the number of NE that need to found. On the other hand, there is no such correspondence between the difference in the solution times for the covariant games.

Table 8.1 presents experimental results when solving NFGs (which were also generated using GAMUT [168]) through support enumeration using Z3 for filtering out unsatisfiable support assignments (with a timeout of 20ms) and Ipopt. For each NFG the table lists the numbers of players, actions of each player and support assignments. The table also includes the supports of each type returned by Z3 and the solution time of Ipopt. As can be seen, by using Z3 we have significantly reduced the assignments Ipopt needs to analyse, by orders of magnitude in some cases. However, as the number of actions grows, the number of assignments that remain for Ipopt

Game	Players	Actions	Supports	Supports returned by Z3			Time(s)
				unsat	*sat*	*unknown*	IPOPT
Majority voting games	3	3,3,3	343	330	12	1	0.309
	3	4,4,4	3,375	3,236	110	29	18.89
	3	5,5,5	29,791	26,250	155	3,386	336.5
	4	2,2,2,2	81	59	22	0	0.184
	4	3,3,3,3	2,401	2,212	87	102	6.847
	4	4,4,4,4	50,625	41,146	518	8,961	1,158
	5	2,2,2,2,2	243	181	62	0	0.591
	5	3,3,3,3,3	16,807	14,950	266	1,591	253.3
Covariant gamem	3	3,3,3	343	304	6	33	7.645
	3	4,4,4	3,375	2,488	16	871	203.8
	3	5,5,5	29,791	14,271	8	15,512	5,801
	4	2,2,2,2	81	76	3	2	0.106
	4	3,3,3,3	2,401	1,831	0	570	183.0
	5	2,2,2,2,2	243	221	8	14	4.128
	5	3,3,3,3,3	16,807	6,600	7	10,200	5,002

Table 8.2: Finding SWNE in NFGs (timeout of 20ms for Z3).

to solve increases rapidly, and therefore also does the solution time. Furthermore, increasing the number of players only magnifies this issue. The solution times for NFGs may appear to be slow; however just finding a NE is a difficult problem, while here we require SWNE. For example, in [174], using a backtracking search algorithm or either of the Simplicial Subdivision [202] and the Govindan-Wilson [97] algorithms for finding a sample NE, there are instances of NFGs with 6 player and 5 actions that timeout after 30 minutes.

Table 8.8 presents a selection of results demonstrating the performance for model checking of multi-player nonzero-sum properties. The results for multi-player equilibria properties were carried out using a 2.10GHz Intel Xeon Gold with 16GB of JVM memory. The table includes statistics for the models: number of players, states, (maximum) actions for each player in a state, transitions and the times to both build and verify the models. All models have been verified in under 2 hours and in most cases much less than this. The largest model, verified in under 15 minutes, has 4 players, almost 1.5 million states and 5 million transitions. The majority of the time is spent solving NFG games and, as shown in Table 8.1, this varies depending on the number of choices and players.

8.3 Summary

In this chapter, we presented a range of case studies that were modelled as stochastic games and verified against rPATL properties with PRISM-games 3.0. We have seen applications that could not be analysed without the possibility of modelling concurrency, compared concurrent and turn-based variants of the same models, compared

zero-sum and equilibria strategies, and considered scenarios in which it was necessary to have methods to handle multi-player properties. We noticed in case studies such as the *Future Market Investors* that CSGs makes it possible to have a more faithful translation of a given problem when comparing to TSGs, as well as computing more accurate values and synthesising better strategies. A similar statement could be made about the *Trust Models for User-centric Networks* case study. Subsequently, when studying applications such as the *Robot Coordination* and the *Aloha Protocol*, it was possible to notice the advantages of considering equilibria properties to synthesise mutually beneficial stable strategies instead of assuming strictly competitive scenarios. Finally, we were able to highlight in applications such as the *Secret Sharing Protocol* and the *Public Good Game* scenarios, which required multi-player capabilities, the additional aspects that multi-player equilibria analysis enables compared to the two-player.

Case study [parameters]	Param. values	Players	States	Transitions	Constr. time (s)
Robot coordination [l]	4	2	226	6,610	0.1
	8	2	3,970	201,650	1.0
	12	2	20,450	1,221,074	3.9
	16	2	65,026	4,198,450	11.8
	24	2	330,626	23,049,650	62.5
Future markets investors [*months*]	6	3	33,338	143,487	1.4
	12	3	254,793	1,257,112	7.7
	24	3	826,617	4,315,864	24.7
	36	3	1,398,441	7,374,616	48.8
	48	3	1,970,265	10,433,368	65.5
Future markets investors [*months*,*pbar*]	3,0.5	2	3,268	11,820	0.27
	6,0.5	2	28,794	123,839	1.2
	12,0.5	2	219,205	1,080,516	6.8
	18,0.5	2	464,989	2,394,276	15.3
User-centric networks [td, K]	1,3	7	32,214	121,659	2.1
	1,4	7	104,897	433,764	6.1
	1,5	7	294,625	1,325,100	17.5
	1,6	7	714,849	3,465,558	42.8
Aloha (deadline) [b_{max},D]	1,8	3	3,519	5,839	0.2
	2,8	3	14,230	28,895	0.5
	3,8	3	72,566	181,438	2.1
	4,8	3	413,035	1,389,128	9.4
	5,8	3	2,237,981	9,561,201	58.4
Aloha [b_{max}]	2	3	5,111	10,100	0.3
	3	3	22,812	56,693	0.8
	4	3	107,799	355,734	2.5
	5	3	556,168	2,401,113	13.4
	6	3	3,334,681	17,834,254	118.8
Intrusion detection system [*rounds*]	25	2	75	483	0.07
	50	2	150	983	0.08
	100	2	300	1,983	0.1
	200	2	600	3,983	0.1
Jamming radio systems [*chans*,*slots*]	4,6	2	531	45,004	0.5
	4,12	2	1,623	174,796	1.4
	6,6	2	1,061	318,392	2.0
	6,12	2	3,245	1,240,376	6.6
Medium access control [e_{max}]	10	3	10,591	135,915	1.7
	15	3	33,886	457,680	5.0
	20	3	78,181	1,083,645	8.2
	25	3	150,226	2,115,060	14.0
Medium access control [e_{max},s_{max}]	4,2	3	14,723	129,097	2.2
	4,4	3	18,751	147,441	4.3
	6,4	3	122,948	1,233,976	11.0
	6,6	3	138,916	1,315,860	12.5
Power control [e_{max}, pow_{max}]	40,8	2	32,812	260,924	1.7
	80,8	2	193,396	1,469,896	6.5
	40,16	2	34,590	291,766	1.6
	80,16	2	301,250	2,627,278	10.5

Table 8.3: Model statistics for the CSG case studies.

Case study & property [parameters]	Param. values	Actions max/avg	Val. iters	Verif. time (s)	
				Prec.	Quant.
Robot coordination $\langle\langle rbt_1 \rangle\rangle \mathtt{P}_{\max=?}[\,\neg \mathtt{c}\, \mathtt{U}^{\leqslant k}\, \mathtt{g}_1\,]$ $[l,k]$	4,4	3,3/2.52,2.52	4	0.02	0.04
	8,8	3,3/2.52,2.52	8	0.27	1.1
	12,12	3,3/2.68,2.68	12	1.61	5.7
	16,16	3,3/2.76,2.76	16	5.88	35.4
	24,24	3,3/2.84,2.84	24	46.5	185.2
Robot coordination $\langle\langle rbt_1 \rangle\rangle \mathtt{R}_{\min=?}[\,\mathtt{F}\, \mathtt{g}_1\,]$ $[l]$	4	3,3/2.52,2.52	10; 9	0.02	0.1
	8	3,3/2.52,2.52	15; 14	0.35	4.1
	12	3,3/2.68,2.68	20; 19	1.99	23.2
	16	3,3/2.76,2.76	24; 24	7.57	96.5
	24	3,3/2.84,2.84	34; 33	56.9	744.7
Future markets investors $\langle\langle i_1 \rangle\rangle \mathtt{R}_{\max=?}[\,\mathtt{F}^c\ \mathsf{cashed}_1\,]$ $[months]$	6	2,8/1.22,1.55	14	1.5	4.4
	12	2,8/1.27,1.68	26	12.1	63.0
	24	2,8/1.29,1.73	50	42.5	344.4
	36	2,8/1.29,1.74	72	70.6	824.7
	48	2,8/1.29,1.75	80	99.6	1,424
User-centric networks $\langle\langle user \rangle\rangle \mathtt{R}_{\max=?}[\,\mathtt{F}^c\ services{=}K\,]$ $[td, K]$	1,3	16,8/2.11,1.91	15	3.3	172.5
	1,4	16,8/2.31,1.92	21	10.3	646.9
	1,5	16,8/2.46,1.94	25	37.6	2,311
	1,6	16,8/2.60,1.96	29	92.9	5,904
Aloha (deadline) $\langle\langle usr_2, usr_3 \rangle\rangle \mathtt{P}_{\max=?}[\,\mathtt{F}\, \mathtt{s}_{1,2} \wedge t \leqslant D\,]$ $[b_{max}, D]$	2,8	4,2/1.01,1.00	24	0.5	0.8
	3,8	4,2/1.01,1.00	23	1.8	1.8
	4,8	4,2/1.01,1.00	23	6.5	4.5
	5,8	4,2/1.01,1.00	23	35.9	9.4
Aloha $\langle\langle usr_2, usr_3 \rangle\rangle \mathtt{R}_{\min=?}[\,\mathtt{F}\, \mathsf{sent}_{2,3}\,]$ $[b_{max}]$	2	4,2/1.01,1.00	58; 47	0.2	2.1
	3	4,2/1.01,1.00	71; 57	0.6	6.0
	4	4,2/1.01,1.00	109; 86	3.7	33.7
	5	4,2/1.01,1.00	193; 150	26.6	317.6
	6	4,2/1.01,1.00	362; 279	314.5	3,836
Intrusion detection system $\langle\langle policy \rangle\rangle \mathtt{R}_{\min=?}[\,\mathtt{C}\,]$ $[rounds]$	25	2,2/1.96,1.96	26; 26	0.03	0.2
	50	2,2/1.98,1.98	51; 51	0.06	0.7
	100	2,2/1.99,1.99	101; 101	0.2	2.5
	200	2,2/2.00,2.00	201; 201	0.5	7.0
Jamming radio systems $\langle\langle user \rangle\rangle \mathtt{P}_{\max=?}[\,\mathtt{F}\ sent \geqslant slots/2\,]$ $[slots]$	4,6	3,3/2.17,2.17	7	0.03	0.1
	4,12	3,3/2.49,2.49	13	0.2	0.5
	6,6	4,4/2.76,2.76	7	0.1	0.3
	6,12	4,4/3.24,3.24	13	0.5	2.0

Table 8.4: Statistics for CSG zero-sum verification instances.

Case study & property [parameters]	Param. values	Inner Formula		Outer Formula	
		Val. iters.	Verif. time (s)	Val. iters.	Verif. time (s)
Robot coordination $\langle\langle rbt_1 \rangle\rangle \mathtt{P}_{\max=?}[\,\mathtt{F}\ \phi\,]$ $\phi{=}\langle\langle rbt_2 \rangle\rangle \mathtt{R}_{\geqslant 10}[\,\mathtt{F}\, \mathtt{g}_2\,]$ $[l]$	4	12; 12	0.1	5	0.1
	8	22; 22	5.0	10	0.8
	12	31; 30	40.5	10	2.6
	16	40; 40	187.7	11	6.6
	24	58; 58	1,235	12	22.6
Robot coordination $\langle\langle rbt_1, rbt_2 \rangle\rangle \mathtt{P}_{\min=?}[\,\mathtt{F}\ \phi\,]$ $\phi{=}\langle\langle rbt_1{:}rbt_2 \rangle\rangle_{\min \leqslant 5}(\mathtt{R}[\,\mathtt{F}\, \mathtt{g}_1\,]{+}\mathtt{R}[\,\mathtt{F}\, \mathtt{g}_2\,])$	4	8	0.1	24	0.1
	8	17	20.8	18	1.4
	12	32	527.3	28	8.4
	16	39	4,664	37	24.9
Aloha $\langle\langle usr_1, usr_2, usr_3 \rangle\rangle \mathtt{P}_{\max=?}[\,\mathtt{F}\ \phi\,]$ $\phi{=}\langle\langle usr_1{:}usr_2, usr_3 \rangle\rangle_{\min \geqslant 2}(\mathtt{P}[\,\mathtt{F}\, \mathtt{s}_1\,]{+}\mathtt{P}[\,\mathtt{F}\, \mathtt{s}_{2,3}\,])$ $[b_{\max}, D]$	1,8	23	0.6	24	0.3
	2,8	23	1.4	23	0.8
	3,8	22	7.9	23	2.8
	4,8	22	39.8	23	9.3
	5,8	22	172.6	23	37.1

Table 8.5: Statistics for CSG nested properties verification instances.

Case study & property [parameters]	Param. values	Actions max/avg	Val. iters.	Verif. time (s) MDP	CSG
Robot coordination $\langle\langle rbt_1{:}rbt_2\rangle\rangle_{\max=?}(\mathtt{P}[\neg \mathtt{c}\,\mathtt{U}^{\leqslant k}\,\mathtt{g}_1]+\mathtt{P}[\neg \mathtt{c}\,\mathtt{U}^{\leqslant k}\,\mathtt{g}_2])$ [l,k]	4,4	3,3/2.07,2.07	4	0.02	0.1
	8,8	3,3/2.52,2.52	8	0.45	1.0
	12,12	3,3/2.68,2.68	12	6.25	5.8
	16,16	3,3/2.76,2.76	16	34.7	22.2
	24,24	3,3/2.84,2.84	24	375.2	1,365
Robot coordination $\langle\langle rbt_1{:}rbt_2\rangle\rangle_{\max=?}(\mathtt{P}[\neg \mathtt{c}\,\mathtt{U}^{\leqslant k}\,\mathtt{g}_1]+\mathtt{P}[\neg \mathtt{c}\,\mathtt{U}\,\mathtt{g}_2])$ [l,k]	4,8	3,3/2.10,2.04	21	0.16	3.3
	4,16	3,3/2.12,2.05	21	0.28	12.0
	8,8	3,3/2.53,2.51	34	5.51	77.2
	8,16	3,3/2.54,2.52	35	17.2	3,513
Robot coordination $\langle\langle rbt_1{:}rbt_2\rangle\rangle_{\min=?}(\mathtt{R}[\mathtt{F}\,\mathtt{g}_1]+\mathtt{R}[\mathtt{F}\,\mathtt{g}_2])$ [l]	4	3,3/2.07,2.07	8	0.05	0.2
	8	3,3/2.52,2.52	15	0.44	10.3
	12	3,3/2.68,2.68	23	2.12	227.7
	16	3,3/2.84,2.84	28	12.1	3,272
Future markets investors $\langle\langle i_1{:}i_2\rangle\rangle_{\max=?}(\mathtt{R}[\mathtt{F}\,\mathtt{c}_1]+\mathtt{R}[\mathtt{F}\,\mathtt{c}_2])$ [*months*]	3	2,2/1.15,1.15	6	0.06	0.2
	6	2,2/1.23,1.23	13	0.8	3.0
	12	2,2/1.28,1.28	25	6.9	265.4
	18	2,2/1.29,1.29	37	26.6	3,997
Aloha (deadline) $\langle\langle usr_1{:}usr_2{,}usr_3\rangle\rangle_{\max=?}(\mathtt{P}[\mathtt{F}\,\mathtt{s}_1]+\mathtt{P}[\mathtt{F}\,\mathtt{s}_{2,3}])$ [b_{max},D]	1,8	2,4/1.00,1.01	23	0.1	0.3
	2,8	2,4/1.00,1.00	23	0.4	1.2
	3,8	2,4/1.00,1.00	22	2.0	3.4
	4,8	2,4/1.00,1.00	22	7.1	18.5
	5,8	2,4/1.00,1.00	22	39.9	103.0
Aloha $\langle\langle usr_1{:}usr_2{,}usr_3\rangle\rangle_{\min=?}(\mathtt{R}[\mathtt{F}\,\mathtt{s}_1]+\mathtt{R}[\mathtt{F}\,\mathtt{s}_{2,3}])$ [b_{max}]	2	2,4/1.00,1.01	54	0.2	0.8
	3	2,4/1.00,1.00	62	0.8	3.1
	4	2,4/1.00,1.00	88	4.5	40.1
	5	2,4/1.00,1.00	145	35.3	187.7
	6	2,4/1.00,1.00	256	453.7	2,396
Medium access control $\langle\langle p_1{:}p_2{,}p_3\rangle\rangle_{\max=?}(\mathtt{R}[\mathtt{C}^{\leqslant k}]+\mathtt{R}[\mathtt{C}^{\leqslant k}])$ [e_{max},k]	10,25	2,4/1.91,3.63	25	0.0	65.8
	15,25	2,4/1.94,3.75	25	0.0	193.3
	20,25	2,4/1.95,3.81	25	0.0	394.8
	25,25	2,4/1.96,3.85	25	0.0	565.4
Medium access control $\langle\langle p_1{:}p_2{,}p_3\rangle\rangle_{\max=?}(\mathtt{P}[\mathtt{F}\,\mathtt{m}_1]+\mathtt{P}[\mathtt{F}\,\mathtt{m}_{2,3}])$ [e_{max},s_{max}]	4,2	2,4/1.70,2.88	10	0.61	96.2
	4,4	2,4/1.64,2.70	12	0.61	22.8
	6,4	2,4/1.77,3.12	17	7.40	1,639
	6,6	2,4/1.74,3.02	18	4.09	93.6
Medium access control $\langle\langle p_1{:}p_2{,}p_3\rangle\rangle_{\max=?}(\mathtt{P}[\mathtt{F}^{\leqslant k}\,\mathtt{m}_1]+\mathtt{P}[\mathtt{F}\,\mathtt{m}_{2,3}])$ [e_{max},s_{max},k]	4,4,4	2,4/1.67,2.70	12	1.64	39.1
	4,4,8	2,4/1.68,2.70	12	3.30	106.3
	6,4,6	2,4/1.76,3.02	18	23.4	341.9
	6,4,12	2,4/1.74,3.02	18	60.0	961.8
Power control $\langle\langle p_1{:}p_2\rangle\rangle_{\max=?}(\mathtt{R}[\mathtt{F}\,e_1{=}0]+\mathtt{R}[\mathtt{F}\,e_2{=}0])$ [e_{max},pow_{max}]	40,8	2,2/1.91,1.91	20	4.1	11.7
	80,8	2,2/1.88,1.88	40	34.8	130.3
	40,16	2,2/1.95,1.95	40	5.4	11.6
	80,16	2,2/1.98,1.98	80	64.4	211.4
Power control $\langle\langle p_1{:}p_2\rangle\rangle_{\max=?}(\mathtt{R}[\mathtt{F}\,e_1{=}0]+\mathtt{R}[\mathtt{C}^{\leqslant k}])$ [e_{max},pow_{max},k]	40,4,20	2,2/1.69,1.69	20	13.3	27.5
	80,4,20	2,2/1.69,1.69	20	83.9	134.2
	40,8,20	2,2/1.91,1.91	20	49.1	84.6
	80,8,20	2,2/1.88,1.88	20	498.6	846.8

Table 8.6: Statistics for CSG two-player nonzero-sum verification instances.

Case study [parameters]	Param. values	Players	States	Transitions
Robot coordination $\langle\langle rbt_1{:}rbt_2\rangle\rangle_{\max=?}(\mathtt{P}[\neg \mathtt{c}\,\mathtt{U}^{\leqslant k}\,\mathtt{g}_1]+\mathtt{P}[\neg \mathtt{c}\,\mathtt{U}\,\mathtt{g}_2])$ $[l,k]$	4,8	2	1,923	56,385
	4,16	2	3,491	104,545
	8,8	2	36,773	1,860,691
	8,16	2	67,525	3,443,443
Medium access control $\langle\langle p_1{:}p_2{,}p_3\rangle\rangle_{\max=?}(\mathtt{P}[\mathtt{F}^{\leqslant k}\,\mathtt{m}_1]+\mathtt{P}[\mathtt{F}\,\mathtt{m}_{2,3}])$ $[e_{max}{,}s_{max}{,}k]$	4,4,4	3	89,405	718,119
	4,4,8	3	158,609	1,282,435
	6,4,6	3	944,727	9,071,885
	6,4,12	3	1,745,001	16,800,083
Power control $\langle\langle p_1{:}p_2\rangle\rangle_{\max=?}(\mathtt{R}[\mathtt{F}\,e_1{=}0]+\mathtt{R}[\mathtt{C}^{\leqslant k}])$ $[e_{max}{,}pow_{max}{,}k]$	40,4,20	2	182,772	1,040,940
	80,4,20	2	917,988	5,153,700
	40,8,20	2	524,473	4,179,617
	80,8,20	2	3,806,240	28,950,948

Table 8.7: Model statistics for CSGs built verifying mixed two-player nonzero-sum properties.

