Assume-Guarantee Strategy Synthesis for Stochastic Games

Clemens Wiltsche

St Cross College

University of Oxford

A thesis submitted for the degree of
Doctor of Philosophy

Michaelmas 2015

Abstract

This thesis presents a framework for the automatic strategy synthesis from quantitative specifications, in order to control autonomous systems. We model systems as turn-based two-player zero-sum stochastic games, which are able to express both stochastic and nondeterministic environmental uncertainty. Given a system model and a specification, we define the strategy synthesis problem as that of finding a strategy for the system that is winning against every environment. Since large, complex systems are typically built from multiple components, we consider synthesis of strategies for the individual components separately, which can then be composed to a winning strategy for the full system. Modelling interaction between components is facilitated using *assume-guarantee rules*, which can express contracts such as "maintain the room temperature of at least 20 °C, as long as the windows are closed at least 30% of the time." For the synthesis of strategies for the individual components, we develop synthesis algorithms for Boolean combinations of long-run objectives, specifically, maintaining above a threshold (i) a mean-payoff, almost surely; (ii) an expected mean-payoff; (iii) a ratio of rewards, almost surely; or (iv) a ratio of expected rewards. We implement our algorithms in the PRISM-games 2.0 tool and demonstrate their viability on four case studies.

Acknowledgements

First and foremost, I would like to express my gratitude towards my supervisor, Marta Kwiatkowska, without whose guidance and generous support this research could not have been completed. I particularly thank Aistis Šimaitis for inspiring me to delve into the world of stochastic games. I also thank Vojtěch Forejt, Dave Parker and Nicolas Basset for always finding time to discuss ideas and their helpful input, and Ufuk Topcu for hosting me during an inspiring visit at UPenn. My thanks go to all my collaborators, colleagues and friends, both in Oxford and further afield, for making the past years a wonderful time. Finally, I would like to thank my family, especially my parents, Renate and Norbert, for supporting me in following my curiosity, and my wife, Sara, for patiently encouraging me and sharing in both the frustrations and successes.

Clemens Wiltsche
Oxford, October 2015

Contents

List of Figures

Chapter 1

Introduction

At the heart of every successful endeavour is a strategy: acting in the presence of an adapting, unpredictable environment requires a determined and structured approach. In virtually every undertaking, we rely on a strategy to achieve our objectives, and the advent of computers opened up the possibility of refining our strategies to unprecedented sophistication. The usefulness of strategies extends well beyond their traditional areas of application, such as in economics or in the military. Since the premise of a strategy is to ensure that an objective is met irrespective of the environment behaviour, the same requirement is shared by controllers for systems traditionally in the domain of control engineering, such as robots, electric power generators, or transport. These applications exemplify that the configurations of a system have to obey predetermined criteria, that the system is affected by its independent environment, and that the strategy can only influence a limited set of variables. Thus, controlling a system becomes a question of finding a winning strategy in a game, where the controller plays against the environment. The appeal of posing control questions in a game-theoretic setting is that we can greatly extend the versatility of controllers that we can design, while maintaining provable correctness with respect to specifications that can be understood at a higher, abstract level.

The need to include the influence of the uncontrollable environment on the system state is a consequence of not considering the systems in isolation. However, often the environment does not behave completely arbitrarily, and so we are interested in achieving a control objective G under some assumption A on the environment, which gives rise to a winning condition of the form $A \rightarrow G$, where we write $\rightarrow$ for logical implication. We can take this idea of placing assumptions on the environment further, and establish contracts between subsystems of a larger system. This approach is par-

ticularly relevant if the system is designed as a composition of multiple components, for example, to manage the design complexity or to increase reliability due to physical separation and by ensuring that components interact in a controlled way. Contracts between components permit one to consider each component in isolation, and to perform analysis under the assumption that all components obey their contracts. This leads us to the development of a framework of assume-guarantee strategy synthesis, in which a strategy for the composed system, guaranteeing a global specification, can be derived by synthesising strategies only for the individual components, assuming adherence to the contracts. A key benefit of assume-guarantee synthesis is the improved scalability of the algorithms, resulting from only having to analyse several smaller systems, albeit, in our case, at the cost of restricting the class of strategies under consideration.

Aims. The main aim of the presented research is the development of algorithms and tools for the automatic synthesis of controllers for autonomous systems from quantitative specifications, that make the controllers more reliable and more convenient to configure and maintain. We provide a comprehensive framework for the *assume-guarantee synthesis* of strategies for systems that exhibit both stochastic and nondeterministic behaviour, aimed at improving scalability of strategy synthesis. Thus, we need to formalise the compositional design of stochastic systems operating in an uncontrollable environment, and provide a framework for the synthesis of strategies.

Approach. This thesis focuses on the development of controllers for autonomous systems via strategy synthesis for turn-based zero-sum stochastic games [121]. Stochastic games can be seen as an extension of Segala's probabilistic automata (PAs) [118], with the difference that the nondeterministic choice of transitions is controlled by separate players that may compete against each other. Since stochastic games can model probabilistic behaviour, they are particularly attractive for the analysis of systems that exhibit uncertainty quantifiable by probability distributions.

Considering a large system as a collection of interacting subsystems has been widely employed for *assume-guarantee verification* of systems, which can be modelled, for example, as PAs, with the desired behaviours expressed via temporal logic, or via maximisation of rewards or minimisations of costs. While verification is useful to check whether or not a given design satisfies a specification, it does not automate the design process of the controlled system. To this end, automated synthesis of strategies in a monolithic (non-compositional) setting has been applied to controller design using a variety of formal models and specifications. Towards an assume-guarantee

framework for strategy synthesis, we define a composition operator for stochastic games, which is closely related to the parallel composition of PAs due to Segala [118] and to that of interface automata [54], where interaction between subsystems is modelled via synchronisation on common actions. By applying a strategy to a stochastic game to resolve the controllable nondeterminism, one obtains a PA, where only uncontrollable nondeterminism remains. This observation allows us to