Case study & property [parameters]	Param. values	Players	CSG statistics			Constr. time(s)	Verif. time (s)
			States	Max. Act.	Trans.		
Secret Sharing $\mathtt{R}_{\max=?}[\mathtt{F}\ \mathtt{d}\vee r{=}r_{max}]$ *model*/$[\alpha{,}r_{max}{,}p_{fail}]$	*raa*/0.3,10,—	3	4,279	2,1,1	5,676	0.057	0.565
	rba/0.3,10,0.2	3	7,095	2,1,1	9,900	0.090	0.939
	rra/0.3,10,—	3	8,525	2,2,1	11,330	0.250	25.79
	rrr/0.3,10,—	3	17,017	2,2,2	22,638	0.250	96.07
Public Good $\mathtt{R}_{\max=?}[\mathtt{I}^{=k_{max}}]$ $[f, k_{max}]$	2.9,2	3	758	3,3,3	1,486	0.098	7.782
	2.9,3	3	16,337	3,3,3	36,019	0.799	110.1
	2.9,4	3	279,182	3,3,3	703,918	6.295	1,459
	2.9,1	4	83	3,3,3,3	163	0.046	0.370
	2.9,2	4	6,644	3,3,3,3	13,204	0.496	7.111
	2.9,3	4	399,980	3,3,3,3	931,420	11.66	99.86
	2.9,1	5	245	3,3,3,3,3	487	0.081	2.427
	2.9,2	5	59,294	3,3,3,3,3	118,342	2.572	2,291
Aloha (deadline) $\mathtt{P}_{\max=?}[\mathtt{F}\ \mathtt{s}_i \wedge t \leqslant D]$ $[b_{max}, D]$	1,8	3	3,519	2,2,2	5,839	0.168	11.23
	2,8	3	14,230	2,2,2	28,895	0.430	14.05
	3,8	3	72,566	2,2,2	181,438	1.466	18.41
	4,8	3	413,035	2,2,2	1,389,128	7.505	43.23
	1,8	4	23,251	2,2,2,2	42,931	0.708	75.59
	2,8	4	159,892	2,2,2,2	388,133	3.439	131.7
	3,8	4	1,472,612	2,2,2,2	4,777,924	28.69	819.2
	1,8	5	176,777	2,2,2,2,2	355,209	3.683	466.3
Aloha $\mathtt{R}_{\min=?}[\mathtt{F}\ \mathtt{s}_i]$ $[b_{max}]$	1	3	1,034	2,2,2	1,777	0.096	40.76
	2	3	5,111	2,2,2	10,100	0.210	29.36
	3	3	22,812	2,2,2	56,693	0.635	51.22
	4	3	107,799	2,2,2	355,734	2.197	150.1
Medium access $\mathtt{R}_{\max=?}[\mathtt{C}^{\leqslant k}]$ $[e_{max}, k]$	5,10	3	1,546	2,2,2	17,100	0.324	147.9
	10,10	3	10,591	2,2,2	135,915	1.688	682.7
	15,20	3	33,886	2,2,2	457,680	4.663	6,448
	5,5	4	15,936	2,2,2,2	333,314	4.932	3,581

Table 8.8: Statistics for CSG multi-player nonzero-sum verification instances.

CHAPTER 9

Conclusion

9.1 Summary

The main problem examined in this thesis is that of model checking and synthesis for systems that feature multiple players or components with distinct objectives making concurrent, rational decisions in an unpredictable environment. Game-theoretic models, particularly CSGs, are an ideal formalism for modelling these scenarios. In order to model uncertainty, we assume that it is a quantifiable measure that can be encoded according to a probability distribution or through nondeterministic choices. At every step, the evolution of the system is determined by which action each player takes, culminating in models that are more general than TSGs and MDPs. Whether agents in the model are expected to collaborate or work against each other is determined at the verification level by specifying sets of objectives and grouping or separating players into coalitions. We are then able to model a system as a concurrent stochastic game, analyse it through the specification of quantitative or qualitative properties in combination with model checking, and synthesise a strategy, which can also be used as evidence of the occurrence of a particular event or a controller to enforce a desirable behaviour.

It is not uncommon in model checking applications, particularly in non-stochastic scenarios, to focus on worst-case analysis and safety objectives, where the properties of interest tend to be of the type $\mathtt{G}\ \neg \mathit{failure}$, that is, making sure that something bad does not happen. It is equally common that systems are only verified against a specification, that is, as a model that translates nothing but its own implementation,

or against a nondeterministic but certain environment, especially in applications that focus on discrete control. As the focus shifts to the interaction among different systems, so does the need for modelling paradigms that allow researchers to have a clear framework for specifying how the interactions take place and reasoning about a more informative set of properties and measures. We have seen in our case studies why being able to model concurrency is important, as it makes possible to translate real-world scenarios more faithfully. Furthermore, we have illustrated the benefits of considering mutually beneficial strategies and an analysis that considers cooperation without disregarding self-interest and rationality. The most advantageous aspect of incorporating equilibria analysis is indeed being able to find a way for rational agents to interact and accomplish a set of individual goals with no reason to second-guess. The ability to apply different criteria when selecting equilibria, such as social welfare or social cost, adds to the stability factor of equilibria strategies by promoting cooperation among agents so as to maximise or minimise a collective measure. In order to achieve this, we introduce in Chapters 5 and 6 a property specification logic and algorithms to compute equilibria values for two and multi-player games, and in Chapter 7 a modelling language and a tool that implements these methods and can be used for automatic verification and synthesis.

Computing equilibria values and strategies of normal form games is a hard task. Across decades, it has been the subject of hundreds of papers from economics, mathematics and computer science. The issue tackled by this thesis is harder still as we set out to find optimal equilibria over extensive games, which required finding all equilibria values for millions of one-shot games in our largest case studies. Further difficulties were brought by considering non-discounted long-run objectives and properties using temporal operators, which considerably add to the complexity of our algorithms. Both the theoretical presentation and implementation could have been made simpler had we chosen to focus on discounted properties, which would also have made it possible to lift some of the restrictions we had to impose to guarantee convergence. Our feeling is that the range of applications reflected in our experimental results shows that those restrictions are not very limiting and both our methods and implementation could easily be adapted to consider a discount factor. The problem of strategy synthesis is no less challenging as it poses the added complication of keeping track of local strategies for all players, which may change after they have reached their goals. Both verification and synthesis additionally support a range of reward operators including expected reachability. For the two-player case, we also develop

and implement methods to enable a mixture of bounded and unbounded operators, for which we need to modify the original game accordingly.

Contributions. Our contributions can be grouped as follows:

- We have developed a framework for verification and strategy synthesis for concurrent stochastic games based on zero-sum and equilibria properties. Our model checking algorithms incorporate results from game theory, probabilistic verification, symbolic methods and branching-time temporal logics. While previous work had already tackled the issue of property specification logics for games and the development of model checking algorithms, we believe this thesis presents a more extensive and comprehensive framework, supporting multi-coalitional properties and a unified property specification logic for competitive and collaborating objectives based on probabilistic and rewards temporal formulae. Chapters 4-6 lay down the theoretical foundations and provide the reader with well defined algorithms that can be implemented and extended. Particularly when dealing with equilibria and concurrent stochastic games, most previous work had focused on complexity results or decision problems but lacked a proper implementation. The work developed in the aforementioned chapters present a non-trivial extension of pre-existing solutions for normal and extensive form games by considering temporal specifications, long-run objectives, incorporating SMT-based procedures and computing two variants of optimal equilibria.

- We showcase the usefulness and the efficiency of our approach by implementing our algorithms in the PRISM-games 3.0 tool, which is the subject of Chapter 7. The tool supports a modelling language for multi-player games, where the user can model multiple potentially synchronising components, describe how they interact and how the system evolves as the result of individual actions. The ability to export synthesised strategies in an intuitive way, plot graphs for experiments and use a simulator to follow an execution provide insight about the system to the designer. We demonstrate the wide applicability scope of both the methods and the tool, as well as the interest in being able to consider cooperation and multiple coalitions in Chapter 8.

9.2 Future Work

The work in this thesis could be the basis for a number of interesting extensions and future research directions. In this section, we highlight some we find particularly relevant and more closely linked.

Improvements to Zero-sum and Equilibria Computation. Scalability is a major concern when developing and implementing model checking algorithms if they are to be capable of handling real-world applications. With respect to concurrent stochastic games, how we compute local (state) solutions can have a major impact on the overall performance. The general approach for zero-sum games is to solve a linear program at each state and each iteration. However, some games can be trivial as they can be solved by simpler methods such as searching for *saddle points*, that is, an entry in the matrix that is the minimal entry in its row and the maximal of its column. Furthermore, there are implementations of polynomial algorithms for solving LPs, which would perform better than that of the solver we used and decrease the model checking times we reported. Similarly, local computation of equilibria values is of crucial importance, particularly when we are dealing with finding an optimal equilibrium. In the current implementation we already search for and filter out dominated strategies in both two and multi-player cases. It would be beneficial to also have a precomputation step to remove *conditionally dominated* strategies [174], particularly when performing support enumeration. For the two-player case particularly, an interesting direction to follow is applying *maximum model counting* [89] when computing optimal equilibria as the SMT encoding can be equally translated into SAT and, given it is a linear problem, one would not need to encode multiplication between variables. It could also be helpful when computing solutions for the local games to use values of the past iteration as the method for the two-player case relies on search and that for the multi-player on nonlinear programming and gradient descent. In both cases, for many states the value computed in the next iteration is likely to be in the vicinity of the last computed value, and starting from that value could speed up computation. More related to calculating the value for the entire game, interesting directions are to explore symmetries in the graph as well as skipping computation for states from which only target states can be reached, as their values are not going to change after the first iteration. At the moment, each iteration does an undiscriminated sweep and repeats computation that is unnecessary. Additionally, solution methods for both zero-sum and equilibria problems are sensitive to the limitations and issues of floating-point

arithmetic, particularly equilibria computation, and might benefit from arbitrary precision representations. Lastly, we mention that recent research has pointed out the shortcomings of only using a lower bound approximation as a stopping criterion for value iteration, as it can lead to inaccuracies [14, 101, 127]. The impact of similar issues on model checking for games is still to be studied.

Partial Observability. The algorithms and modelling language presented in this thesis assume that we have full knowledge of every aspect of the system beforehand. In some cases, however, it is not possible to be sure in which state a system currently is, as some of its parameters might not be observable. In such cases, it is common to say that one has a *belief* of where the system is and verification has then to account for that. Solutions to handle this type of problem include partial observable Markov decision process [123, 1] and partially observable games [46, 45]. Partial observability adds to the complexity of model checking and can also make problems undecidable. The problem of partial observability for concurrent stochastic games and zero-sum properties has been recently explored in [116]. Bayesian games [108] would be a starting point to consider new algorithms that combine partial observability and equilibria.

Correlated Equilibria. Correlated equilibria were introduced by Aumann in [8]. It is a concept that generalises NE in the sense that is possible to construct a correlated equilibrium for every NE. The opposite, however, is not true and thus correlated equilibria are a strictly weaker notion than NE. The idea behind correlated equilibria is that players condition their strategies based on a probability distribution over a set of outcomes, e.g., cooperate or defect in a prisoner's dilemma based on the outcome of a coin flip. The outcome is revealed to all players in the game and they can condition their decisions accordingly. Correlated equilibria may lead to payoffs which are fairer and achieve higher expected utilities. Moreover, it has been shown in [95] that finding a correlated equilibrium with maximal payoff for games in normal form can be done in polynomial time, whereas finding a maximum NE is NP-hard (NP-complete for two players). As they may return fairer equilibria and are computationally less costly, it is a interesting direction to follow.

Mechanism Design and Coalition Building. In a way, mechanism design [165, 163] goes in the opposite direction of verification. Here we start with a full model, which may or may not have reward structures, and a property where players are separated into coalitions. We then try to compute the value associated to that property and either output the result and/or compare it with a given bound. Mechanism

design, on the other hand, asks how can we set incentives, which could be seen as reward structures or probabilities, in such way that we have an equilibrium among the participants. We have seen the application of that in some of our examples, particularly the secret sharing case study, and specifically how having the right incentives (or punishments) are crucial to make sure that rational agents act according to a specification that guarantees the good functioning of a protocol and are in accordance with other principles such as fairness and social welfare. In our example though, we have a varying parameter and instantiate different models to see how the behaviour of the agents change. Naturally, if we consider non-discrete increments or a parameter that can take a very large range of values, that approach becomes impractical. Mechanism design tackles this issue by presenting a more structured way of finding these values. Similarly, whenever we verify a property or synthesise a strategy, we do so by grouping the players into coalitions. One could raise the question of whether those coalitions are indeed an optimal split when considering a given criterion, as for instance, social welfare. It is possible that it would be collectively more beneficial for a player to cooperate with another player than the one he was grouped with. Research on how to group agents to optimise cooperation, and whether certain coalitions are even feasible, given that a player may get the same utility by playing by itself (as seen in the public good game case study) is also a thriving topic within game theory.

9.3 Final Remarks

With the ever growing complexity of autonomous systems and their interactions, we believe that formal modelling and synthesis methods will become more and more relevant to ensure they behave safely. Automatic controller synthesis for coordination among different agents is still in its inception, and the usefulness of game-theoretic concepts such as equilibria remains largely obscure to computer scientists and engineers alike. We think that the work reported in this thesis contributes to addressing some gaps and presents a comprehensive account of the challenges and the importance of bridging verification and game theory. It is our opinion that developments in this area have the potential to be pivotal in transforming the design and certification process of autonomous systems as well as having an impact in several domains within and beyond computer science.

APPENDIX A

Correctness

A.1 Correctness of the Model Checking Algorithms

The overall (recursive) approach and the reduction to solution of a two-player game is essentially the same as for TSGs [53], and therefore the same correctness arguments apply. In the case of *zero-sum* formulae, the correctness of value iteration for infinite-horizon properties follows from [177] and for finite-horizon properties from Definition 4.4.3 and the solution of matrix games (see Section 4.2.1). Below, we show the correctness of the model checking algorithms for *nonzero-sum* formulae.

A.1.1 Nonzero-sum Two-player Formulae

We fix a game G and a nonzero-sum formula $\langle\langle C{:}C' \rangle\rangle_{\mathrm{opt}\sim x}(\theta)$. For the case of *finite-horizon* nonzero-sum formulae, the correctness of the model checking algorithms follows from the fact that we use backward induction [181, 205, 188]. For *infinite-horizon* nonzero-sum formulae, the proof is based on showing that the values computed during value iteration correspond to subgame-perfect SWNE values of finite game trees, and the values of these game trees converge uniformly and are bounded from above by the actual values of $\mathsf{G}^{\mathcal{C}}$.

The fact that we use MDP model checking when the goal of one of the players is reached means that the values computed during value iteration are not finite approximations for the values of $\mathsf{G}^{\mathcal{C}}$. Therefore we must also show that the values computed during value iteration are bounded from below by finite approximations for the values

of $\mathsf{G}^{\mathcal{C}}$. We first consider the case when both the objectives in the sum θ are infinite-horizon objectives. Below we assume opt = max and the case when opt = min follow similarly. For any $(v_1, v_2), (v_1', v_2') \in \mathbb{Q}^2$, let $(v_1, v_2) \leqslant (v_1', v_2')$ if and only if $v_1 \leqslant v_1'$ and $v_2 \leqslant v_2'$. The following lemma follows by definition of subgame-perfect SWNE values.

Lemma 4. Consider any strategy profile σ and state s of $\mathsf{G}^{\mathcal{C}}$ and let $(v_1^{\sigma,s}, v_2^{\sigma,s})$ be the corresponding values of the players in s for the objectives (X_1^θ, X_2^θ). Considering subgame-perfect SWNE values of the objectives (X_1^θ, X_2^θ) in state s, in the case that θ is of the form $\mathtt{P}[\,\phi_1^1\ \mathtt{U}\ \phi_2^1\,]{+}\mathtt{P}[\,\phi_1^2\ \mathtt{U}\ \phi_2^2\,]$:

- if $s \models \phi_2^1 \wedge \phi_2^2$, then $(1,1)$ are the unique subgame-perfect SWNE values for state s and $(v_1^{\sigma,s}, v_2^{\sigma,s}) \leqslant (1,1)$;
- if $s \models \phi_2^1 \wedge \phi_1^2 \wedge \neg\phi_2^2$, then $(1, \mathtt{P}^{\max}_{\mathsf{G},s}(\phi_1^2\ \mathtt{U}\ \phi_2^2))$ are the unique subgame-perfect SWNE values for state s and $(v_1^{\sigma,s}, v_2^{\sigma,s}) \leqslant (1, \mathtt{P}^{\max}_{\mathsf{G},s}(\phi_1^2\ \mathtt{U}\ \phi_2^2))$;
- if $s \models \phi_1^1 \wedge \neg\phi_2^1 \wedge \phi_2^2$, then $(\mathtt{P}^{\max}_{\mathsf{G},s}(\phi_1^1\ \mathtt{U}\ \phi_2^1), 1)$ are the unique subgame-perfect SWNE values for state s and $(v_1^{\sigma,s}, v_2^{\sigma,s}) \leqslant (\mathtt{P}^{\max}_{\mathsf{G},s}(\phi_1^1\ \mathtt{U}\ \phi_2^1), 1)$;
- if $s \models \phi_2^1 \wedge \neg\phi_1^1 \wedge \neg\phi_2^2$, then $(1,0)$ are the unique subgame-perfect SWNE values for state s and $(v_1^{\sigma,s}, v_2^{\sigma,s}) \leqslant (1,0)$;
- if $s \models \neg\phi_1^1 \wedge \neg\phi_2^1 \wedge \phi_2^2$, then $(0,1)$ are the unique subgame-perfect SWNE values for state s and $(v_1^{\sigma,s}, v_2^{\sigma,s}) \leqslant (0,1)$;
- if $s \models \neg\phi_1^1 \wedge \neg\phi_2^1 \wedge \phi_1^2 \wedge \neg\phi_2^2$, then $(0, \mathtt{P}^{\max}_{\mathsf{G},s}(\phi_1^2\ \mathtt{U}\ \phi_2^2))$ are the unique subgame-perfect SWNE values for state s and $(v_1^{\sigma,s}, v_2^{\sigma,s}) \leqslant (0, \mathtt{P}^{\max}_{\mathsf{G},s}(\phi_1^2\ \mathtt{U}\ \phi_2^2))$;
- if $s \models \phi_1^1 \wedge \neg\phi_2^1 \wedge \neg\phi_1^2 \wedge \neg\phi_2^2$, then $(\mathtt{P}^{\max}_{\mathsf{G},s}(\phi_1^1\ \mathtt{U}\ \phi_2^1), 0)$ are the unique subgame-perfect SWNE values for state s and $(v_1^{\sigma,s}, v_2^{\sigma,s}) \leqslant (\mathtt{P}^{\max}_{\mathsf{G},s}(\phi_1^1\ \mathtt{U}\ \phi_2^1), 0)$;
- if $s \models \neg\phi_1^1 \wedge \neg\phi_2^1 \wedge \neg\phi_1^2 \wedge \neg\phi_2^2$, then $(0,0)$ are the unique subgame-perfect SWNE values for state s and $(v_1^{\sigma,s}, v_2^{\sigma,s}) \leqslant (0,0)$.

On the other hand, in the case that θ is of the form $\mathtt{R}^{r_1}[\,\mathtt{F}\ \phi^1\,]{+}\mathtt{R}^{r_2}[\,\mathtt{F}\ \phi^2\,]$:

- if $s \models \phi^1 \wedge \phi^2$, then $(0,0)$ are the unique subgame-perfect SWNE values for state s and $(v_1^{\sigma,s}, v_2^{\sigma,s}) \leqslant (0,0)$;
- if $s \models \phi^1 \wedge \neg\phi^2$, then $(0, \mathtt{R}^{\max}_{\mathsf{G},s}(r_2, \mathtt{F}\ \phi^2))$ are the unique subgame-perfect SWNE values for state s and $(v_1^{\sigma,s}, v_2^{\sigma,s}) \leqslant (0, \mathtt{R}^{\max}_{\mathsf{G},s}(r_2, \mathtt{F}\ \phi^2))$;

- if $s \models \neg\phi^1 \wedge \phi^2$, then $(\mathtt{R}^{\max}_{\mathsf{G},s}(r_1, \mathtt{F}\ \phi^1), 0)$ are the unique subgame-perfect SWNE values for state s and $(v_1^{\sigma,s}, v_2^{\sigma,s}) \leqslant (\mathtt{R}^{\max}_{\mathsf{G},s}(r_1, \mathtt{F}\ \phi^1), 0)$.

Next we require the following objectives of $\mathsf{G}^{\mathcal{C}}$.

Definition A.1.1. For any sum of two probabilistic or reward objectives θ, $1 \leqslant i \leqslant 2$ and $n \in \mathbb{N}$, let $X^{\theta}_{i,n}$ be the objective where for any path π of $\mathsf{G}^{\mathcal{C}}$:

$$X^{\mathtt{P}[\,\phi_1^1\ \mathtt{U}\ \phi_2^1\,]+\mathtt{P}[\,\phi_1^2\ \mathtt{U}\ \phi_2^2\,]}_{i,n}(\pi) \;=\; \begin{cases} 1 & \text{if } \exists k \leqslant n.\, (\pi(k) \models \phi_2^i \wedge \forall j < k.\, \pi(j) \models \phi_1^i) \\ 0 & \text{otherwise} \end{cases}$$

$$X^{\mathtt{R}^{r_1}[\,\mathtt{F}\ \phi^1\,]+\mathtt{R}^{r_2}[\,\mathtt{F}\ \phi^2\,]}_{i,n}(\pi) \;=\; \begin{cases} \infty & \text{if } \forall k \in \mathbb{N}.\, \pi(k) \not\models \phi^i \\ \sum_{k=0}^{k_{\phi_i}-1} \big(r_A(\pi(k), \pi[k]) + r_S(\pi(k))\big) & \text{if } k_{\phi^i} \leqslant n-1 \\ 0 & \text{otherwise} \end{cases}$$

and $k_{\phi_i} = \min\{k \mid k \in \mathbb{N} \wedge \pi(k) \models \phi^i\}$.

The following lemma demonstrates that, for a fixed strategy profile and state, the values of these objectives are non-decreasing and converge uniformly to the values of θ.

Lemma 5. For any sum of two probabilistic or reward objectives θ and $\varepsilon > 0$, there exists $N \in \mathbb{N}$ such that, for any $n \geqslant N$, $s \in S$, $\sigma \in \Sigma^1_{\mathsf{G}^{\mathcal{C}}} \times \Sigma^2_{\mathsf{G}^{\mathcal{C}}}$ and $1 \leqslant i \leqslant 2$:

$$0 \;\leqslant\; \mathbb{E}^{\sigma}_{\mathsf{G}^{\mathcal{C}},s}(X^{\theta}_i) - \mathbb{E}^{\sigma}_{\mathsf{G}^{\mathcal{C}},s}(X^{\theta}_{i,n}) \;\leqslant\; \varepsilon\,.$$

Proof. Consider any sum of two probabilistic or reward objectives θ, state s and $1 \leqslant i \leqslant 2$. Using Assumption 4 we have that, for subformulae $\mathtt{R}^r[\,\mathtt{F}\ \phi^i\,]$, the set $Sat(\phi^i)$ is reached with probability 1 from all states of G under all profiles, and therefore $\mathbb{E}^{\sigma}_{\mathsf{G}^{\mathcal{C}},s}(X^{\theta}_i)$ is finite. Furthermore, for any $n \geqslant N$, by Definitions 5.2.2 and A.1.1 we have that $\mathbb{E}^{\sigma}_{\mathsf{G}^{\mathcal{C}},s}(X^{\theta}_{i,n})$ is the value of state s for the nth iteration of value iteration [47] when computing $\mathbb{E}^{\sigma}_{\mathsf{G}^{\mathcal{C}},s}(X^{\theta}_i)$ in the DTMC obtained from $\mathsf{G}^{\mathcal{C}}$ by following the strategy σ, and the sequence is both non-decreasing and converges. The fact that we can choose an N independent of the strategy profile for uniform convergence follows from Assumptions 3 and 4. □

In the proof of correctness, we will use the fact that n iterations of value iteration is equivalent to performing backward induction on the following game trees.

Definition A.1.2. For any state s and $n \in \mathbb{N}$, let $\mathsf{G}^{\mathcal{C}}_{n,s}$ be the game tree corresponding to playing $\mathsf{G}^{\mathcal{C}}$ for n steps when starting from state s and then terminating.

We can map any strategy profile σ of $\mathsf{G}^{\mathcal{C}}$ to a strategy profile of $\mathsf{G}^{\mathcal{C}}_{n,s}$ by only considering the choices of the profile over the first n steps when starting from state s. This mapping is clearly surjective, i.e., we can generate all profiles of $\mathsf{G}^{\mathcal{C}}_{n,s}$, but is not injective. We also need the following objectives corresponding to the values computed during value iteration for the game trees of Definition A.1.2.

Definition A.1.3. For any sum of two probabilistic or reward objectives θ, $s \in S$, $n \in \mathbb{N}$, $1 \leqslant i \leqslant 2$ and $j = i{+}1 \bmod 2$, let Y_i^θ be the objective where, for any path π of $\mathsf{G}^{\mathcal{C}}_{n,s}$:

$$Y_i^{\mathtt{P}[\,\phi_1^1\ \mathtt{U}\ \phi_2^1\,]+\mathtt{P}[\,\phi_1^2\ \mathtt{U}\ \phi_2^2\,]}(\pi) =$$

$$\begin{cases} 1 & \text{if } \exists m \leqslant n.\, (\pi(m) \models \phi_2^i \wedge \forall k < m.\, \pi(k) \models \phi_1^1 \wedge \neg\phi_2^1 \wedge \phi_1^2 \wedge \neg\phi_2^2) \\ \mathsf{P}^{\max}_{\mathsf{G},\pi(m)}(\phi_1^i\ \mathtt{U}\ \phi_2^i) & \text{else if } \exists m \leqslant n.\, (\pi(m) \models \phi_2^j \wedge \forall k < m.\, \pi(k) \models \phi_1^1 \wedge \neg\phi_2^1 \wedge \phi_1^2 \wedge \neg\phi_2^2) \\ 0 & \text{otherwise} \end{cases}$$

$$Y_i^{\mathtt{R}^{r_1}[\,\mathtt{F}\ \phi^1\,]+\mathtt{R}^{r_2}[\,\mathtt{F}\ \phi^2\,]}(\pi) =$$

$$\begin{cases} \infty & \text{if } \forall k \leqslant n.\, \pi(k) \not\models \phi^i \\ \sum_{k=0}^{k_{\phi^1\vee\phi^2}-1} \big(r_A(\pi(k), \pi[k]) + r_S(\pi(k))\big) + r_S^i(\pi(k)) & \text{otherwise} \end{cases}$$

where

$$r_S^i(s') \;=\; \begin{cases} \mathtt{R}^{\max}_{\mathsf{G},s'}(r_i, \mathtt{F}\ \phi^i) & \text{if } s' \models \neg\phi^i \wedge \phi^j \\ 0 & \text{otherwise} \end{cases}$$

for $s' \in S$ and $k_{\phi^1\vee\phi^2} = \min\{k \mid k \leqslant n \wedge \pi(k) \models \phi^1 \vee \phi^2\}$.

Similarly to Lemma 5, the lemma below demonstrates, for a fixed strategy profile and state s of $\mathsf{G}^{\mathcal{C}}$, that the values for the objectives given in Definition A.1.3 when played on the game trees $\mathsf{G}^{\mathcal{C}}_{n,s}$ are non-decreasing and converge uniformly. As with Lemma 5 the result follows from Assumptions 3 and 4.

Lemma 6. For any sum of two probabilistic or reward objectives θ and $\varepsilon > 0$, there exists $N \in \mathbb{N}$ such that for any $m \geqslant n \geqslant N$, $\sigma \in \Sigma^1_{\mathsf{G}^{\mathcal{C}}} \times \Sigma^2_{\mathsf{G}^{\mathcal{C}}}$, $s \in S$ and $1 \leqslant i \leqslant 2$:

$$0 \;\leqslant\; \mathbb{E}^{\sigma}_{\mathsf{G}^{\mathcal{C}}_{m,s}}(Y_i^\theta) - \mathbb{E}^{\sigma}_{\mathsf{G}^{\mathcal{C}}_{n,s}}(Y_i^\theta) \;\leqslant\; \varepsilon\,.$$

We require the following lemma relating the values of the objectives $X^\theta_{i,n}$, Y_i^θ and X_i^θ for $1 \leqslant i \leqslant 2$.

Lemma 7. For any sum of two probabilistic or reward objectives θ, state s of $\mathsf{G}^{\mathcal{C}}$, strategy profile σ such that when one of the targets of the objectives of θ is reached, the profile then collaborates to maximise the value of the other objective, $n \in \mathbb{N}$ and $1 \leqslant i \leqslant 2$:

$$\sup_{\sigma_i \in \Sigma_i^{\mathsf{G}^{\mathcal{C}}_{n,s}}} \mathbb{E}^{\sigma_{-i}[\sigma_i]}_{\mathsf{G}^{\mathcal{C}},s}(X^\theta_{i,n}) \;\leqslant\; \sup_{\sigma_i \in \Sigma_i^{\mathsf{G}^{\mathcal{C}}_{n,s}}} \mathbb{E}^{\sigma_{-i}[\sigma_i]}_{\mathsf{G}^{\mathcal{C}}_{n,s}}(Y_i^\theta) \;\leqslant\; \sup_{\sigma_i \in \Sigma_i^{\mathsf{G}^{\mathcal{C}}}} \mathbb{E}^{\sigma_{-i}[\sigma_i]}_{\mathsf{G}^{\mathcal{C}},s}(X_i^\theta)\,.$$

Proof. Consider any strategy profile σ, $n \in \mathbb{N}$ and $1 \leqslant i \leqslant 2$. By Definitions A.1.1 and A.1.3 it follows that:

$$\mathbb{E}^{\sigma}_{\mathsf{G}^{\mathcal{C}},s}(X^{\theta}_{i,n}) \;\leqslant\; \mathbb{E}^{\sigma}_{\mathsf{G}^{\mathcal{C}}_{n,s}}(Y^{\theta}_{i}).$$

Furthermore, if we restrict the profile σ such that, when one of the targets of the objectives of θ is reached, the profile then collaborates to maximise the value of the other objective, then by Definitions A.1.3 and 5.2.2:

$$\mathbb{E}^{\sigma}_{\mathsf{G}^{\mathcal{C}}_{n,s}}(Y^{\theta}_{i}) \;\leqslant\; \mathbb{E}^{\sigma}_{\mathsf{G}^{\mathcal{C}},s}(X^{\theta}_{i}).$$

Combining these results with Lemma 4, we have:

$$\sup\nolimits_{\sigma_i \in \Sigma_i^{\mathsf{G}^{\mathcal{C}}_{n,s}}} \mathbb{E}^{\sigma_{-i}[\sigma_i]}_{\mathsf{G}^{\mathcal{C}},s}(X^{\theta}_{i,n}) \;\leqslant\; \sup\nolimits_{\sigma_i \in \Sigma_i^{\mathsf{G}^{\mathcal{C}}_{n,s}}} \mathbb{E}^{\sigma_{-i}[\sigma_i]}_{\mathsf{G}^{\mathcal{C}}_{n,s}}(Y^{\theta}_{i}) \;\leqslant\; \sup\nolimits_{\sigma_i \in \Sigma_i^{\mathsf{G}^{\mathcal{C}}}} \mathbb{E}^{\sigma_{-i}[\sigma_i]}_{\mathsf{G}^{\mathcal{C}},s}(X^{\theta}_{i})$$

as required. □

We now define the strategy profiles synthesised during value iteration.

Definition A.1.4. For any $n \in \mathbb{N}$ and $s \in S$, let $\sigma^{n,s}$ be the strategy profile generated for the game tree $\mathsf{G}^{\mathcal{C}}_{n,s}$ (when considering value iteration as backward induction) and $\sigma^{n,\star}$ be the synthesised strategy profile for $\mathsf{G}^{\mathcal{C}}$ after n iterations.

Before giving the proof of correctness we require the following results.

Lemma 8. For any state s of $\mathsf{G}^{\mathcal{C}}$, sum of two probabilistic or reward objectives θ and $n \in \mathbb{N}$ we have that $\sigma^{n,s}$ is a subgame-perfect SWNE profile of the CSG $\mathsf{G}^{\mathcal{C}}_{n,s}$ for the objectives $(Y^{\theta}_1, Y^{\theta}_2)$.

Proof. The result follows from the fact that value iteration selects SWNE profiles, value iteration corresponds to performing backward induction for objectives $(Y^{\theta}_1, Y^{\theta}_2)$ and backward induction returns a subgame-perfect NE [205, 188]. □

The following proposition demonstrates that value iteration converges and depends on Assumptions 3 and 4. Without this assumption, convergence cannot be guaranteed as demonstrated by the counterexamples in Appendices C.3 and C.4. Although value iteration converges, unlike value iteration for MDPs or zero-sum games, the generated sequence of values is not necessarily non-decreasing.

Proposition 1. For any sum of two probabilistic or reward objectives θ and state s, the sequence $\langle \mathsf{V}_{\mathsf{G}^{\mathcal{C}}}(s, \theta, n) \rangle_{n \in \mathbb{N}}$ converges.

Proof. For any state s and $n \in \mathbb{N}$ we can consider $\mathsf{G}^{\mathcal{C}}_{n,s}$ as two-player infinite-action NFGs $\mathsf{N}_{n,s}$ where for $1 \leqslant i \leqslant 2$:

- the set of actions of player i equals the set of strategies of player i in $\mathsf{G}^{\mathcal{C}}$;
- for the strategy profile (σ_1, σ_2), the utility function for player i returns $\mathbb{E}^{\sigma}_{\mathsf{G}^{\mathcal{C}}_{n,s}}(Y^{\theta}_i)$.

The correctness of this construction relies on the mapping of strategy profiles from the game $\mathsf{G}^{\mathcal{C}}$ to $\mathsf{G}^{\mathcal{C}}_{n,s}$ being surjective. Using Lemma 6, we have that the sequence $\langle \mathsf{N}_{n,s} \rangle_{n \in N}$ of NFGs converges uniformly, and therefore, since $\mathtt{V}_{\mathsf{G}^{\mathcal{C}}}(s, \theta, n)$ are subgame-perfect SWNE values of $\mathsf{G}^{\mathcal{C}}_{n,s}$ (see Lemma 8), the sequence $\langle \mathtt{V}_{\mathsf{G}^{\mathcal{C}}}(s, \theta, n) \rangle_{n \in \mathbb{N}}$ also converges. □

A similar convergence result to Proposition 1 has been shown for the simpler case of discounted properties in [90].

Lemma 9. For any $\varepsilon > 0$, there exists $N \in \mathbb{N}$ such that for any $s \in S$ and $1 \leqslant i \leqslant 2$:

$$\left| \mathbb{E}^{\sigma^{n,\star}}_{\mathsf{G}^{\mathcal{C}},s}(X^{\theta}_i) - \mathbb{E}^{\sigma^{n,s}}_{\mathsf{G}^{\mathcal{C}}_{n,s}}(Y^{\theta}_i) \right| \leqslant \varepsilon \,.$$

Proof. Using Lemma 6 and Proposition 1, we can choose N such that the choices of the profile $\sigma^{n,s}$ agree with those of $\sigma^{n,\star}$ for a sufficient number of steps such that the inequality holds. □

Theorem 2. For a given sum of two probabilistic or reward objectives θ and $\varepsilon > 0$, there exists $N \in \mathbb{N}$ such that for any $n \geqslant N$ the strategy profile $\sigma^{n,\star}$ is a subgame-perfect ε-SWNE profile of $\mathsf{G}^{\mathcal{C}}$ and the objectives $(X^{\theta_1}, X^{\theta_2})$.

Proof. Consider any $\varepsilon > 0$. From Lemma 9 there exists $N_1 \in \mathbb{N}$ such that for any $s \in S$ and $n \geqslant N_1$:

$$\left| \mathbb{E}^{\sigma^{n,\star}}_{\mathsf{G}^{\mathcal{C}},s}(X^{\theta}_i) - \mathbb{E}^{\sigma^{n,s}}_{\mathsf{G}^{\mathcal{C}}_{n,s}}(Y^{\theta}_i) \right| \leqslant \tfrac{\varepsilon}{2} \,. \tag{A.1}$$

For any $m \in \mathbb{N}$ and $s \in S$, using Lemma 8 we have that $\sigma^{m,s}$ is a NE of $\mathsf{G}^{\mathcal{C}}_{m,s}$, and therefore for any $m \in \mathbb{N}$, $s \in S$ and $1 \leqslant i \leqslant 2$:

$$\mathbb{E}^{\sigma^{m,s}}_{\mathsf{G}^{\mathcal{C}}_{m,s}}(Y^{\theta}_i) \geqslant \sup_{\sigma_i \in \Sigma_i^{\mathsf{G}^{\mathcal{C}}_{m,s}}} \mathbb{E}^{\sigma^{m,s}_{-i}[\sigma_i]}_{\mathsf{G}^{\mathcal{C}}_{m,s}}(Y^{\theta}_i) \,. \tag{A.2}$$

From Lemma 5 there exists $N_2 \in \mathbb{N}$ such that for any $n \geqslant N_2$, $s \in S$ and $1 \leqslant i \leqslant 2$:

$$\sup_{\sigma_i \in \Sigma_i^{\mathsf{G}^{\mathcal{C}}}} \mathbb{E}^{\sigma^{n,\star}_{-i}[\sigma_i]}_{\mathsf{G}^{\mathcal{C}},s}(X^{\theta}_i) - \sup_{\sigma_i \in \Sigma_i^{\mathsf{G}^{\mathcal{C}}}} \mathbb{E}^{\sigma^{n,\star}_{-i}[\sigma_i]}_{\mathsf{G}^{\mathcal{C}},s}(X^{\theta}_{i,n}) \leqslant \tfrac{\varepsilon}{2} \,. \tag{A.3}$$

By construction, $\sigma^{n,\star}$ is a profile for which, if one of the targets of the objectives of θ is reached, the profile maximises the value of the objective. We can thus rearrange (A.3) and apply Lemma 7 to yield for any $n \geqslant N_2$, $s \in S$ and $1 \leqslant i \leqslant 2$:

$$\sup_{\sigma_i \in \Sigma_i^{\mathsf{G}^{\mathcal{C}}_{n,s}}} \mathbb{E}^{\sigma_{-i}^{n,s}[\sigma_i]}_{\mathsf{G}^{\mathcal{C}}_{n,s}}(Y_i^\theta) \;\geqslant\; \sup_{\sigma_i \in \Sigma_i^{\mathsf{G}^{\mathcal{C}}}} \mathbb{E}^{\sigma_{-i}^{n,\star}[\sigma_i]}_{\mathsf{G}^{\mathcal{C}},s}(X_i^\theta) - \tfrac{\varepsilon}{2}\,. \tag{A.4}$$

Letting $N = \max\{N_1, N_2\}$, for any $n \geqslant N$, $s \in S$ and $1 \leqslant i \leqslant 2$:

$$\begin{aligned}
\mathbb{E}^{\sigma^{n,\star}}_{\mathsf{G}^{\mathcal{C}},s}(X_i^\theta) &\geqslant \mathbb{E}^{\sigma^{n,s}}_{\mathsf{G}^{\mathcal{C}}_{n,s}}(Y_i^\theta) - \tfrac{\varepsilon}{2} && \text{by (A.1) since } n \geqslant N_1 \\
&\geqslant \sup_{\sigma_i \in \Sigma_i^{\mathsf{G}^{\mathcal{C}}_{n,s}}} \mathbb{E}^{\sigma_{-i}^{n,s}[\sigma_i]}_{\mathsf{G}^{\mathcal{C}}_{n,s}}(Y_i^\theta) - \tfrac{\varepsilon}{2} && \text{by (A.2)} \\
&\geqslant \left(\sup_{\sigma_i \in \Sigma_i^{\mathsf{G}^{\mathcal{C}}}} \mathbb{E}^{\sigma_{-i}^{n,\star}[\sigma_i]}_{\mathsf{G}^{\mathcal{C}},s}(X_i^\theta) - \tfrac{\varepsilon}{2}\right) - \tfrac{\varepsilon}{2} && \text{by (A.4) since } n \geqslant N_2 \\
&= \sup_{\sigma_i \in \Sigma_i^{\mathsf{G}^{\mathcal{C}}}} \mathbb{E}^{\sigma_{-i}^{n,\star}[\sigma_i]}_{\mathsf{G}^{\mathcal{C}},s}(X_i^\theta) - \varepsilon
\end{aligned}$$

and hence, since $\varepsilon > 0$, $s \in S$ and $1 \leqslant i \leqslant 2$ were arbitrary, $\sigma^{n,\star}$ is a subgame-perfect ε-NE. It remains to show that the strategy profile is a subgame-perfect social welfare optimal ε-NE, which follows from the fact that when solving the bimatrix games during value iteration social welfare optimal NE are returned. □

It remains to consider the model checking algorithms for nonzero-sum properties for which the sum of objectives contains both a finite-horizon and an infinite-horizon objective. In this case (see Section 5.3.5), for a given game $\mathsf{G}^{\mathcal{C}}$ and sum of objectives θ, the algorithms first build a modified game G' with states $S' \subseteq S \times \mathbb{N}$ and sum of infinite-horizon objectives θ' and then computes SWNE/SCNE values of θ' in G'. The correctness of these algorithms follows by first showing there exists a bijection between the profiles of $\mathsf{G}^{\mathcal{C}}$ and G' and then that, for any profile σ of $\mathsf{G}^{\mathcal{C}}$ and σ', the corresponding profile of G' under this bijection, we have:

$$\mathbb{E}^{\sigma}_{\mathsf{G}^{\mathcal{C}},s}(X_i^\theta) = \mathbb{E}^{\sigma'}_{\mathsf{G}',(s,0)}(X_i^{\theta'})$$

for all states s of $\mathsf{G}^{\mathcal{C}}$ and $1 \leqslant i \leqslant 2$. This result follows from the fact that in Section 5.3.5 we used a standard construction for converting the verification of finite-horizon properties to infinite-horizon properties.

A.1.2 Nonzero-sum Multi-player Formulae

We fix a game G and nonzero-sum state formula $\langle\!\langle C_1 : \cdots : C_m \rangle\!\rangle_{\mathrm{opt} \sim x}(\theta)$ and let $\mathcal{C} = \{C_1, \ldots, C_m\}$. For the case of finite-horizon nonzero-sum formulae the correctness of the model checking algorithm follows from the fact that we use backward induction [181, 205]. Below we consider probabilistic and expected reachability objectives

in the case that opt = max. The remaining cases for infinite-horizon nonzero-sum formulae follow similarly. For a nonzero-sum formula θ, we denote by θ_i the ith term in θ. We first introduce the following objectives for the coalition game $\mathsf{G}^{\mathcal{C}}$ which are n-step approximations of X_i^θ.

Definition A.1.5. For any probabilistic or expected reachability nonzero-sum formula θ, $1 \leqslant i \leqslant m$ and $n \in \mathbb{N}$, let $X_{i,n}^\theta$ be the objective where for any path π of $\mathsf{G}^{\mathcal{C}}$:

$$X_{i,n}^{\mathtt{P}[\mathtt{F}\ \phi^1]+\cdots+\mathtt{P}[\mathtt{F}\ \phi^m]}(\pi) \quad = \quad \begin{cases} 1 & \text{if } \exists k \leqslant n.\, \pi(k) \models \phi^i \\ 0 & \text{otherwise} \end{cases}$$

$$X_{i,n}^{\mathtt{R}^{r_1}[\mathtt{F}\ \phi^1]+\cdots+\mathtt{R}^{r_m}[\mathtt{F}\ \phi^m]}(\pi) \quad = \quad \begin{cases} \infty & \text{if } \forall k \in \mathbb{N}.\, \pi(k) \not\models \phi^i \\ \sum_{k=0}^{k^i} r(\pi,k) & \text{if } k^i \leqslant n-1 \\ 0 & \text{otherwise} \end{cases}$$

$r(\pi,k) = r_A(\pi(k), \pi[k]) + r_S(\pi(k))$ and $k^i = \min\{k-1 \mid k \in \mathbb{N} \wedge \pi(k) \models \phi^i\}$.

The following lemma demonstrates that, for a fixed strategy profile and state, the values of these objectives are non-decreasing and converge uniformly to the values of θ.

Lemma 10. For any probabilistic or expected reachability nonzero-sum formula θ we have that the sequence $\langle \mathbb{E}^\sigma_{\mathsf{G}^{\mathcal{C}},s}(X_i^\theta)\rangle_{n\in\mathbb{N}}$ is non-decreasing and, for any $\varepsilon > 0$, there exists $N \in \mathbb{N}$ such that for any $n \geqslant N$, $s \in S$, $\sigma \in \Sigma_{\mathsf{G}^{\mathcal{C}}}$ and $1 \leqslant i \leqslant m$:

$$0 \ \leqslant \ \mathbb{E}^\sigma_{\mathsf{G}^{\mathcal{C}},s}(X_i^\theta) - \mathbb{E}^\sigma_{\mathsf{G}^{\mathcal{C}},s}(X_{i,n}^\theta) \ \leqslant \ \varepsilon\,.$$

Proof. Consider any probabilistic or expected reachability nonzero-sum formula θ, state s and $1 \leqslant i \leqslant m$. Using Assumption 6 we have that for subformulae $\mathtt{R}^r[\mathtt{F}\ \phi^i]$, the set $Sat(\phi^i)$ is reached with probability 1 from all states of G under all profiles, and therefore $\mathbb{E}^\sigma_{\mathsf{G}^{\mathcal{C}},s}(X_i^\theta)$ is finite. Furthermore, for any $n \in N$, by Definitions 6.2.2 and A.1.5 we have that $\mathbb{E}^\sigma_{\mathsf{G}^{\mathcal{C}},s}(X_{i,n}^\theta)$ is the value of state s for the nth iteration of value iteration [47] when computing $\mathbb{E}^\sigma_{\mathsf{G}^{\mathcal{C}},s}(X_i^\theta)$ in the DTMC obtained from $\mathsf{G}^{\mathcal{C}}$ by following the strategy σ. It therefore follows that the sequence is both non-decreasing and converges uniformly. □

In the proof of correctness, we will use the fact that n iterations of value iteration is equivalent to performing backward induction on the following game trees.

Definition A.1.6. For any state s and $n \in \mathbb{N}$, let $\mathsf{G}^{\mathcal{C}}_{n,s}$ be the game tree corresponding to playing $\mathsf{G}^{\mathcal{C}}$ for n steps when starting from state s and then terminating.

We can map any strategy profile σ of $\mathsf{G}^{\mathcal{C}}$ to a strategy profile of $\mathsf{G}^{\mathcal{C}}_{n,s}$ by only considering the choices of the profile over the first n steps when starting from state s. This mapping is clearly surjective, i.e. we can generate all profiles of $\mathsf{G}^{\mathcal{C}}_{n,s}$, but is not injective. We require the following lemma relating the values of the objectives $X^{\theta}_{i,n}$ and X^{θ}_{i} over $\mathsf{G}^{\mathcal{C}}$ and $\mathsf{G}^{\mathcal{C}}_{n,s}$ for any $n \in \mathbb{N}$ and $s \in S$.

Lemma 11. For any probabilistic or expected reachability nonzero-sum formula θ, state s of $\mathsf{G}^{\mathcal{C}}$, strategy profile σ, $n \in \mathbb{N}$ and $1 \leqslant i \leqslant m : \mathbb{E}^{\sigma}_{\mathsf{G}^{\mathcal{C}},s}(X^{\theta}_{i,n}) = \mathbb{E}^{\sigma}_{\mathsf{G}^{\mathcal{C}}_{n,s}}(X^{\theta}_{i})$.

Proof. The proof follows from Definitions A.1.5 and A.1.6, in particular, we have that $X^{\theta}_{i,n}$ is the n-step approximations of X^{θ}_{i} and $\mathsf{G}^{\mathcal{C}}_{n,s}$ corresponds to playing game $\mathsf{G}^{\mathcal{C}}$ from state s for n steps. □

We now define the strategy profiles synthesised during value iteration.

Definition A.1.7. For any $n \in \mathbb{N}$ and $s \in S$, let $\sigma^{n,s}$ be the strategy profile generated for the game tree $\mathsf{G}^{\mathcal{C}}_{n,s}$ (when considering value iteration as backward induction) and $\sigma^{n,\star}$ be the synthesised strategy profile for $\mathsf{G}^{\mathcal{C}}$ after n iterations.

Before giving the proof of correctness we require the following results.

Lemma 12. For any state s of $\mathsf{G}^{\mathcal{C}}$, probabilistic or expected reachability nonzero-sum formula θ and $n \in \mathbb{N}$ we have that $\sigma^{n,s}$ is a subgame-perfect SWNE of the CSG $\mathsf{G}^{\mathcal{C}}_{n,s}$ for the objectives $(X^{\theta}_{1}, \ldots, X^{\theta}_{m})$.

Proof. The result follows from the fact that, for any $n \in \mathbb{N}$ and $s \in S$, the value iteration procedure selects SWNE, n steps of value iteration corresponds to performing backward induction for the objectives $(X^{\theta}_{1}, \ldots, X^{\theta}_{m})$ in the game $\mathsf{G}^{\mathcal{C}}_{n,s}$ and backward induction returns a subgame-perfect NE [205, 188]. □

The following proposition demonstrates that value iteration converges and depends on Assumptions 5 and 6. Without this assumption convergence cannot be guaranteed as demonstrated by the counterexamples in Appendices C.3 and C.4. Although value iteration converges, unlike value iteration for MDPs or zero-sum games the generated sequence of values is not necessarily non-decreasing.

Proposition 2. For any probabilistic or expected reachability nonzero-sum formula θ and state s, the sequence $\langle \mathsf{V}_{\mathsf{G}^{\mathcal{C}}}(s, \theta, n) \rangle_{n \in \mathbb{N}}$ converges.

Proof. For any state s and $n \in \mathbb{N}$ we can consider $\mathsf{G}^{\mathcal{C}}_{n,s}$ as an m-player infinite-action NFG $\mathsf{N}_{n,s}$ where for $1 \leqslant i \leqslant m$:

- the set of actions of player i equals the set of strategies of player i in $\mathsf{G}^{\mathcal{C}}$;
- for the strategy profile (σ_1, σ_2), the utility function for player i returns $\mathbb{E}^{\sigma}_{\mathsf{G}^{\mathcal{C}}_{n,s}}(X^{\theta}_i)$.

The correctness of this construction relies on the mapping of strategy profiles from the game $\mathsf{G}^{\mathcal{C}}$ to $\mathsf{G}^{\mathcal{C}}_{n,s}$ being surjective. Using Lemma 10, we have that the sequence $\langle \mathsf{N}_{n,s} \rangle_{n \in N}$ of NFGs converges uniformly, and therefore, since $\mathtt{V}_{\mathsf{G}^{\mathcal{C}}}(s, \theta, n)$ are subgame-perfect SWNE values of $\mathsf{G}^{\mathcal{C}}_{n,s}$ (see Lemma 12), the sequence $\langle \mathtt{V}_{\mathsf{G}^{\mathcal{C}}}(s, \theta, n) \rangle_{n \in \mathbb{N}}$ also converges. □

Again, we point out to the fact that a similar convergence result to Proposition 2 has been shown for discounted properties of two-player games in [90].

Theorem 3. For a given probabilistic or expected reachability nonzero-sum formula θ and $\varepsilon > 0$, there exists $N \in \mathbb{N}$ such that for any $n \geqslant N$ the strategy profile $\sigma^{n,\star}$ is a subgame-perfect ε-SWNE for $\mathsf{G}^{\mathcal{C}}$ and the objectives $(X^{\theta_1}, \ldots, X^{\theta_m})$.

Proof. Consider any $\varepsilon > 0$. From Definition A.1.7 for any $s \in S$, $n \geqslant \mathbb{N}$ and $1 \leqslant i \leqslant m$:

$$\mathbb{E}^{\sigma^{n,\star}}_{\mathsf{G}^{\mathcal{C}},s}(X^{\theta}_i) = \mathbb{E}^{\sigma^{n,s}}_{\mathsf{G}^{\mathcal{C}}_{n,s}}(X^{\theta}_i)\,. \tag{A.5}$$

For any $k \in \mathbb{N}$ and $s \in S$, using Lemma 12 we have that $\sigma^{k,s}$ is a NE of $\mathsf{G}^{\mathcal{C}}_{k,s}$, and therefore for any $k \in \mathbb{N}$, $s \in S$ and $1 \leqslant i \leqslant m$:

$$\mathbb{E}^{\sigma^{k,s}}_{\mathsf{G}^{\mathcal{C}}_{k,s}}(X^{\theta}_i) \;=\; \sup_{\sigma_i \in \Sigma_i^{\mathsf{G}^{\mathcal{C}}_{k,s}}} \mathbb{E}^{\sigma^{k,s}_{-i}[\sigma_i]}_{\mathsf{G}^{\mathcal{C}}_{k,s}}(X^{\theta}_i)\,. \tag{A.6}$$

From Lemma 10 there exists $N \in \mathbb{N}$ such that for any $n \geqslant N$, $s \in S$ and $1 \leqslant i \leqslant m$:

$$\sup_{\sigma_i \in \Sigma_i^{\mathsf{G}^{\mathcal{C}}}} \mathbb{E}^{\sigma^{n,\star}_{-i}[\sigma_i]}_{\mathsf{G}^{\mathcal{C}},s}(X^{\theta}_i) - \sup_{\sigma_i \in \Sigma_i^{\mathsf{G}^{\mathcal{C}}}} \mathbb{E}^{\sigma^{n,\star}_{-i}[\sigma_i]}_{\mathsf{G}^{\mathcal{C}},s}(X^{\theta}_{i,n}) \;\leqslant\; \varepsilon\,. \tag{A.7}$$

Therefore, for any $n \geqslant N$, $s \in S$ and $1 \leqslant i \leqslant m$, using (A.5) we have:

$$\begin{aligned}
\mathbb{E}^{\sigma^{n,\star}}_{\mathsf{G}^{\mathcal{C}},s}(X^{\theta}_i) &= \mathbb{E}^{\sigma^{n,s}}_{\mathsf{G}^{\mathcal{C}}_{n,s}}(X^{\theta}_i) \\
&= \sup_{\sigma_i \in \Sigma_i^{\mathsf{G}^{\mathcal{C}}_{n,s}}} \mathbb{E}^{\sigma^{n,s}_{-i}[\sigma_i]}_{\mathsf{G}^{\mathcal{C}}_{n,s}}(X^{\theta}_i) && \text{by (A.6)} \\
&= \sup_{\sigma_i \in \Sigma_i^{\mathsf{G}^{\mathcal{C}}}} \mathbb{E}^{\sigma^{n,\star}_{-i}[\sigma_i]}_{\mathsf{G}^{\mathcal{C}},s}(X^{\theta}_{i,n}) && \text{by Lemma 11} \\
&\leqslant \sup_{\sigma_i \in \Sigma_i^{\mathsf{G}^{\mathcal{C}}}} \mathbb{E}^{\sigma^{n,\star}_{-i}[\sigma_i]}_{\mathsf{G}^{\mathcal{C}},s}(X^{\theta}_i) - \varepsilon && \text{by (A.7) since } n \geqslant N
\end{aligned}$$

and hence, since $\varepsilon > 0$, $s \in S$ and $1 \leqslant i \leqslant m$ were arbitrary, $\sigma^{n,\star}$ is a subgame-perfect ε-NE. It remains to show that the strategy profile is a subgame-perfect social welfare optimal ε-NE, which follows from the fact that when solving the normal form games during value iteration social welfare optimal NE are returned. □

APPENDIX B

Precomputation and Model Checking Algorithms

B.1 Precomputation Algorithms for Zero-sum Formulae

For zero-sum formulae, players are necessarily split into two-coalitions: one whose objective is minimising/maximising the expected value of the random variable associated to its objective, and the other who is acting to oppose that. Given a two-player CSG $\mathsf{G}^{\mathcal{C}}$, where $\mathcal{C} = \{C_1, C_2\} = \{C, N \backslash C\}$, we let $\mathsf{G}^{\overline{\mathcal{C}}}$ be the game were the indices of coalitions C_1 and C_2 are swapped.

Safety. Implementation for computing the set of states satisfying the safety objective $\Box B = \nu X.(Pre_1(X) \wedge B)$.

Algorithm 4 Model checking routine for $\mathrm{G}(X, \mathsf{G}^{\mathcal{C}})$

Input: Set of states $X \subseteq S$.
Output: A subset of X for which the row player has a strategy to remain in X.

1: **procedure** $\mathrm{G}(X, \mathsf{G}^{\mathcal{C}})$
2: $sol_1 \leftarrow S$; $done \leftarrow$ **false**
3: **while** $\neg done$ **do**
4: $\quad sol_2 \leftarrow \textsc{Pre}_1(sol_1, \mathsf{G}^{\mathcal{C}}) \cap X$
5: $\quad done \leftarrow (sol_2 = sol_1)$
6: $\quad sol_1 \leftarrow sol_2$
7: **end while**
8: **return** sol_1
9: **end procedure**

For a state s and a set $X \subseteq S$, we let $A_i^{\mathcal{C}}(s) = A_i^{\mathcal{C}} \cap \Delta(s)$ and $\neg X = S \setminus X$. Furthermore, given a two-player CSG $\mathsf{G}^{\mathcal{C}}$, where $\mathcal{C} = \{C_1, C_2\} = \{C, N \backslash C\}$, we let $\mathsf{G}^{\overline{\mathcal{C}}}$ be the game were the indices of coalitions C_1 and C_2 are swapped.

The pseudocode for the different algorithms is based on an explicit-state representation and all procedures take a coalitional game $\mathsf{G}^{\mathcal{C}}$ as input. The implementation follows from [74] and we present the corresponding functions and objectives in μ-calculus notation as in the original paper. In the following, we let $X, Y, Z \subseteq S$, $V_i \subseteq A_i^{\mathcal{C}}$, $A_Y^s : 2^{A_2^{\mathcal{C}}(s)} \to 2^{A_1^{\mathcal{C}}(s)}$ and $B_X^s : 2^{A_1^{\mathcal{C}}(s)} \to 2^{A_2^{\mathcal{C}}(s)}$. Moreover, by abuse of notation, we let $\delta(s, a, b)$ also refer to the set of possible successor states when playing (a, b) from s.

Predecessor operator. Implementation for computing the set of states satisfying the operator $Pre_1(X)$, following the condition $s \in Pre_1(X)$ *iff* $\exists a \in A_1^{\mathcal{C}}(s)$ such that $\forall b \in A_2^{\mathcal{C}}(s)$, $\delta(s, a, b) \subseteq X$.

Algorithm 5 Model checking routine for $\textsc{Pre}_1(X, \mathsf{G}^{\mathcal{C}})$

Input: Set of states $X \subseteq S$.
Output: A subset of X satisfying the predecessor operator $\textsc{Pre}_1(s, X)$.

```
1: procedure Pre_1(X, G^C)
2: sol ← ∅
3: for all s ∈ X do
4:    if Pre_1(s, X, G^C) then sol ← sol ∪ {s} end if
5: end for
6: return sol
7: end procedure
```

Algorithm 6 Model checking routine for $\textsc{Pre}_1(s, X, \mathsf{G}^{\mathcal{C}})$

```
1: procedure Pre_1(s, X, G^C)
2: sat ← false
3: for all a ∈ A_1^C(s) do
4:    sat ← true
5:    for all b ∈ A_2^C(s) do
6:       sat ← sat ∧ δ^C(s, a, b) ⊆ X
7:    end for
8:    if sat then
9:       break
10:   end if
11: end for
12: return sat
13: end procedure
```

Almost-sure reachability. Implementation for computing the set of states satisfying the reachability objective $\langle\langle 1\rangle\rangle_\lambda \Diamond B = \nu Y.\mu X.(\lambda pre_1(Y, X) \vee B)$, for $\lambda = almost$.

Algorithm 7 Model checking routine for $\mathrm{AF}(X, \mathsf{G}^{\mathcal{C}})$

Input: Set of states $X \subseteq S$.
Output: The subset of states from which player 1 can guarantee to reach a state in X almost-surely.

1: **procedure** $\mathrm{AF}(X, \mathsf{G}^{\mathcal{C}})$
2: $sol_Y \leftarrow S$; $done_Y \leftarrow$ **false**
3: **while** $\neg done_Y$ **do**
4: $done_X \leftarrow$ **false**; $sol_X \leftarrow \varnothing$
5: **while** $\neg done_X$ **do**
6: $sol \leftarrow \textsc{Apre}_1(sol_X, sol_Y, \mathsf{G}^{\mathcal{C}})$
7: $sol \leftarrow sol \cup X$
8: $done_X \leftarrow (sol_X = sol)$
9: $sol_X \leftarrow sol$
10: **end while**
11: $done_Y \leftarrow (sol_Y = sol_X)$
12: $sol_Y \leftarrow sol_X$
13: **end while**
14: **return** sol_Y
15: **end procedure**

Predecessor operator. Implementation for computing the set of states satisfying the operator $\textsc{Apre}_1(Y, X)$, following the condition $s \in \textsc{Apre}_1(Y, X)$ *iff* $B^s_X(A^s_Y(\varnothing)) = A^{\mathcal{C}}_2(s)$.

Algorithm 8 Model checking routine for $\textsc{Apre}_1(X, Y, \mathsf{G}^{\mathcal{C}})$

Input: Sets of states $X, Y \subseteq S$.
Output: A subset of states which, for all actions in $A^{\mathcal{C}}_2$, there exists an action in $A^{\mathcal{C}}_1$ that reaches X while remaining in Y.

1: **procedure** $\textsc{Apre}_1(X, Y, \mathsf{G}^{\mathcal{C}})$
2: $sol_1 \leftarrow \varnothing$
3: **for all** $s \in S$ **do**
4: $sol_2 \leftarrow \mathrm{B}(s, X, \mathrm{A}(s, Y, \varnothing, \mathsf{G}^{\mathcal{C}}), \mathsf{G}^{\mathcal{C}})$
5: **if** $|sol_2| = |A^{\mathcal{C}}_2(s)|$ **then** $sol_1 \leftarrow sol_1 \cup \{s\}$ **end if**
6: **end for**
7: **return** sol_1
8: **end procedure**

Auxiliary method for predecessor operators. Implementation for the auxiliary method for predecessor operators returning the set of actions from $A_1^{\mathcal{C}}$, following the condition $A_Y^s(V_2) = \{a \in A_1^{\mathcal{C}}(s) \mid \forall b \in A_2^{\mathcal{C}}(s).\ \delta(s,a,b) \not\subseteq Y \rightarrow b \in V_2\}$

Algorithm 9 Model checking routine for $\mathrm{A}(s, Y, V_2, \mathsf{G}^{\mathcal{C}})$

Input: State $s \in S$, set of states $Y \subseteq S$ and set of actions $V_2 \subseteq A_2^{\mathcal{C}}$.
Output: A subset of $A_1^{\mathcal{C}}$ which, for all actions in $b \in A_2^{\mathcal{C}}$, either the states reached are in Y or $b \in V_2$.

1: **procedure** $\mathrm{A}(s, Y, V_2, \mathsf{G}^{\mathcal{C}})$
2: $sol \leftarrow \varnothing$; $sat \leftarrow$ **false**
3: **for all** $a \in A_1^{\mathcal{C}}(s)$ **do**
4: $sat \leftarrow$ **true**
5: **for all** $b \in A_2^{\mathcal{C}}(s)$ **do**
6: $sat \leftarrow sat \wedge (\delta^{\mathcal{C}}(s,a,b) \subseteq Y) \vee (b \in V_2))$
7: **end for**
8: **if** sat **then** $sol \leftarrow sol \cup \{a\}$ **end if**
9: **end for**
10: **return** sol
11: **end procedure**

Auxiliary method for predecessor operators. Implementation for the auxiliary method for predecessor operators returning the set of actions from $A_2^{\mathcal{C}}$, following the condition $B_X^s(V_1) = \{b \in A_2^{\mathcal{C}}(s) \mid \exists a \in V_1.\ \delta(s,a,b) \cap X \neq \varnothing\}$

Algorithm 10 Model checking routine for $\mathrm{B}(s, X, V_1, \mathsf{G}^{\mathcal{C}})$

Input: State $s \in S$, set of states $X \subseteq S$ and set of actions $V_1 \subseteq A_1^{\mathcal{C}}$.
Output: A subset of $A_2^{\mathcal{C}}$ for which there exists an action in V_1 such that a state in X is reached.

1: **procedure** $\mathrm{B}(s, X, V_1, \mathsf{G}^{\mathcal{C}})$
2: $sol \leftarrow \varnothing$
3: **for all** $b \in A_2^{\mathcal{C}}(s)$ **do**
4: **for all** $a \in V_1(s)$ **do**
5: **if** $(\delta^{\mathcal{C}}(s,a,b) \cap X) \neq \varnothing)$ **then**
6: $sol \leftarrow sol \cup \{b\}$
7: **break**
8: **end if**
9: **end for**
10: **end for**
11: **return** sol
12: **end procedure**

Eventually always. Implementation for computing the set of states satisfying the Co-Büchi objective $\langle\langle 1\rangle\rangle_\lambda \Diamond\Box B = \nu Z.\mu X.\nu Y.((B \wedge \lambda F pre_1(Z,Y,X)) \vee (\neg B \wedge \lambda pre_1(Z,X)))$, for $\lambda = almost$.

Algorithm 11 Model checking routine for $\mathrm{AFG}(X, \mathsf{G}^{\mathcal{C}})$

Input: Set of states $X \subseteq S$.

Output: The subset of states from which player 1 can guarantee to eventually restrict the game to states in X almost-surely.

1: **procedure** $\mathrm{AFG}(X, \mathsf{G}^{\mathcal{C}})$
2: $sol_Z \leftarrow S$; $done_Z \leftarrow$ **false**
3: **while** $\neg done_Z$ **do**
4: $sol_X \leftarrow \varnothing$;$done_X \leftarrow$ **false**
5: **while** $\neg done_X$ **do**
6: $sol_Y \leftarrow S$; $done_Y \leftarrow$ **false**
7: **while** $\neg done_Y$ **do**
8: $sol \leftarrow (X \cap \mathrm{AF}\textsc{pre}_1(sol_X, sol_Y, sol_Z, \mathsf{G}^{\mathcal{C}})) \cup (\neg X \cap \textsc{Apre}_1(sol_X, sol_Z, \mathsf{G}^{\mathcal{C}}))$
9: $done_Y \leftarrow (sol_Y = sol)$
10: $sol_Y \leftarrow sol$
11: **end while**
12: $done_X \leftarrow (sol_X = sol_Y)$
13: $sol_X \leftarrow sol_Y$
14: **end while**
15: $done_Z \leftarrow (sol_X = sol_Z)$
16: $sol_Z \leftarrow sol_X$
17: **end while**
18: **return** sol_Z
19: **end procedure**

Predecessor operator. Implementation for computing the set of states satisfying the operator $\text{AFPRE}_1(Z, Y, X)$, following the condition $s \in \text{AFPRE}_1(Z, Y, X)$ *iff* $\nu V.(A^s_Z(\varnothing) \wedge A^s_Y(B^s_X(V))) \neq \varnothing$.

Algorithm 12 Model checking routine for $\text{AFPRE}_1(X, Y, Z, \mathsf{G}^{\mathcal{C}})$

Input: Sets of states $X, Y, Z \subseteq S$
Output: Set of states that satisfy the predecessor operator $\text{AFPRE}_1(X, Y)$.

```
1: procedure AFPRE_1(X, Y, Z, G^C)
2: sol ← ∅
3: for all s ∈ S do
4:    if AFPRE_1(s, X, Y, Z, G^C) then
5:       sol ← sol ∪ {s}
6:    end if
7: end for
8: return sol
9: end procedure
```

Algorithm 13 Model checking routine for $\text{AFPRE}_1(s, X, Y, Z, \mathsf{G}^{\mathcal{C}})$

```
1: procedure AFPRE_1(s, X, Y, Z, G^C)
2: sol_1 ← A^C_1(s); done ← false
3: while ¬done do
4:    sol_2 ← A(s, Z, ∅, G^C) ∩ A(s, Y, B(s, X, sol_1, G^C), G^C)
5:    done ← (sol_1 = sol_2)
6:    sol_1 ← sol_2
7: end while
8: return sol_1 ≠ ∅
9: end procedure
```

B.2 Algorithms for Zero-sum rPATL Formulae

In this section, we present pseudocode for the model checking algorithms for zero-sum properties of CSGs. The algorithms are implementations of the methods developed in Chapter 4, which, given a rPATL formula, proceeds by recursively computing the sets of states satisfying each sub-formula. We employ an undefined method CheckConvergence, which checks whether the maximal difference between values of two consecutive iterations are smaller than ε according to a convergence criterion. We assume that a coalitional game $\mathsf{G}^{\mathcal{C}}$ has been built beforehand.

Algorithm 14 Model checking routine for rPATL (zero-sum formulae).

procedure ModelCheck($\phi, \mathsf{G}^{\mathcal{C}}$)
$sat \leftarrow \varnothing$
if $\phi = \top$ **then**
  $sat \leftarrow S$
else if $\phi = l \in AP$ **then**
  $sat \leftarrow \{s \in S \mid l \in L(s)\}$
else if $\phi = \neg\phi_1$ **then**
  $sat \leftarrow S \setminus$ ModelCheck(ϕ_1, $\mathsf{G}^{\mathcal{C}}$)
else if $\phi = \phi_1 \wedge \phi_2$ **then**
  $sat \leftarrow$ ModelCheck(ϕ_1, $\mathsf{G}^{\mathcal{C}}$) $\cap$ ModelCheck(ϕ_2, $\mathsf{G}^{\mathcal{C}}$)
else if $\phi = \langle\langle C \rangle\rangle \mathtt{P}_{\sim x}[\,\psi\,]$ **then**
  $sat \leftarrow$ ModelCheckProb(ϕ, $\mathsf{G}^{\mathcal{C}}$)
else if $\phi = \langle\langle C \rangle\rangle \mathtt{R}^{r}_{\sim x}[\,\psi\,]$ **then**
  $sat \leftarrow$ ModelCheckRew(ϕ, $\mathsf{G}^{\mathcal{C}}$)
end if
return sat
end procedure

Algorithm 15 Model checking routine for the reward operator.

procedure ModelCheckRew($\phi, \mathsf{G}^{\mathcal{C}}$)
if $\rho = \mathtt{F}^c\ \phi$ **then**
  $sat \leftarrow$ ModelCheckRewFc($\langle\langle C \rangle\rangle \mathtt{R}^r_{\sim x}[\,\mathtt{F}^c\ \phi\,], \mathsf{G}^{\mathcal{C}}$)
else if $\rho = \mathtt{F}\ \phi$ **then**
  $sat \leftarrow$ ModelCheckRewFinf($\langle\langle C \rangle\rangle \mathtt{R}^r_{\sim x}[\,\mathtt{F}\ \phi\,], \mathsf{G}^{\mathcal{C}}$)
else if $\rho = \mathtt{C}$ **then**
  $sat \leftarrow$ ModelCheckRewFc($\langle\langle C \rangle\rangle \mathtt{R}^r_{\sim x}[\,\mathtt{C}\,], \mathsf{G}^{\mathcal{C}}$)
else if $\rho = \mathtt{C}^{\leqslant k}$ **then**
  $sat \leftarrow$ ModelCheckRewCumul($\langle\langle C \rangle\rangle \mathtt{R}^r_{\sim x}[\,\mathtt{C}^{\leqslant k}\,], \mathsf{G}^{\mathcal{C}}$)
else if $\rho = \mathtt{I}^{=k}$ **then**
  $sat \leftarrow$ ModelCheckRewInst($\langle\langle C \rangle\rangle \mathtt{R}^r_{\sim x}[\,\mathtt{I}^{=k}\phi\,], \mathsf{G}^{\mathcal{C}}$)
end if
return sat
end procedure

Algorithm 16 Model checking routine for the probability operator (Next).

procedure ModelCheckProb($\phi, \mathsf{G}^{\mathcal{C}}$)
$sat \leftarrow \varnothing$
if $\psi = \mathtt{X}\,\phi$ **then**
  $T \leftarrow$ ModelCheck($\phi,\ \mathsf{G}^{\mathcal{C}}$)
  for all $s \in S$ **do**
    if opt = min **then**
      $val \leftarrow \min\limits_{a_1 \in A_1^{\mathcal{C}}} \max\limits_{a_2 \in A_2^{\mathcal{C}}} \sum\limits_{t \in T} \delta^{\mathcal{C}}(s, a_1, a_2)(t)$
    else
      $val \leftarrow \max\limits_{a_1 \in A_1^{\mathcal{C}}} \min\limits_{a_2 \in A_2^{\mathcal{C}}} \sum\limits_{t \in T} \delta^{\mathcal{C}}(s, a_1, a_2)(t)$
    end if
    if $val \sim x$ **then**
      $sat \leftarrow sat \cup \{s\}$
    end if
  end for

Algorithm 16 Model checking routine for the probability operator (Until).

15: **else if** $\psi = \phi_1 \ \mathtt{U} \ \phi_2$ **or** $\psi_1 \ \mathtt{U}^{\leqslant k} \ \phi_2$ **then**
16: $T_2 \leftarrow \textsc{ModelCheck}(\phi_2, \ \mathsf{G}^{\mathcal{C}})$
17: $T_1 \leftarrow \textsc{ModelCheck}(\phi_1, \ \mathsf{G}^{\mathcal{C}}) \setminus T_2$
18: $iter \leftarrow 0$; $X \leftarrow 0$; $X' \leftarrow 0$
19: **for all** $s \in S$ **do**
20: **if** $s \in T_2$ **then** $X[s] \leftarrow 1$ **else** $X[s] \leftarrow 0$ **end if**
21: **end for**
22: **while true do**
23: $X' \leftarrow X$
24: $iter \leftarrow iter + 1$
25: **for all** $s \in S$ **do**
26: **if** $s \in T_1$ **then**
27: **if** opt $=$ min **then**
28: $X[s] \leftarrow \min\limits_{a_1 \in A_1^{\mathcal{C}}} \max\limits_{a_2 \in A_2^{\mathcal{C}}} \sum\limits_{t \in S} \delta^{\mathcal{C}}(s, a_1, a_2, t) \cdot X'[t]$
29: **else**
30: $X[s] \leftarrow \max\limits_{a_1 \in A_1^{\mathcal{C}}} \min\limits_{a_2 \in A_2^{\mathcal{C}}} \sum\limits_{t \in S} \delta^{\mathcal{C}}(s, a_1, a_2, t) \cdot X'[t]$
31: **end if**
32: **end if**
33: **if** $X[s] \sim x$ **then**
34: $sat \leftarrow sat \cup \{s\}$
35: **end if**
36: **end for**
37: **if** $k \leqslant iter$ **then**
38: **break**
39: **else if** $\textsc{CheckConvergence}(\varepsilon, X, X')$ **then**
40: **break**
41: **end if**
42: **end while**
43: **end if**
44: **return** sat
45: **end procedure**

Algorithm 17 Model checking routing for the reward $\mathtt{I}^{=k}$ operator.

procedure ModelCheckRewInst($\phi, \mathsf{G}^{\mathcal{C}}$)
$iter \leftarrow 0$
for all $s \in S$ **do**
  $X[s] \leftarrow r^{\mathcal{C}}_S(s)$
end for
while true do
  for all $s \in S$ **do**
    $X'[s] \leftarrow X[s]$
    if opt = min **then**
      $X[s] \leftarrow \min\limits_{a_1 \in A_1^{\mathcal{C}}} \max\limits_{a_2 \in A_2^{\mathcal{C}}} \sum\limits_{t \in S} \delta(s, a_1, a_2, t) \cdot X'[t]$
    else
      $X[s] \leftarrow \max\limits_{a_1 \in A_1^{\mathcal{C}}} \min\limits_{a_2 \in A_2^{\mathcal{C}}} \sum\limits_{t \in S} \delta(s, a_1, a_2, t) \cdot X'[t]$
    end if
    if $X[s] \sim x$ **then**
      $sat \leftarrow sat \cup \{s\}$
    end if
  end for
  $iter \leftarrow iter + 1$
  if $k \leqslant iter$ **then**
    break
  else if CheckConvergence(ε, X, X') **then**
    break
  end if
end while
return sat
end procedure

Algorithm 18 Model checking routing for the reward $\mathsf{C}^{\leqslant k}$ operator.

procedure ModelCheckRewCumul($\phi, \mathsf{G}^{\mathcal{C}}$)
$iter \leftarrow 0$; $X \leftarrow 0$; $X' \leftarrow 0$
while true do
  for all $s \in S$ **do**
    $X'[s] \leftarrow X[s]$
    if opt = min **then**
      $X[s] \leftarrow r^{\mathcal{C}}(s) + \min\limits_{a_1 \in A_1^{\mathcal{C}}} \max\limits_{a_2 \in A_2^{\mathcal{C}}} (r^{\mathcal{C}}(s, a_1, a_2) + \sum\limits_{t \in S} \delta(s, a_1, a_2, t) \cdot X'[t])$
    else
      $X[s] \leftarrow r^{\mathcal{C}}(s) + \max\limits_{a_1 \in A_1^{\mathcal{C}}} \min\limits_{a_2 \in A_2^{\mathcal{C}}} (r^{\mathcal{C}}(s, a_1, a_2) + \sum\limits_{t \in S} \delta(s, a_1, a_2, t) \cdot X'[t])$
    end if
    if $X[s] \sim x$ **then**
      $sat \leftarrow sat \cup \{s\}$
    end if
  end for
  $iter \leftarrow iter + 1$
  if $k \leqslant iter$ **then**
    break
  else if CheckConvergence(ε, X, X') **then**
    break
  end if
end while
return sat
end procedure

Algorithm 19 Model checking routine for the $\mathtt{F}^c$ and $\mathtt{C}$ reward operators.

1: **procedure** ModelCheckRewFc($\phi, \mathsf{G}^{\mathcal{C}}$)
2: $T \leftarrow$ ModelCheck(ϕ, $\mathsf{G}^{\mathcal{C}}$)
3: $AP \leftarrow AP \cup \{a_{rew}\}$
4: **for all** $s \in S$ **do**
5: **if** $r^{\mathcal{C}}(s) > 0$ **then** $L(s) \leftarrow L(s \cup \{a_{rew}\})$ **end if**
6: **end for**
7: $S^{\infty} \leftarrow S \setminus \mathrm{AFG}(\text{ModelCheck}(a_{rew})\mathsf{G}^{\overline{\mathcal{C}}})$
8: **if** $\psi = \mathtt{F}^c\ \phi$ **then**
9: $T \leftarrow$ ModelCheck(ϕ, $\mathsf{G}^{\mathcal{C}}$)
10: $S \leftarrow S \setminus T$
11: **end if**
12: $S \leftarrow S \setminus S^{\infty}$
13: $iter \leftarrow 0; sat \leftarrow \varnothing; unsat \leftarrow \varnothing; X \leftarrow 0; X' \leftarrow 0$
14: **while true do**
15: $X' \leftarrow X$
16: **for all** $s \in S$ **do**
17: **if** opt = min **then**
18: $X[s] \leftarrow r^{\mathcal{C}}(s) + \min\limits_{a_1 \in A_1^{\mathcal{C}}} \max\limits_{a_2 \in A_2^{\mathcal{C}}} (r^{\mathcal{C}}(s, a_1, a_2) + \sum\limits_{t \in S} \delta(s, a_1, a_2, t) \cdot X'[t])$
19: **else**
20: $X[s] \leftarrow r^{\mathcal{C}}(s) + \max\limits_{a_1 \in A_1^{\mathcal{C}}} \min\limits_{a_2 \in A_2^{\mathcal{C}}} (r^{\mathcal{C}}(s, a_1, a_2) + \sum\limits_{t \in S} \delta(s, a_1, a_2, t) \cdot X'[t])$
21: **end if**
22: **if** $X[s] \sim x$ **then**
23: $sat \leftarrow sat \cup \{s\}$
24: **end if**
25: **end for**
26: $iter \leftarrow iter + 1$
27: **if** CheckConvergence(ε, X, X') **then**
28: break
29: **end if**
30: **end while**
31: **return** *sat*
32: **end procedure**

Algorithm 20 Model checking routine for the `F` reward operator.

procedure ModelCheckRewInf($\phi, \mathsf{G}^{\mathcal{C}}$)
$T \leftarrow$ ModelCheck(ϕ, $\mathsf{G}^{\mathcal{C}}$)
$S^{\infty} \leftarrow S \setminus \mathrm{AF}(T, \mathsf{G}^{\overline{\mathcal{C}}})$
$S \leftarrow S \setminus S^{\infty}$
if $\sim \in \{\geqslant, >\}$ **then**
  $sat \leftarrow S^{\infty}$
else
  $sat \leftarrow \varnothing$
end if
$X \leftarrow 0; X' \leftarrow 0$
while true do
  $X' \leftarrow X$
  for all $s \in S \setminus T$ **do**
    if opt $=$ min **then**
      $X[s] \leftarrow (r^{\mathcal{C}}(s) + \gamma) + \min\limits_{a_1 \in A_1^{\mathcal{C}}} \max\limits_{a_2 \in A_2^{\mathcal{C}}} (r^{\mathcal{C}}(s, a_1, a_2) + \sum\limits_{t \in S} \delta(s, a_1, a_2, t) \cdot X'[t])$
    else
      $X[s] \leftarrow (r^{\mathcal{C}}(s) + \gamma) + \max\limits_{a_1 \in A_1^{\mathcal{C}}} \min\limits_{a_2 \in A_2^{\mathcal{C}}} (r^{\mathcal{C}}(s, a_1, a_2) + \sum\limits_{t \in S} \delta(s, a_1, a_2, t) \cdot X'[t])$
    end if
  end for
  if CheckConvergence(ε, X, X') **then**
    if $\gamma > 0$ **then**
      $\gamma \leftarrow 0$
    else
      break
    end if
  end if
end while
for all $s \in S$ **do**
  if $X[s] \sim x$ **then**
    $sat \leftarrow sat \cup \{s\}$
  end if
end for
return sat
end procedure

B.3 Algorithms for Two-player Equilibria rPATL Formulae

In this section, we present pseudocode for the model checking algorithms for two-player nonzero-sum properties of CSGs. The algorithms are implementations of the methods developed in Chapter 5, which, given a rPATL formula, proceeds by recursively computing the sets of states satisfying each sub-formula. The list of algorithms presented in this section is not exhaustive and focuses on the more intricate operators (and combinations) of rPATL. More specifically, we do not include algorithms for computation of formulae with the *next* operator and also do not include pseudocode for the SMT-based implementation for computing SNWE or SCNE values for bimatrix games, here referred to as OptNash. We also do not include code for the computation of mixed nonzero-sum formulae as, in that case, the finite-horizon sub-formula is transformed into infinite-horizon and thus the algorithms for infinite-horizon properties could be used, along with the model transformations described in Section 5.3.5. As for zero-sum formulae, for infinite-horizon properties we employ an undefined method CheckConvergence, which checks whether the values between to consecutive iterations are ε-close according to a convergence criterion. We assume that a coalitional game $\mathsf{G}^{\mathcal{C}}$ has been built beforehand.

Algorithm 21 Model checking routine for rPATL (two-player nonzero-sum formulae).

procedure ModelCheck($\phi, \mathsf{G}^{\mathcal{C}}$)
  $sat \leftarrow \varnothing$
  if $\phi = \top$ **then**
    $sat \leftarrow S$
  else if $\phi = l \in AP$ **then**
    $sat \leftarrow \{s \in S \mid l \in L(s)\}$
  else if $\phi = \neg\phi_1$ **then**
    $sat \leftarrow S \setminus$ ModelCheck(ϕ_1, $\mathsf{G}^{\mathcal{C}}$)
  else if $\phi = \phi_1 \wedge \phi_2$ **then**
    $sat \leftarrow$ ModelCheck(ϕ_1, $\mathsf{G}^{\mathcal{C}}$) $\cap$ ModelCheck(ϕ_2, $\mathsf{G}^{\mathcal{C}}$)
  else if $\phi = \langle\langle C{:}C' \rangle\rangle_{\text{opt}\sim x}(\mathtt{P}[\,\psi_1\,] + \mathtt{P}[\,\psi_2\,])$ **then**
    $sat \leftarrow$ ModelCheckEqProb(ϕ, $\mathsf{G}^{\mathcal{C}}$)
  else if $\phi = \langle\langle C{:}C' \rangle\rangle_{\text{opt}\sim x}(\mathtt{R}^{r_1}[\,\rho_1\,] + \mathtt{R}^{r_2}[\,\rho_2\,])$ **then**
    $sat \leftarrow$ ModelCheckEqRew(ϕ, $\mathsf{G}^{\mathcal{C}}$)
  end if
  return sat
end procedure

Algorithm 22 Model checking routine for the probability operator.

procedure (ModelCheckEqProb($\phi, \mathsf{G}^{\mathcal{C}}$)
if $\psi_1 = \phi_1^1 \ \mathtt{U} \ \phi_2^1$ **and** $\psi_2 = \phi_1^2 \ \mathtt{U} \ \phi_2^2$ **then**
  $sat \leftarrow$ ModelCheckEqInfHorizonProb($\phi, \mathsf{G}^{\mathcal{C}}$)
else if ($\psi_1 = \phi_1^1 \ \mathtt{U}^{\leqslant k_1} \ \phi_2^1$ **or** $\psi_1 = \mathtt{X} \, \phi^1$) **and** ($\psi_2 = \phi_1^2 \ \mathtt{U}^{\leqslant k_2} \ \phi_2^2$ **or** $\psi_2 = \mathtt{X} \, \phi^2$) **then**
  $sat \leftarrow$ ModelCheckEqFinHorizonRew($\phi, \mathsf{G}^{\mathcal{C}}$)
else
  $sat \leftarrow$ ModelCheckEqMixedProb($\phi, \mathsf{G}^{\mathcal{C}}$)
end if
return *sat*
end procedure

Algorithm 23 Model checking routine for the reward operator.

procedure ModelCheckEqRew($\phi, \mathsf{G}^{\mathcal{C}}$)
if $\psi_1 = \mathtt{F} \ \phi^1$ **and** $\psi_2 = \mathtt{F} \ \phi^2$ **then**
  $sat \leftarrow$ ModelCheckEqInfHorizonRew($\phi, \mathsf{G}^{\mathcal{C}}$)
else if ($\psi_1 = \mathtt{I}^{=k_1}$ **or** $\psi_1 = \mathtt{C}^{\leqslant k_1}$) **and** ($\psi_2 = \mathtt{I}^{=k_2}$ **or** $\psi_2 = \mathtt{C}^{\leqslant k_2}$) **then**
  $sat \leftarrow$ ModelCheckEqFinHorizonRew($\phi, \mathsf{G}^{\mathcal{C}}$)
else
  $sat \leftarrow$ ModelCheckEqMixedRew($\phi, \mathsf{G}^{\mathcal{C}}$)
end if
return *sat*
end procedure

Algorithm 24 Model checking routine for finite-horizon probability formulae (Until).

procedure ModelCheckEqFinHorizonProb($\phi, \mathsf{G}^{\mathcal{C}}$)
$sat \leftarrow \varnothing; k \leftarrow |k_1 - k_2|$
$T_{\phi_j^i} \leftarrow$ ModelCheck(ϕ_j^i, $\mathsf{G}^{\mathcal{C}}$)
if $k_1 > k_2$ **then** $n_1 \leftarrow k$ **else** $n_1 \leftarrow 0$ **end if**
if $k_2 > k_1$ **then** $n_2 \leftarrow k$ **else** $n_2 \leftarrow 0$ **end if**
while true do
  if $n_1 > 0$ **then** $X_{\psi_1} \leftarrow \mathtt{P}_{\mathrm{opt}}[\, \phi_1^1 \ \mathtt{U}^{\leqslant n_1} \ \phi_2^1 \,]$ **end if**
  if $n_2 > 0$ **then** $X_{\psi_2} \leftarrow \mathtt{P}_{\mathrm{opt}}[\, \phi_1^2 \ \mathtt{U}^{\leqslant n_2} \ \phi_2^2 \,]$ **end if**

Algorithm 24 Model checking routine for finite-horizon probability formulae (Until)

9: **if** $\min(n_1, n_2) > 0$ **then**
10: **for all** $s \in S$ **do**
11: **if** $s \in T_{\phi_2^1} \cap T_{\phi_2^2}$ **then**
12: $X_1'[s] \leftarrow 1; X_2'[s] \leftarrow 1$
13: **else if** $s \in T_{\phi_2^1}$ **then**
14: $X_1'[s] \leftarrow 1; X_2'[s] \leftarrow X_{\psi_2}[s]$
15: **else if** $s \in T_{\phi_2^2}$ **then**
16: $X_1'[s] \leftarrow X_{\psi_1}[s]; X_2'[s] \leftarrow 1$
17: **else if** $s \in T_{\phi_1^1} \setminus T_{\phi_1^2}$ **then**
18: $X_1'[s] \leftarrow X_{\psi_1}[s]; X_2'[s] \leftarrow 0$
19: **else if** $s \in T_{\phi_1^2} \setminus T_{\phi_1^1}$ **then**
20: $X_1'[s] \leftarrow 0; X_2'[s] \leftarrow X_{\psi_2}[s]$
21: **else if** $s \notin T_{\phi_1^1} \cap T_{\phi_1^2}$ **then**
22: $X_1'[s] \leftarrow 0; X_2'[s] \leftarrow 0$
23: **else**
24: $\text{EQ} \leftarrow \text{OPTNASH}(s, X_1, X_2, \text{opt})$
25: $X_1'[s] \leftarrow \text{EQ}[1]; X_2'[s] \leftarrow \text{EQ}[2]$
26: **end if**
27: **end for**
28: $X_1 \leftarrow X_1'$; $X_2 \leftarrow X_2'$
29: **else**
30: **for all** $s \in S$ **do**
31: **if** $n_1 = 0$ **and** $n_2 = 0$ **then**
32: **if** $s \in T_{\phi_2^1}$ **then** $X_1[s] \leftarrow 1$ **else** : **end if** $X_1[s] \leftarrow 0$
33: **if** $s \in T_{\phi_2^2}$ **then** $X_2[s] \leftarrow 1$ **else** : **end if** $X_2[s] \leftarrow 0$
34: **else if** $n_1 = 0$ **then**
35: **if** $s \in T_{\phi_2^1}$ **then** $X_1[s] \leftarrow 1$ **else** : **end if** $X_1[s] \leftarrow 0$
36: **if** $s \in T_{\phi_2^2}$ **then** $X_2[s] \leftarrow 1$ **else** : **end if** $X_2[s] \leftarrow X_{\psi_2}[s]$
37: **else**
38: **if** $s \in T_{\phi_2^1}$ **then** $X_1[s] \leftarrow 1$ **else** : **end if** $X_1[s] \leftarrow X_{\psi_1}[s]$
39: **if** $s \in T_{\phi_2^2}$ **then** $X_2[s] \leftarrow 1$ **else** : **end if** $X_2[s] \leftarrow 0$
40: **end if**
41: **end for**
42: **end if**
43: **if** $k = \max(k_1, k_2)$ **then**
44: **break**
45: **end if**
46: $k \leftarrow k + 1$; $n_1 \leftarrow \min(n_1 + 1, k_1)$; $n_2 \leftarrow \min(n_2 + 1, k_2)$
47: **end while**
48: **for all** $s \in S$ **do**
49: **if** $X_1[s] + X_2[s] \sim q$ **then** $sat \leftarrow sat \cup \{s\}$ **end if**
50: **end for**
51: **return** *sat*
52: **end procedure**

Algorithm 25 Model checking routine for finite-horizon reward formulae.

procedure ModelCheckEqFinHorizonRew($\phi, \mathsf{G}^{\mathcal{C}}$)
$sat \leftarrow \varnothing; k \leftarrow |k_1 - k_2|$
if $k_1 > k_2$ **then** $n_1 \leftarrow k$ **else** $n_1 \leftarrow 0$ **end if**
if $k_2 > k_1$ **then** $n_2 \leftarrow k$ **else** $n_2 \leftarrow 0$ **end if**
while true do
  if $n_1 > 0$ **then**
    if $\rho_1 = \mathtt{I}^{=k_1}$ **then** $X_{\rho_1} \leftarrow \mathtt{R}^{r_1}_{\mathrm{opt}}[\,\mathtt{I}^{=n_1}\,]$ **else** $X_{\rho_1} \leftarrow \mathtt{R}^{r_1}_{\mathrm{opt}}[\,\mathtt{C}^{\leqslant n_1}\,]$ **end if**
  end if
  if $n_2 > 0$ **then**
    if $\rho_2 = \mathtt{I}^{=k_2}$ **then** $X_{\rho_2} \leftarrow \mathtt{R}^{r_2}_{\mathrm{opt}}[\,\mathtt{I}^{=n_2}\,]$ **else** $X_{\rho_2} \leftarrow \mathtt{R}^{r_2}_{\mathrm{opt}}[\,\mathtt{C}^{\leqslant n_2}\,]$ **end if**
  end if
  if $\min(n_1, n_2) > 0$ **then**
    for all $s \in S$ **do**
      $\mathrm{Eq} \leftarrow \mathrm{OptNash}(s, X_1, X_2, \mathrm{opt})$
      $X_1'[s] \leftarrow \mathrm{Eq}[1]; X_2'[s] \leftarrow \mathrm{Eq}[2]$
    end for
    $X_1 \leftarrow X_1'$; $X_2 \leftarrow X_2'$
  else
    for all $s \in S$ **do**
      if $n_1 = 0$ **and** $n_2 = 0$ **then**
        if $\rho_1 = \mathtt{I}^{=k_1}$ **then** $X_1[s] \leftarrow r^1_S(s)$ **else** $X_1[s] \leftarrow 0$ **end if**
        if $\rho_2 = \mathtt{I}^{=k_2}$ **then** $X_2[s] \leftarrow r^2_S(s)$ **else** $X_2[s] \leftarrow 0$ **end if**
      else if $n_1 = 0$ **then**
        if $\rho_1 = \mathtt{I}^{=k_1}$ **then** $X_1[s] \leftarrow r^1_S(s)$ **else** $X_1[s] \leftarrow 0$ **end if**
        $X_2[s] \leftarrow X_{\rho_2}[s]$
      else
        $X_1[s] \leftarrow X_{\rho_1}[s]$
        if $\rho_2 = \mathtt{I}^{=k_2}$ **then** $X_2[s] \leftarrow r^2_S(s)$ **else** $X_2[s] \leftarrow 0$ **end if**
      end if
    end for
  end if
  if $k = \max(k_1, k_2)$ **then**
    break
  end if
  $k \leftarrow k + 1$; $n_1 \leftarrow \min(n_1 + 1, k_1)$; $n_2 \leftarrow \min(n_2 + 1, k_2)$
end while
for all $s \in S$ **do**
  if $X_1[s] + X_2[s] \sim q$ **then**
    $sat \leftarrow sat \cup \{s\}$
  end if
end for
return sat
end procedure

Algorithm 26 Model checking routine for infinite-horizon probability formulae (Until).

procedure ModelCheckEqInfHorizonProb(ϕ, $\mathsf{G}^{\mathcal{C}}$)
$sat \leftarrow \varnothing; R \leftarrow \varnothing$
$T_{\phi_j^i} \leftarrow$ ModelCheck(ϕ_j^i, $\mathsf{G}^{\mathcal{C}}$)
$X_{\psi_1} \leftarrow \mathtt{P}_{\text{opt}}[\, \phi_1^1 \ \mathtt{U} \ \phi_2^1 \,]$
$X_{\psi_2} \leftarrow \mathtt{P}_{\text{opt}}[\, \phi_1^2 \ \mathtt{U} \ \phi_2^2 \,]$
for all $s \in S$ **do**
  $R \leftarrow R \cup \{s\}$
  if $s \in T_{\phi_2^1} \cap T_{\phi_2^2}$ **then**
    $X_1[s] \leftarrow 1; X_2[s] \leftarrow 1$
  else if $s \in T_{\phi_2^1}$ **then**
    $X_1[s] \leftarrow 1; X_2[s] \leftarrow X_{\psi_2}[s]$
  else if $s \in T_{\phi_2^2}$ **then**
    $X_1[s] \leftarrow X_{\psi_1}[s]; X_2[s] \leftarrow 1$
  else if $s \in T_{\phi_1^1} \setminus T_{\phi_1^2}$ **then**
    $X_1[s] \leftarrow X_{\psi_1}[s]; X_2[s] \leftarrow 0$
  else if $s \in T_{\phi_1^2} \setminus T_{\phi_1^1}$ **then**
    $X_1[s] \leftarrow 0; X_2[s] \leftarrow X_{\psi_2}[s]$
  else if $s \notin T_{\phi_1^1} \cap T_{\phi_1^2}$ **then**
    $X_1[s] \leftarrow 0; X_2[s] \leftarrow 0$
  else
    $X_1[s] \leftarrow 0; X_2[s] \leftarrow 0; R \leftarrow R \setminus \{s\}$
  end if
end for
while true do
  $X_1' \leftarrow X_1; X_2' \leftarrow X_2$
  for all $s \in S \setminus R$ **do**
    Eq $\leftarrow$ OptNash(s, X_1', X_2', opt)
    $X_1[s] \leftarrow$ Eq[1]; $X_2[s] \leftarrow$ Eq[2]
  end for
  if CheckConvergence($\varepsilon, X_1, X_1', X_2, X_2'$) **then**
    break
  end if
end while
for all $s \in S$ **do**
  if $X_1[s] + X_2[s] \sim q$ **then**
    $sat \leftarrow sat \cup \{s\}$
  end if
end for
return sat
end procedure

Algorithm 27 Model checking routine for infinite-horizon reward formulae.

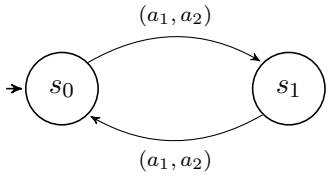
procedure ModelCheckEqInfHorizonRew($\phi, \mathsf{G}^{\mathcal{C}}$)
$sat \leftarrow \varnothing; R \leftarrow \varnothing$
$T_{\phi^i} \leftarrow$ ModelCheck(ϕ^i, $\mathsf{G}^{\mathcal{C}}$)
$X_{\psi_1} \leftarrow \mathtt{R}^{r_1}_{\mathrm{opt}}[\,\mathtt{F}\ \phi^1\,]$
$X_{\psi_2} \leftarrow \mathtt{R}^{r_2}_{\mathrm{opt}}[\,\mathtt{F}\ \phi^2\,]$
for all $s \in S$ **do**
  $R \leftarrow R \cup \{s\}$
  if $s \in T_{\phi^1} \cap T_{\phi^2}$ **then**
    $X_1[s] \leftarrow 0; X_2[s] \leftarrow 0$
  else if $s \in T_{\phi_1}$ **then**
    $X_1[s] \leftarrow 0; X_2[s] \leftarrow X_{\psi_2}[s]$
  else if $s \in T_{\phi_2}$ **then**
    $X_1[s] \leftarrow X_{\psi_1}[s]; X_2[s] \leftarrow 0$
  else
    $X_1[s] \leftarrow 0; X_2[s] \leftarrow 0; R \leftarrow R \setminus \{s\}$
  end if
end for
while true do
  $X_1' \leftarrow X_1; X_2' \leftarrow X_2$
  for all $s \in S \setminus R$ **do**
    Eq $\leftarrow$ OptNash($s, X_1', X_2', \mathrm{opt}$)
    $X_1[s] \leftarrow$ Eq[1]; $X_2[s] \leftarrow$ Eq[2]
  end for
  if CheckConvergence($\varepsilon, X_1, X_1', X_2, X_2'$) **then**
    break
  end if
end while
for all $s \in S$ **do**
  if $X_1[s] + X_2[s] \sim q$ **then**
    $sat \leftarrow sat \cup \{s\}$
  end if
end for
return sat
end procedure

APPENDIX C

Convergence and Assumptions

C.1 Convergence of Zero-sum Total Reward Formulae

In this section, we give a witness to the failure of convergence for value iteration for zero-sum total expected reward properties if Assumption 1 does not hold.



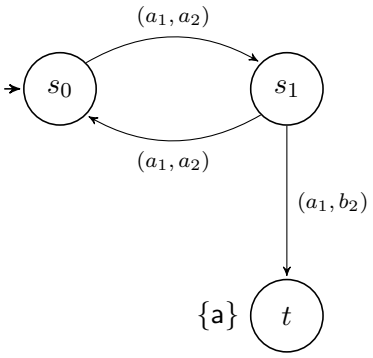
$$r_A(s, (a_1, a_2)) = \begin{cases} 1 & \text{if } s = s_0 \\ -1 & \text{if } s = s_1 \\ 0 & \text{otherwise} \end{cases}$$

Figure C.1: Counterexample for zero-sum total reward properties.

Consider the CSG in Figure C.1 with players p_1 and p_2 and the zero-sum state formula $\phi = \langle\langle p_1, p_2 \rangle\rangle \mathtt{R}^r_{\max=?}[\,\mathtt{C}\,]$. In the CSG, we have that $r_S(s) = 0$ for all s. Clearly, state s_1 does not reach an absorbing state while the reward for the state-action pair $(s_1, (a_1, a_2))$ is negative. Applying the value iteration algorithm of Section 4.5.2, we see that the values for state s_0 oscillate between 0 and 1, whereas the values for state s_1 oscillate between 0 and -1.

C.2 Convergence of Zero-sum Reachability Reward Formulae

In this section, we give a witness to the failure of convergence for value iteration when verifying zero-sum formulae with an infinite horizon reward objective if Assumption 2 does not hold.



$$r_A(s, (c_1, c_2)) = \begin{cases} -1 & \text{if } s = s_0 \text{ and } (c_1, c_2) = (a_1, a_2) \\ 1 & \text{if } s = s_1 \text{ and } (c_1, c_2) = (a_1, a_2) \\ 0 & \text{otherwise} \end{cases}$$

Figure C.2: Counterexample for zero-sum expected reachability reward properties.

Consider the CSG in Figure C.2 with players p_1 and p_2 and the zero-sum state formula $\phi = \langle\langle p_1, p_2 \rangle\rangle \mathtt{R}^r_{\max=?}[\,\mathtt{F}\ \mathtt{a}\,]$, where $\mathtt{a}$ is the atomic proposition satisfied only by state t. In the CSG, we have that $r_S(s) = 0$ for all s. Clearly, state s_0 does not reach either the target of the formula or an absorbing state with probability 1 under all strategy profiles, while the reward for the state-action pair $(s_0, (a_1, a_2))$ is negative. Applying the value iteration algorithm of Section 4.5.2, we see that the values for state s_0 oscillate between 0 and -1, while the values for state s_1 oscillate between 0 and 1.

C.3 Convergence of Nonzero-sum Probabilistic Reachability Properties

In this section, we give a witness to the failure of convergence for value iteration when verifying nonzero-sum formulae with infinite horizon probabilistic objectives if Assumption 3 does not hold.

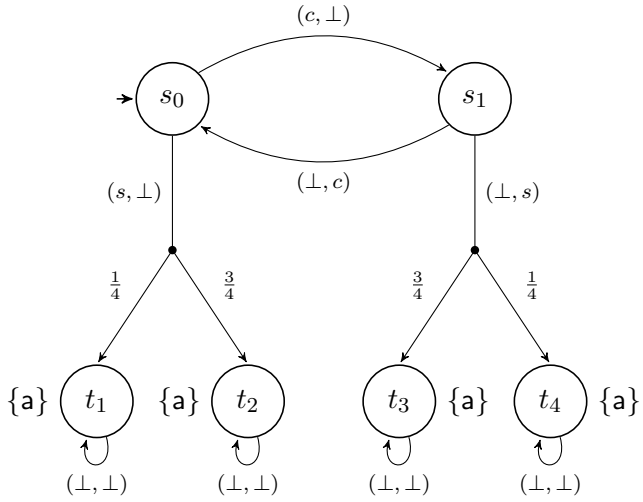


Figure C.3: Counterexample for nonzero-sum probabilistic reachability properties.

Consider the CSG in Figure C.3 with players p_1 and p_2 (an adaptation of a TSG example from [30]) and the nonzero-sum state formula $\langle\langle p_1{:}p_2 \rangle\rangle_{\max=?}(\theta)$, where $\theta = \mathtt{P}[\,\mathtt{F}\ \mathsf{a}_1\,]{+}\mathtt{P}[\,\mathtt{F}\ \mathsf{a}_2\,]$ and a_i is the atomic proposition satisfied only by the state t_i. Clearly, this CSG has a non-terminal end component as one can remain in $\{s_0, s_1\}$ indefinitely or leave at any time.

Applying the value iteration algorithm of Section 5.3.2, we have:

- In the first iteration $\mathtt{V}_{\mathsf{G}^C}(s_0, \theta, 1)$ are the SWNE values of the bimatrix game:

$$\mathsf{Z}_1 \;=\; \begin{matrix} \\ c \\ s \end{matrix} \overset{\bot}{\begin{pmatrix} 0 \\ \frac{1}{4} \end{pmatrix}} \quad \text{and} \quad \mathsf{Z}_2 \;=\; \begin{matrix} \\ c \\ s \end{matrix} \overset{\bot}{\begin{pmatrix} 0 \\ \frac{3}{4} \end{pmatrix}}$$

  i.e. the values $(\frac{1}{4}, \frac{3}{4})$, and $\mathtt{V}_{\mathsf{G}^C}(s_1, \theta, 1)$ are the SWNE values of the bimatrix game:

$$\mathsf{Z}_1 \;=\; \bot \overset{\begin{matrix} c & s \end{matrix}}{\begin{pmatrix} 0 & \frac{3}{4} \end{pmatrix}} \quad \text{and} \quad \mathsf{Z}_2 \;=\; \bot \overset{\begin{matrix} c & s \end{matrix}}{\begin{pmatrix} 0 & \frac{1}{4} \end{pmatrix}}$$

  i.e. the values $(\frac{3}{4}, \frac{1}{4})$.

- In the second iteration $\mathtt{V}_{\mathsf{G}^C}(s_0, \theta, 2)$ are the SWNE values of the bimatrix game:

$$\mathsf{Z}_1 \;=\; \begin{matrix} \\ c \\ s \end{matrix} \begin{pmatrix} \perp \\ \frac{3}{4} \\ \frac{1}{4} \end{pmatrix} \quad \text{and} \quad \mathsf{Z}_2 \;=\; \begin{matrix} \\ c \\ s \end{matrix} \begin{pmatrix} \perp \\ \frac{1}{4} \\ \frac{3}{4} \end{pmatrix}$$

  i.e. the values $(\frac{3}{4}, \frac{1}{4})$, and $\mathtt{V}_{\mathsf{G}^C}(s_1, \theta, 2)$ are the SWNE values of the bimatrix games:

$$\mathsf{Z}_1 \;=\; \perp \begin{pmatrix} c & s \\ \frac{1}{4} & \frac{3}{4} \end{pmatrix} \quad \text{and} \quad \mathsf{Z}_2 \;=\; \perp \begin{pmatrix} c & s \\ \frac{3}{4} & \frac{1}{4} \end{pmatrix}$$

  i.e. the values $(\frac{1}{4}, \frac{3}{4})$.

- In the third iteration $\mathtt{V}_{\mathsf{G}^C}(s_0, \theta, 3)$ are the SWNE values of the bimatrix game:

$$\mathsf{Z}_1 \;=\; \begin{matrix} \\ c \\ s \end{matrix} \begin{pmatrix} \perp \\ \frac{1}{4} \\ \frac{1}{4} \end{pmatrix} \quad \text{and} \quad \mathsf{Z}_2 \;=\; \begin{matrix} \\ c \\ s \end{matrix} \begin{pmatrix} \perp \\ \frac{3}{4} \\ \frac{3}{4} \end{pmatrix}$$

  i.e. the values $(\frac{1}{4}, \frac{3}{4})$, and $\mathtt{V}_{\mathsf{G}^C}(s_1, \theta, 3)$ are the SWNE values of the bimatrix game:

$$\mathsf{Z}_1 \;=\; \perp \begin{pmatrix} c & s \\ \frac{3}{4} & \frac{3}{4} \end{pmatrix} \quad \text{and} \quad \mathsf{Z}_2 \;=\; \perp \begin{pmatrix} c & s \\ \frac{1}{4} & \frac{1}{4} \end{pmatrix}$$

  i.e. the values $(\frac{3}{4}, \frac{1}{4})$.

- In the fourth iteration $\mathtt{V}_{\mathsf{G}^C}(s_0, \theta, 4)$ are the SWNE values of the bimatrix game:

$$\mathsf{Z}_1 \;=\; \begin{matrix} \\ c \\ s \end{matrix} \begin{pmatrix} \perp \\ \frac{3}{4} \\ \frac{1}{4} \end{pmatrix} \quad \text{and} \quad \mathsf{Z}_2 \;=\; \begin{matrix} \\ c \\ s \end{matrix} \begin{pmatrix} \perp \\ \frac{1}{4} \\ \frac{3}{4} \end{pmatrix}$$

  i.e. the values $(\frac{3}{4}, \frac{1}{4})$, and $\mathtt{V}_{\mathsf{G}^C}(s_1, \theta, 4)$ are the SWNE values of the bimatrix game:

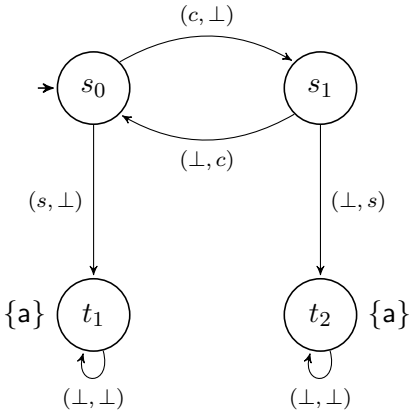
$$\mathsf{Z}_1 \;=\; \perp \begin{pmatrix} c & s \\ \frac{3}{4} & \frac{3}{4} \end{pmatrix} \quad \text{and} \quad \mathsf{Z}_2 \;=\; \perp \begin{pmatrix} c & s \\ \frac{1}{4} & \frac{1}{4} \end{pmatrix}$$

  i.e. the values $(\frac{3}{4}, \frac{1}{4})$.

As can be seen the values computed at each iteration for the states s_0 and s_1 will oscillate between $(\frac{1}{4}, \frac{3}{4})$ and $(\frac{3}{4}, \frac{1}{4})$.

C.4 Convergence of Nonzero-sum Expected Reachability Properties

In this section, we give a witness to the failure of convergence for value iteration when verifying nonzero-sum formulae with infinite horizon reward objectives if Assumption 4 does not hold.



$$r_A^1(s,(a_1,a_2)) = \begin{cases} \frac{1}{3} & \text{if } s = s_0 \text{ and } (a_1,a_2) = (s,\perp) \\ 2 & \text{if } s = s_1 \text{ and } (a_1,a_2) = (\perp,s) \\ 0 & \text{otherwise} \end{cases}$$

$$r_A^2(s,(a_1,a_2)) = \begin{cases} 1 & \text{if } s = s_0 \text{ and } (a_1,a_2) = (s,\perp) \\ \frac{1}{3} & \text{if } s = s_1 \text{ and } (a_1,a_2) = (\perp,s) \\ 0 & \text{otherwise} \end{cases}$$

Figure C.4: Counterexample for nonzero-sum expected reachability properties.

Consider the CSG in Figure C.4 with players p_1 and p_2 (which again is an adaptation of a TSG example from [30]) and the nonzero-sum state formula $\langle\langle p_1{:}p_2 \rangle\rangle_{\max=?}(\theta)$, where $\theta = \mathtt{R}^{r_1}[\,\mathtt{F}\ \mathtt{a}\,]{+}\mathtt{R}^{r_2}[\,\mathtt{F}\ \mathtt{a}\,]$ and $\mathtt{a}$ is the atomic proposition satisfied only by the states t_1 and t_2. In the CSG, we have that $r_S^1(s) = r_S^2(s) = 0$ for all s. Clearly, there are strategy profiles for which the targets are not reached with probability 1.

Applying the value iteration algorithm of Section 5.3.2, we have:

- In the first iteration $\mathtt{V}_{\mathsf{G}^C}(s_0, \theta, 1)$ are the SWNE values of the bimatrix game:

$$\mathsf{Z}_1 = \begin{matrix} \\ c \\ s \end{matrix} \begin{matrix} \perp \\ \begin{pmatrix} 0 \\ \frac{1}{3} \end{pmatrix} \end{matrix} \quad \text{and} \quad \mathsf{Z}_2 = \begin{matrix} \\ c \\ s \end{matrix} \begin{matrix} \perp \\ \begin{pmatrix} 0 \\ 1 \end{pmatrix} \end{matrix}$$

  i.e. the values $(\frac{1}{3}, 1)$, and $\mathtt{V}_{\mathsf{G}^C}(s_1, \theta, 1)$ are the SWNE values of the bimatrix game:

$$\mathsf{Z}_1 = \begin{matrix} \\ \perp \end{matrix} \begin{matrix} \begin{matrix} c & s \end{matrix} \\ \begin{pmatrix} 0 & 2 \end{pmatrix} \end{matrix} \quad \text{and} \quad \mathsf{Z}_2 = \begin{matrix} \\ \perp \end{matrix} \begin{matrix} \begin{matrix} c & s \end{matrix} \\ \begin{pmatrix} 0 & \frac{1}{3} \end{pmatrix} \end{matrix}$$

  i.e. the values $(2, \frac{1}{3})$.

- In the second iteration $\mathtt{V}_{\mathsf{G}^C}(s_0, \theta, 2)$ are the SWNE values of the bimatrix game:

$$\mathsf{Z}_1 \;=\; \begin{matrix} \\ c \\ s \end{matrix} \overset{\perp}{\begin{pmatrix} 2 \\ \frac{1}{3} \end{pmatrix}} \quad \text{and} \quad \mathsf{Z}_2 \;=\; \begin{matrix} \\ c \\ s \end{matrix} \overset{\perp}{\begin{pmatrix} \frac{1}{3} \\ 1 \end{pmatrix}}$$

  i.e. the values $(2, \frac{1}{3})$, and $\mathtt{V}_{\mathsf{G}^C}(s_1, \theta, 2)$ are the SWNE values of the bimatrix games:

$$\mathsf{Z}_1 \;=\; \perp \overset{c \quad s}{\begin{pmatrix} \frac{1}{3} & 2 \end{pmatrix}} \quad \text{and} \quad \mathsf{Z}_2 \;=\; \perp \overset{c \quad s}{\begin{pmatrix} 1 & \frac{1}{3} \end{pmatrix}}$$

  i.e. the values $(\frac{1}{3}, 1)$.

- In the third iteration $\mathtt{V}_{\mathsf{G}^C}(s_0, \theta, 3)$ are the SWNE values of the bimatrix game:

$$\mathsf{Z}_1 \;=\; \begin{matrix} \\ c \\ s \end{matrix} \overset{\perp}{\begin{pmatrix} \frac{1}{3} \\ \frac{1}{3} \end{pmatrix}} \quad \text{and} \quad \mathsf{Z}_2 \;=\; \begin{matrix} \\ c \\ s \end{matrix} \overset{\perp}{\begin{pmatrix} 1 \\ 1 \end{pmatrix}}$$

  i.e. the values $(\frac{1}{3}, 1)$, and $\mathtt{V}_{\mathsf{G}^C}(s_1, \theta, 3)$ are the SWNE values of the bimatrix game:

$$\mathsf{Z}_1 \;=\; \perp \overset{c \quad s}{\begin{pmatrix} 2 & 2 \end{pmatrix}} \quad \text{and} \quad \mathsf{Z}_2 \;=\; \perp \overset{c \quad s}{\begin{pmatrix} \frac{1}{3} & \frac{1}{3} \end{pmatrix}}$$

  i.e. the values $(2, \frac{1}{3})$.

- In the fourth iteration $\mathtt{V}_{\mathsf{G}^C}(s_0, \theta, 4)$ are the SWNE values of the bimatrix game:

$$\mathsf{Z}_1 \;=\; \begin{matrix} \\ c \\ s \end{matrix} \overset{\perp}{\begin{pmatrix} 2 \\ \frac{1}{3} \end{pmatrix}} \quad \text{and} \quad \mathsf{Z}_2 \;=\; \begin{matrix} \\ c \\ s \end{matrix} \overset{\perp}{\begin{pmatrix} \frac{1}{3} \\ 1 \end{pmatrix}}$$

  i.e. the values $(2, \frac{1}{3})$, and $\mathtt{V}_{\mathsf{G}^C}(s_1, \theta, 4)$ are the SWNE values of the bimatrix game:

$$\mathsf{Z}_1 \;=\; \perp \overset{c \quad s}{\begin{pmatrix} \frac{1}{3} & 2 \end{pmatrix}} \quad \text{and} \quad \mathsf{Z}_2 \;=\; \perp \overset{c \quad s}{\begin{pmatrix} 1 & \frac{1}{3} \end{pmatrix}}$$

  i.e. the values $(\frac{1}{3}, 1)$.

As can be seen the values computed during value iteration oscillate for both s_0 and s_1.

Bibliography

[1] K. J. Åström. Optimal control of Markov processes with incomplete state information i. 10:174–205, 1965.

[2] R. Alur, T. Henzinger, F. Mang, S. Qadeer, S. Rajamani, and S. Tasiran. MOCHA: Modularity in model checking. In *Proc. CAV '98*, volume 1427 of *LNCS*, pages 521–525. Springer, 1998.

[3] R. Alur and T. A. Henzinger. Reactive modules. *FMSD*, 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] B. Aminof, M. Kwiatkowska, B. Maubert, A. Murano, and S. Rubin. Probabilistic strategy logic. In *Proc. IJCAI'19*, pages 32–38. International Joint Conferences on Artificial Intelligence Organization, 2019.

[6] B. An, E. Shieh, M. Tambe, R. Yang, C. Baldwin, J. DiRenzo, B. Maule, and G. Meyer. PROTECT – a deployed game theoretic system for strategic security allocation for the united states coast guard. *AI Magazine*, 33(4):96, 2012.

[7] P. Ashok, K. Chatterjee, J. Kretínský, M. Weininger, and T. Winkler. Approximating values of generalized-reachability stochastic games. In *Proc. LICS'20*, pages 102–115. ACM, 2020.

[8] R. Aumann. Subjectivity and correlation in randomized strategies. *Journal of Mathematical Economics*, 1(1):67–96, 1974.

[9] D. Avis, G. Rosenberg, R. Savani, and B. von Stengel. Enumeration of Nash equilibria for two-player games. *Economic Theory*, 42(1):9–37, 2010.

[10] C. Baier, T. Brázdil, M. Größer, and A. Kučera. Stochastic game logic. *Acta Informatica*, 49(4):203–224, 2012.

[11] C. Baier and F. Ciesinski. LiQuor: A tool for qualitative and quantitative linear time analysis of reactive systems. In *Proc. QEST'06*, pages 131–132. IEEE CS Press, 2006.

[12] C. Baier, F. Ciesinski, and M. Grosser. PROBMELA: a modeling language for communicating probabilistic processes. In *Proc. MEMOCODE'04*, pages 57–66. IEEE Computer Society, 2004.

[13] C. Baier and J-P. Katoen. *Principles of Model Checking*. The MIT Press, 2008.

[14] C. Baier, J. Klein, L. Leuschner, D. Parker, and S. Wunderlich. Ensuring the reliability of your model checker: Interval iteration for Markov decision processes. In *Proc. CAV'17*, volume 10426 of *LNCS*, pages 160–180. Springer, 2017.

[15] P. Ballarini, B.Barbot, M. Duflot, S. Haddad, and N. Pekergin. HASL: A new approach for performance evaluation and model checking from concepts to experimentation. *Performance Evaluation*, 90:53–77, 2015.

[16] N. Basset, M. Kwiatkowska, U. Topcu, and C. Wiltsche. Strategy synthesis for stochastic games with multiple long-run objectives. In *Proc. TACAS'15*, volume 9035 of *LNCS*, pages 256–271. Springer, 2015.

[17] R. Bellman. A Markovian decision process. *Indiana University Mathematics Journal*, 6:679–684, 1957.

[18] C. Belta, A. Bicchi, M. Egerstedt, E. Frazzoli, E. Klavins, and G. J. Pappas. Symbolic planning and control of robot motion. *IEEE Robotics Automation Magazine*, 14(1):61–70, 2007.

[19] A. Bianco and L. de Alfaro. Model checking of probabilistic and nondeterministic systems. In *Proc. FSTTCS'95*, volume 1026 of *LNCS*, pages 499–513. Springer, 1995.

[20] A. Biere, A. Cimatti, E. Clarke, M. Fujita, and Y. Zhu. Symbolic model checking using SAT procedures instead of bdds. In *Proc. DAC'99*, pages 317–320. Association for Computing Machinery, 1999.

[21] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu. Symbolic model checking without BDDs. In *Proc. TACAS'99*, pages 193–207. Springer Berlin Heidelberg, 1999.

[22] A. Biere, M. Heule, H. van Maaren, and T. Walsh. *Handbook of Satisfiability: Volume 185 Frontiers in Artificial Intelligence and Applications*. IOS Press, 2009.

[23] P. Billingsley. *Probability and Measure*. Wiley Series in Probability and Statistics. Wiley, 1995.

[24] R. Bloem, K. Chatterjee, T. Henzinger, and B. Jobstmann. Better quality in synthesis through quantitative objectives. In *Computer Aided Verification*, pages 140–156. Springer Berlin Heidelberg, 2009.

[25] B. Blum, C. Shelton, and D. Koller. A continuation method for Nash equilibria in structured games. *J. Artif. Intell. Res. (JAIR)*, 25:457–502, 2006.

[26] R. H. Bordini, M. Fisher, C. Pardavila, and M. Wooldridge. Model checking Agentspeak. In *Proc. AAMAS'03*, pages 409–416. Association for Computing Machinery, 2003.

[27] R. H. Bordini, M. Fisher, W. Visser, and M. Wooldridge. Verifying multi-agent programs by model checking. *Autonomous Agents and Multi-Agent Systems*, 12(2):239–256, 2006.

[28] É. Borel. La théorie du jeu et les équations intégrales à nouyau symétrique. *Comptes Rendus hebdomadaires des séances de l'Académie des Sciences*, 173:1304–1308, 1921.

[29] P. Bouyer, N. Markey, and D. Stan. Mixed Nash equilibria in concurrent games. In *Proc. FSTTCS'14*, volume 29 of *LIPICS*, pages 351–363. Leibniz-Zentrum für Informatik, 2014.

[30] P. Bouyer, N. Markey, and D. Stan. Stochastic equilibria under imprecise deviations in terminal-reward concurrent games. In *Proc. GandALF'16*, volume 226 of *EPTCS*, pages 61–75. Open Publishing Association, 2016.

[31] T. Brázdil, V. Brožek, V. Forejt, and A. Kučera. Stochastic games with branching-time winning objectives. In *Proc. LICS'06*, pages 349–358, 2006.

[32] T. Brázdil, K. Chatterjee, M. Chmelík, V. Forejt, J. Křetínský, M. Kwiatkowska, D. Parker, and M. Ujma. Verification of Markov decision processes using learning algorithms. In *Proc. ATVA'14*, volume 8837, pages 98–114. Springer, 2014.

[33] R. Brenguier. PRALINE: A tool for computing Nash equilibria in concurrent games. In *Proc. CAV'13*, volume 8044 of *LNCS*, pages 890–895. Springer, 2013.

[34] T. Brihaye, V. Bruyère, A. Goeminne, and J-F. Raskin. Constrained existence problem for weak subgame perfect equilibria with ω-regular boolean objectives. In *Proc. GandALF'18*, volume 277 of *EPTCS*, pages 16–29, 2018.

[35] T. Brihaye, V. Bruyère, A. Goeminne, J.-F. Raskin, and M. van den Bogaard. The complexity of subgame perfect equilibria in quantitative reachability games. In *Proc. CONCUR'19*, volume 140 of *LIPIcs*, pages 13:1–13:16. Leibniz-Zentrum für Informatik, 2019.

[36] R. Bryant. Chain reduction for binary and zero-suppressed decision diagrams. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 81–98. Springer International Publishing, 2018.

[37] J. Büchi and L. Landweber. Solving sequential conditions by finite-state strategies. *Transactions of the American Mathematical Society*, 138:295–311, 1969.

[38] C. E. Budde, C. Dehnert, E. M. Hahn, A. Hartmanns, S. Junges, and A. Turrini. JANI: quantitative model and tool interaction. In *Proc. TACAS'17*, volume 10206 of *LNCS*, pages 151–168, 2017.

[39] N. Bulling and W. Jamroga. What agents can probably enforce. *Fundamenta Informaticae*, 93(1–3):81–96, 2009.

[40] P. Čermák, A. Lomuscio, F. Mogavero, and A. Murano. MCMAS-SLK: A model checker for the verification of strategy logic specifications. In *Proc. CAV'14*, volume 8559 of *LNCS*, pages 525–532. Springer, 2014.

[41] K. Chatterjee. Two-player nonzero-sum ω-regular games. In *Proc. CONCUR'05*, pages 413–427. Springer Berlin Heidelberg, 2005.

[42] K. Chatterjee. Nash equilibrium for upward-closed objectives. In *Proc. CSL'06*, volume 4027 of *LNCS*, pages 271–286. Springer, 2006.

[43] K. Chatterjee. *Stochastic ω-Regular Games*. PhD thesis, University of California at Berkeley, 2007.

[44] K. Chatterjee, L. de Alfaro, and T. Henzinger. Strategy improvement for concurrent reachability and turn-based stochastic safety games. *Journal of Computer and System Sciences*, 79(5):640–657, 2013.

[45] K. Chatterjee and L. Doyen. Partial-observation stochastic games: How to win when belief fails. *ACM Trans. Comput. Logic*, 15(2), 2014.

[46] K. Chatterjee, L. Doyen, and T. Henzinger. A survey of partial-observation stochastic parity games. *Formal Methods in System Design*, 43(2):268–284, 2013.

[47] K. Chatterjee and T. Henzinger. Value iteration. In *25 Years of Model Checking*, volume 5000 of *LNCS*, pages 107–138. Springer, 2008.

[48] K. Chatterjee and T. Henzinger. A survey of stochastic ω-regular games. *Journal of Computer and System Sciences*, 78(2):394–413, 2012.

[49] K. Chatterjee, T. Henzinger, B. Jobstmann, and A. Radhakrishna. GIST: A solver for probabilistic games. In *Proc. CAV'10*, volume 6174 of *LNCS*, pages 665–669. Springer, 2010.

[50] K. Chatterjee, T. Henzinger, and N. Piterman. Strategy logic. In *Proc. CONCUR'07*, pages 59–73. Springer Berlin Heidelberg, 2007.

[51] K. Chatterjee, M. Jurdziński, and T. Henzinger. Simple stochastic parity games. In *Computer Science Logic*, pages 100–113. Springer Berlin Heidelberg, 2003.

[52] K. Chatterjee, R. Majumdar, and M. Jurdziński. On Nash equilibria in stochastic games. In *Proc. CSL'04*, volume 3210 of *LNCS*, pages 26–40. Springer, 2004.

[53] T. Chen, V. Forejt, M. Kwiatkowska, D. Parker, and A. Simaitis. Automatic verification of competitive stochastic systems. *Formal Methods in System Design*, 43(1):61–92, 2013.

[54] T. Chen, V. Forejt, M. Kwiatkowska, A. Simaitis, and C. Wiltsche. On stochastic games with multiple objectives. In *Proc. MFCS'13*, volume 8087 of *LNCS*, pages 266–277. Springer, 2013.

[55] T. Chen, M. Kwiatkowska, A. Simaitis, and C. Wiltsche. Synthesis for multi-objective stochastic games: An application to autonomous urban driving. In *Proc. QEST'13*, volume 8054 of *LNCS*, pages 322–337. Springer, 2013.

[56] T. Chen and J. Lu. Probabilistic alternating-time temporal logic and model checking algorithm. In *Proc. FSKD'07*, pages 35–39. IEEE Computer Society, 2007.

[57] X. Chen, X. Deng, and S-H. Teng. Settling the complexity of computing two-player Nash equilibria. *J. ACM*, 56(3), 2009.

[58] C. Cheng, A. Knoll, M. Luttenberger, and C. Buckl. GAVS+: An open platform for the research of algorithmic game solving. In *Proc. TACAS'11*, volume 6605 of *LNCS*, pages 258–261. Springer, 2011.

[59] A. Church. Application of recursive arithmetic to the problem of circuit synthesis. *Journal of Symbolic Logic*, 28(4):289–290, 1963.

[60] E. Clarke and E. Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In *Logic of Programs, Workshop*, pages 52–71. Springer-Verlag, 1982.

[61] J. L. Cohon. *Multiobjective Programming and Planning*. Academic Press, New York, 1978.

[62] A. Condon. *Computational Models of Games*. PhD thesis, University of Washington, 1987.

[63] A. Condon. The complexity of stochastic games. *Information and Computation*, 96(2):203–224, 1992.

[64] R. W. Cottle and G. B. Dantzig. Complementary pivot theory of mathematical programming. *Linear Algebra and its Applications*, 1(1):103–125, 1968.

[65] C. Courcoubetis and M. Yannakakis. Verifying temporal properties of finite state probabilistic programs. In *Proc. FOCS'88*, pages 338–345. IEEE Computer Society Press, 1988.

[66] A. Czumaj, A. Deligkas, M. Fasoulakis, J. Fearnley, M. Jurdziński, and R. Savani. Distributed methods for computing approximate equilibria. *Algorithmica*, 81(3):1205–1231, 2019.

[67] A. Czumaj, M. Fasoulakis, and M. Jurdziński. Approximate plutocratic and egalitarian Nash equilibria: (extended abstract). In *Proc. AAMAS'16*, pages 1409–1410. ACM, 2016.

[68] A. Czumaj, M. Fasoulakis, and M. Jurdziński. Multi-player approximate Nash equilibria. In *Proc. AAMAS'17*, pages 1511–1513. ACM, 2017.

[69] A. Czumaj, M. Fasoulakis, and M. Jurdziński. Zero-sum game techniques for approximate Nash equilibria. In *Proc. AAMAS'17*, pages 1514–1516. ACM, 2017.

[70] G. B. Dantzig. *Linear Programming and Extensions*. Princeton University Press, 1965.

[71] C. Daskalakis, P. Goldberg, and C. Papadimitriou. The complexity of computing a Nash equilibrium. *Communications of the ACM*, 52(2):89–97, 2009.

[72] L. de Alfaro. *Formal Verification of Probabilistic Systems*. PhD thesis, Stanford University, 1997.

[73] L. de Alfaro. Computing minimum and maximum reachability times in probabilistic systems. In *Proc. CONCUR'99*, volume 1664 of *LNCS*, pages 66–81. Springer, 1999.

[74] L. de Alfaro and T. Henzinger. Concurrent omega-regular games. In *LICS'00*, pages 141–154. ACM, 2000.

[75] L. de Alfaro, T. Henzinger, and O. Kupferman. Concurrent reachability games. *Theoretical Computer Science*, 386(3):188–217, 2007.

[76] L. de Alfaro and R. Majumdar. Quantitative solution of omega-regular games. *Journal of Computer and System Sciences*, 68(2):374–397, 2004.

[77] Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient SMT solver. In *Proc. TACAS'08*, volume 4963 of *LNCS*, pages 337–340. Springer, 2008. github.com/Z3Prover/z3.

[78] C. Dehnert, S. Junges, N. Jansen, F. Corzilius, M. Volk, H. Bruintjes, J-P. Katoen, and E. Ábrahám. PROPhESY: A PRObabilistic ParamEter SYnthesis tool. In *Proc. CAV'15*, pages 214–231. Springer International Publishing, 2015.

[79] B. Dutertre. Yices 2.2. In *Proc CAV'14*, volume 8559 of *LNCS*, pages 737–744. Springer, 2014.

[80] B. C. Eaves. The linear complementarity problem. *Management Science*, 17(9):612–634, 1971.

[81] Y. Elderhalli, O. Hasan, W. Ahmad, and Sofiène S. Tahar. Formal dynamic fault trees analysis using an integration of theorem proving and model checking. In *Proc. NFM'18*, pages 139–156. Springer International Publishing, 2018.

[82] K. Etessami and M. Yannakakis. Recursive concurrent stochastic games. In *Proc. ICALP '06*, volume 4052 of *LNCS*, pages 324–335. Springer, 2006.

[83] K. Etessami and M. Yannakakis. On the complexity of Nash equilibria and other fixed points (extended abstract). In *Proc. FOCS'07*, pages 113–123. IEEE Computer Society, 2007.

[84] J. Fearnley and R. Savani. The complexity of the simplex method. In *Proc. STOC'15*, pages 201–208. ACM, 2015.

[85] D. Fernando, N. Dong, C. Jegourel, and J. S. Dong. Verification of strong Nash-equilibrium for probabilistic BAR systems. In *Formal Methods and Software Engineering*, pages 106–123. Springer International Publishing, 2018.

[86] J. Filar and K. Vrieze. *Competitive Markov Decision Processes*. Springer, 1996.

[87] V. Forejt, M. Kwiatkowska, G. Norman, D. Parker, and H. Qu. Quantitative multi-objective verification for probabilistic systems. In *Proc. TACAS'11*, volume 6605 of *LNCS*, pages 112–127. Springer, 2011.

[88] D. Fremont, M. Rabe, and S. Seshia. Maximum model counting. Technical Report UCB/EECS-2016-169, EECS Department, University of California, Berkeley, 2016.

[89] D. Fremont, M. N. Rabe, and S. A. Seshia. Maximum model counting. In *Proc. AAAI'17*, pages 3885–3892, 2017.

[90] D. Fudenberg and D. Levine. Subgame-perfect equilibria of finite- and infinite-horizon games. *Journal of Economic Theory*, 31(2):251–268, 1983.

[91] M. Fujita, P. McGeer, and J. Yang. Multi-terminal binary decision diagrams: An efficient data structure for matrix representation. *Formal Methods in System Design*, 10(2):149–169, 1997.

[92] Gerald G. Brown, M. Carlyle, J. Salmerón, and R. Wood. Defending critical infrastructure. *Interfaces*, 36:530–544, 2006.

[93] E. Gansner and S. North. An open graph visualization system and its applications to software engineering. *Software - Practice and Experience*, 30(11):1203–1233, 2000.

[94] H. Garavel, F. Lang, R. Mateescu, and W. Serwe. CADP 2011: a toolbox for the construction and analysis of distributed processes. *International Journal on Software Tools for Technology Transfer*, 15(2):89–107, 2013.

[95] I. Gilboa and E. Zemel. Nash and correlated equilibria: Some complexity considerations. *Games and Economic Behavior*, 1(1):80–93, 1989.

[96] H. Gimbert and F. Horn. Solving simple stochastic games. In *Logic and Theory of Algorithms*, pages 206–209. Springer Berlin Heidelberg, 2008.

[97] S. Govindan and R. Wilson. A global Newton method to compute Nash equilibria. *Journal of Economic Theory*, 110(1):65–86, 2003.

[98] S. Govindan and R. Wilson. Computing Nash equilibria by iterated polymatrix approximation. *Journal of Economic Dynamics and Control*, 28(7):1229 – 1241, 2004.

[99] S. Govindan and R. Wilson. Computing equilibria of n-player games with arbitrary accuracy. *Stanford University, Graduate School of Business, Research Papers*, 2008.

[100] F. Gretz, N. Jansen, B. Kaminski, J-P. Katoen, A. McIver, and F. Olmedo. Conditioning in probabilistic programming. *Electronic Notes in Theoretical Computer Science*, 319, 2015.

[101] S. Haddad and B. Monmege. Interval iteration algorithm for MDPs and IMDPs. *Theoretical Computer Science*, 735:111–131, 2018.

[102] E. Hahn, H. Hermanns, B. Wachter, and L. Zhang. PARAM: A model checker for parametric Markov models. In *Proc. CAV'10*, volume 6174 of *LNCS*, pages 660–664. Springer, 2010.

[103] E. Hahn, Y. Li, S. Schewe, A. Turrini, and L. Zhang. IscasMC: A web-based probabilistic model checker. In *Proc. FM'14*, volume 8442 of *LNCS*, pages 312–317. Springer, 2014.

[104] J. Halpern and V. Teague. Rational secret sharing and multiparty computation: Extended abstract. In *Proc. STOC'04*, pages 623–632. ACM, 2004.

[105] Z. Han, D. Niyato, W. Saad, T. Başar, and A. Hjørungnes. *Game Theory in Wireless and Communication Networks: Theory, Models, and Applications*. Cambridge University Press, 2011.

[106] Z. Han, D. Niyato, W. Saad, T. Başar, and A. Hjørungnes. *Game Theory for Next Generation Wireless and Communication Networks: Modeling, Analysis, and Design*. Cambridge University Press, 2019.

[107] H. Hansson and B. Jonsson. A logic for reasoning about time and reliability. *Formal Aspects of Computing*, 6(5):512–535, 1994.

[108] J. Harsanyi. Games with incomplete information played by 'bayesian' players, I-III. volume 14, pages 159–182, 1967.

[109] J. C. Harsanyi and R. Selten. *A General Theory of Equilibrium Selection in Games*. MIT Press Classics. IMW, 1988.

[110] A. Hartmanns and H. Hermanns. The Modest toolset: An integrated environment for quantitative modelling and verification. In *Proc. TACAS'14*, volume 8413 of *LNCS*, pages 593–598. Springer, 2014.

[111] O. Hauser, C. Hilbe, K. Chatterjee, and M. Nowak. Social dilemmas among unequals. *Nature*, 572:524–527, 2019.

[112] C. Hensel, S. Junges, J-P. Katoen, T. Quatmann, and M. Volk. The probabilistic model checker Storm, 2020.

[113] J. Hillston. *PEPA: Performance enchaced process algebra*. Phd thesis, University of Edinburgh, 1993.

[114] G. J. Holzmann. The model checker SPIN. *IEEE Transactions on Software Engineering*, 23(5):279–295, 1997.

[115] Gerard J. Holzmann. Design and validation of protocols: a tutorial. *Computer Networks and ISDN Systems*, 25(9):981–1017, 1993.

[116] K. Horák, B. Bošanský, and M. Péchouček. Heuristic search value iteration for one-sided partially observable stochastic games. In *Proc. AAAI'17*, pages 558–564. AAAI Press, 2017.

[117] The HSL Mathematical Software Library. www.hsl.rl.ac.uk/.

[118] Julian J. Gutierrez, M. Najib, G. Perelli, and M. Wooldridge. Eve: A tool for temporal equilibrium analysis. In *Proc. ATVA'18*, volume 11138 of *LNCS*, pages 551–557. Springer, 2018.

[119] N. Jansen, E. Ábrahám, M. Volk, R. Wimmer, J-P. Katoen, and B. Becker. The COMICS tool – computing minimal counterexamples for DTMCs. In *Proc. ATVA'12*, pages 349–353. Springer Berlin Heidelberg, 2012.

[120] C. Jegourel, A. Legay, and S. Sedwards. A platform for high performance statistical model checking – PLASMA. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 498–503. Springer Berlin Heidelberg, 2012.

[121] S. Junges, N. Jansen, J-P. Katoen, U. Topcu, R. Zhang, and M. Hayhoe. Model checking for safe navigation among humans. In *Proc. QEST'18*, pages 207–222. Springer International Publishing, 2018.

[122] T. Brázdil K., Chatterjee, V. Forejt, and A. Kučera. Multigain: A controller synthesis tool for MDPs with multiple mean-payoff objectives. In *Proc. TACAS'15*, pages 181–187. Springer, 2015.

[123] L. Kaelbling, M. Littman, and A. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1):99–134, 1998.

[124] E. Kalai. Games, computers, and O.R. In *Proc. SODA'96*, pages 468–473. Society for Industrial and Applied Mathematics, 1996.

[125] N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4(4):373–395, 1984.

[126] J-P Katoen, I. Zapreev, E. Hahn, H. Hermanns, and D. Jansen. The ins and outs of the probabilistic model checker MRMC. *Performance Evaluation*, 68(2):90–104, 2011. Advances in Quantitative Evaluation of Systems.

[127] E. Kelmendi, J. Krämer, J. Kretínský, and M. Weininger. Value iteration for simple stochastic games: Stopping criterion and learning algorithm. In *Proc. CAV'18*, volume 10981 of *LNCS*, pages 623–642. Springer, 2018.

[128] J. Kemeny, J. Snell, and A. Knapp. *Denumerable Markov Chains*. Springer, 1976.

[129] V. Knight and J. Campbell. Nashpy: A python library for the computation of Nash equilibria. *Journal of Open Source Software*, 3(30):904, 2018.

[130] Donald E. Knuth. *The Art of Computer Programming, Volume 4, Fascicle 1: Bitwise Tricks & Techniques; Binary Decision Diagrams*. Addison-Wesley Professional, 12th edition, 2009.

[131] D. Koller, N. Megiddo, and B. von Stengel. Efficient computation of equilibria for extensive two-person games. *Games and Economic Behavior*, 14(2):247–259, 1996.

[132] H. Kress-Gazit, G. Fainekos, and G. Pappas. Where's Waldo? Sensor-based temporal logic motion planning. In *Proc. of IEEE International Conference on Robotics and Automation*, pages 3116–3121, 2007.

[133] H. W. Kuhn. Extensive games. *Proceedings of the National Academy of Sciences*, 36(10):570–576, 1950.

[134] M. Kwiatkowska, G. Norman, and D. Parker. Stochastic model checking. In *Formal Methods for the Design of Computer, Communication and Software Systems: Performance Evaluation (SFM'07)*, volume 4486 of *LNCS (Tutorial Volume)*, pages 220–270. Springer, 2007.

[135] M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *Proc CAV'11*, volume 6806 of *LNCS*, pages 585–591. Springer, 2011.

[136] M. Kwiatkowska, G. Norman, D. Parker, and G. Santos. Automated verification of concurrent stochastic games. In *Proc. QEST'18*, volume 11024 of *LNCS*, pages 223–239. Springer, 2018.

[137] M. Kwiatkowska, G. Norman, D. Parker, and G. Santos. Equilibria-based probabilistic model checking for concurrent stochastic games. In *Proc. FM'19*, LNCS. Springer, 2019.

[138] M. Kwiatkowska, G. Norman, D. Parker, and G. Santos. Multi-player equilibria verification for concurrent stochastic games. In *Proc. QEST'20*, LNCS. Springer, 2020.

[139] M. Kwiatkowska, G. Norman, D. Parker, and G. Santos. Prism-games 3.0: Stochastic game verification with concurrency, equilibria and time. In *Proc. CAV'20*, LNCS. Springer, 2020.

[140] M. Kwiatkowska, G. Norman, D. Parker, and G. Santos. Automatic verification of concurrent stochastic systems. *Formal Methods in System Design*, 2021.

[141] M. Kwiatkowska and D. Parker. Automated verification and strategy synthesis for probabilistic systems. In *Proc. ATVA'13*, volume 8172 of *LNCS*, pages 5–22. Springer, 2013.

[142] M. Kwiatkowska, D. Parker, and A. Simaitis. Strategic analysis of trust models for user-centric networks. In *Proc. SR'13*, volume 112 of *EPTCS*, pages 53–60. Open Publishing Association, 2013.

[143] M. Kwiatkowska, D. Parker, and C. Wiltsche. PRISM-games 2.0: A tool for multi-objective strategy synthesis for stochastic games. In *Proc. TACAS'16*, volume 9636 of *LNCS*. Springer, 2016.

[144] M. Kwiatkowska, D. Parker, and C. Wiltsche. PRISM-games: Verification and strategy synthesis for stochastic multi-player games with multiple objectives. *Software Tools for Technology Transfer*, 20(2):195–210, 2018.

[145] C. Lemke and Jr J. Howson. Equilibrium points of bimatrix games. *Journal of the Society for Industrial and Applied Mathematics*, 12(2):413–423, 1964.

[146] M. Littman, N. Ravi, A. Talwar, and M. Zinkevich. An efficient optimal-equilibrium algorithm for two-player game trees. In *Proc. UAI'06*, pages 298–305. AUAI Press, 2006.

[147] A. Lomuscio, H. Qu, and F. Raimondi. MCMAS: an open-source model checker for the verification of multi-agent systems. *International Journal on Software Tools for Technology Transfer*, 19(1):9–30, 2017.

[148] LPSolve (version 5.5). lpsolve.sourceforge.net/5.5/.

[149] A. MacKenzie and S. Wicker. Stability of multipacket slotted Aloha with selfish users and perfect information. In *Proc. IEEE INFOCOM'03*, pages 1583–1590. IEEE Computer Society, 2003.

[150] O. L. Mangasarian. Equilibrium points of bimatrix games. *Journal of the Society for Industrial and Applied Mathematics*, 12(4):778–780, 1964.

[151] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems*. Springer-Verlag, 1992.

[152] F. Mari, I. Melatti, I. Salvo, E. Tronci, L. Alvisi, A. Clement, and H. Li. Model checking coalition Nash equilibria in MAD distributed systems. In *Stabilization, Safety, and Security of Distributed Systems*, pages 531–546. Springer Berlin Heidelberg, 2009.

[153] R. Marler and J. Arora. Survey of multi-objective optimization methods for engineering. *Structural and Multidisciplinary Optimization*, 26(6):369–395, 2004.

[154] M. Marsan, G. Conte, and G. Balbo. A class of generalized stochastic Petri nets for the performance evaluation of multiprocessor systems. *ACM Trans. Comput. Syst.*, 2(2):93–122, 1984.

[155] D. Martin. The determinacy of blackwell games. *The Journal of Symbolic Logic*, 63(4):1565–1581, 1998.

[156] A. McIver and C. Morgan. *Abstraction, Refinement And Proof For Probabilistic Systems (Monographs in Computer Science)*. Springer-Verlag, 2004.

[157] A. McIver and C. Morgan. Results on the quantitative mu-calculus qMu. *ACM Trans. Computational Logic*, 8(1), 2007.

[158] R. McKelvey. A Liapunov function for Nash equilibria. California Institute of Technology, 1998.

[159] R. McKelvey, A. McLennan, and T. Turocy. Gambit: Software tools for game theory, version 16.0.1. gambit-project.org, 2016.

[160] R. McKelvey and T. Palfrey. Quantal response equilibria for normal form games. *Games and Economic Behavior*, 10(1):6–38, 1995.

[161] K. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, 1993.

[162] A. W. Mostowski. Regular expressions for infinite trees and a standard form of automata. In *Computation Theory*, pages 157–168. Springer Berlin Heidelberg, 1985.

[163] Y. Narahari, R. Narayanam, D. Garg, and H. Prakash. Foundations of mechanism design. In *Game Theoretic Problems in Network Economics and Mechanism Design Solutions*, Advanced Information and Knowledge Processing, pages 1–131. Springer, 2009.

[164] J. Nash. Equilibrium points in n-person games. *Proc. Natl. Acad. Sci*, 36:48–49, 1950.

[165] N. Nisan and A. Ronen. Algorithmic mechanism design. *Games and Economic Behavior*, 35(1):166–196, 2001.

[166] N. Nisan, T. Roughgarden, E. Tardos, and V. Vazirani. *Algorithmic Game Theory*. Cambridge University Press, 2007.

[167] J. Nocedal, A. Wächter, and R. Waltz. Adaptive barrier update strategies for nonlinear interior methods. *SIAM Journal on Optimization*, 19(4):1674–1693, 2009.

[168] E. Nudelman, J. Wortman, Y. Shoham, and K. Leyton-Brown. Run the GAMUT: A comprehensive approach to evaluating game-theoretic algorithms. In *Proc. AAMAS'04*, pages 880–887. ACM, 2004. gamut.stanford.edu.

[169] M. Osborne and A. Rubinstein. *An Introduction to Game Theory*. Oxford University Press, 2004.

[170] N. Ozay, U. Topcu, R. Murray, and T. Wongpiromsarn. Distributed synthesis of control protocols for smart camera networks. In *2011 IEEE/ACM Second International Conference on Cyber-Physical Systems*, pages 45–54, 2011.

[171] C. Papadimitriou. On the complexity of the parity argument and other inefficient proofs of existence. *Journal of Computer and System Sciences*, 48(3):498–532, 1994.

[172] D. Parker. *Implementation of Symbolic Model Checking for Probabilistic Systems*. Phd thesis, University of Birmingham, 2002.

[173] D. Parkes and S. Singh. An MDP-based approach to online mechanism design. In *Proc. NIPS'03*, pages 791–798. MIT Press, 2003.

[174] R. Porter, E. Nudelman, and Y. Shoham. Simple search methods for finding a Nash equilibrium. In *Proc. AAAI'04*, pages 664–669. AAAI Press, 2004.

[175] M. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., 1st edition, 1994.

[176] J-P. Queille and J. Sifakis. Specification and verification of concurrent systems in CESAR. In *Proceedings of the 5th Colloquium on International Symposium on Programming*, pages 337–351. Springer-Verlag, 1982.

[177] T. Raghavan and J. Filar. Algorithms for stochastic games — a survey. *Zeitschrift für Operations Research*, 35(6):437–472, 1991.

[178] E. Ruijters and M. Stoelinga. Fault tree analysis: A survey of the state-of-the-art in modeling, analysis and tools. *Computer Science Review*, 15-16:29–62, 2015.

[179] T. Sandholm, A. Gilpin, and V. Conitzer. Mixed-integer programming methods for finding Nash equilibria. In *Proc. AAAI'05*, pages 495–501. AAAI Press, 2005.

[180] S. Schewe. Synthesis for probabilistic environments. In *Automated Technology for Verification and Analysis*, pages 245–259. Springer Berlin Heidelberg, 2006.

[181] U. Schwalbe and P. Walker. Zermelo and the early history of game theory. *Games and Economic Behavior*, 34(1):123–137, 2001.

[182] P. Secchi and W. Sudderth. Stay-in-a-set games. *International Journal of Game Theory*, 30(4):479–490, 2002.

[183] R. Segala. Modeling and verification of randomized distributed real-time systems. Technical report, Massachusetts Institute of Technology, 1996.

[184] R. Selten. Spieltheoretische behandlung eines oligopolmodells mit nachfrageträgheit: Teil i: Bestimmung des dynamischen preisgleichgewichts. *Zeitschrift für die gesamte Staatswissenschaft / Journal of Institutional and Theoretical Economics*, 121(2):301–324, 1965.

[185] L. Shapley. Stochastic games. *Proceedings of the National Academy of Sciences*, 39(10):1095–1100, 1953.

[186] L. Shapley. A note on the Lemke-Howson algorithm. In *Pivoting and Extension: In honor of A.W. Tucker*, volume 1 of *Mathematical Programming Studies*, pages 175–189. Springer, 1974.

[187] Y. Shoham. Computer science and game theory. *Commun. ACM*, 51(8):74–79, 2008.

[188] Y. Shoham and K. Leyton-Brown. *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge University Press, 2009.

[189] A. Simaitis. *Automatic Verication of Competitive Stochastic Systems*. PhD thesis, Department of Computer Science, University of Oxford, 2014.

[190] S. Soudjani, C. Gevaerts, and A. Abate. Faust^2: Formal abstractions of uncountable-state stochastic processes. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 272–286. Springer Berlin Heidelberg, 2015.

[191] J. Sun, Y. Liu, J. Dong, and J. Pang. PAT: Towards flexible verification under fairness. In *Proc. CAV'09*, volume 5643 of *LNCS*, pages 709–714. Springer, 2009.

[192] M. Svoreňová and M. Kwiatkowska. Quantitative verification and strategy synthesis for stochastic games. *European Journal of Control*, 30:15–30, 2016.

[193] M. Svoreňová, I. Černá, and C. Belta. Optimal control of MDPs with temporal logic constraints. In *52nd IEEE Conference on Decision and Control*, pages 3938–3943, 2013.

[194] W. Thomas. *Languages, Automata, and Logic*, pages 389–455. Springer Berlin Heidelberg, 1997.

[195] I. Tkachev and A. Abate. Formula-free finite abstractions for linear temporal verification of stochastic hybrid systems. In *Proc. HSCC '13*, pages 283–292. ACM, 2013.

[196] M. Todd. The many facets of linear programming. *Mathematical Programming*, 91(3):417–436, 2002.

[197] S. Topaloglu. A multi-objective programming model for scheduling emergency medicine residents. *Computers & Industrial Engineering*, 51(3):375–388, 2006.

[198] A. Toumi, J. Gutierrez, and M. Wooldridge. A tool for the automated verification of Nash equilibria in concurrent games. In *Proc. ICTAC'15*, volume 9399 of *LNCS*, pages 583–594. Springer, 2015.

[199] T. L. Turocy. A dynamic homotopy interpretation of the logistic quantal response equilibrium correspondence. *Games and Economic Behavior*, 51(2):243–263, 2005. Special Issue in Honor of Richard McKelvey.

[200] T. L. Turocy. Computing sequential equilibria using agent quantal response equilibria. *Economic Theory*, 42(1):255–269, 2010.

[201] M. Ummels. *Stochastic Multiplayer Games: Theory and Algorithms*. PhD thesis, RWTH Aachen University, 2010.

[202] G. van der Laan, A. Talman, and L. van der Heyden. Simplicial variable dimension algorithms for solving the nonlinear complementarity problem on a product of unit simplices using a general labelling. *Mathematics of Operations Research*, 12(3):377–397, 1987.

[203] C. von Essen and B. Jobstmann. Synthesizing efficient controllers. In *Proc. VMCAI'12*, volume 7148 of *LNCS*, pages 428–444. Springer, 2012.

[204] J. von Neumann. Zur theorie der gesellschaftsspiele. *Mathematische Annalen*, 100:295–320, 1928.

[205] J. von Neumann and O. Morgenstern. *Theory of Games and Economic Behavior*. Princeton University Press, 1944.

[206] B. von Stengel. Computing equilibria for two-person games. volume 3 of *Handbook of Game Theory with Economic Applications*, pages 1723–1759. Elsevier, 2002.

[207] A. Wächter. Short tutorial: Getting started with IPOPT in 90 minutes. In *Combinatorial Scientific Computing*, number 09061 in Dagstuhl Seminar Proceedings. Leibniz-Zentrum für Informatik, 2009. github.com/coin-or/Ipopt.

[208] A. Wächter and L. Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106(1):25–57, 2006.

[209] I. Wegener. *Branching Programs and Binary Decision Diagrams*. SIAM, 2000.

[210] R. Wilson. Computing equilibria of n-person games. *SIAM Journal on Applied Mathematics*, 21(1):80–87, 1971.

[211] C. Wiltsche. *Assume-Guarantee Strategy Synthesis for Stochastic Games*. Phd thesis, University of Oxford, 2015.

[212] E. Wolff, U. Topcu, and R. Murray. Optimal control of non-deterministic systems for a computationally efficient fragment of temporal logic. In *52nd IEEE Conference on Decision and Control*, pages 3197–3204, 2013.

[213] H. Xu, U. Topcu, and R. M. Murray. A case study on reactive protocols for aircraft electric power distribution. In *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*, pages 1124–1129, 2012.

[214] Z. Yang. *Computing Equilibria and Fixed Points*. Springer, 2014.

[215] Z. Yin, A. Jiang, M. Tambe, C. Kiekintveld, K. Leyton-Brown, T. Sandholm, and J. Sullivan. TRUSTS: Scheduling randomized patrols for fare inspection in transit systems using game theory. *AI Magazine*, 33(4):59, 2012.

[216] Z. Zamani, S. Sanner, and C. Fang. Symbolic dynamic programming for continuous state and action mdps. In *Proc. AAAI'12*, pages 1839–1845. AAAI Press, 2012.

[217] W. I. Zangwill and C. Garcia. *Pathways to Solutions, Fixed Points, and Equilibria*. Computational Mathematics Series. Prentice-Hall, 1981.

[218] E. Zermelo. Über eine anwendung der mengenlehre auf die theorie des schachspiels. *Proc. of the 5th International Congress of Matematicians*, 2:501–504, 1913.

[219] Q. Zhu and T. Başar. Dynamic policy-based IDS configuration. In *CDC'09*, pages 8600–8605. IEEE, 2009.

[220] Q. Zhu, H. Li, Z. Han, and T. Başar. A stochastic game model for jamming in multi-channel cognitive radio systems. In *Proc. ICC'10*, pages 1–6. IEEE, 2010.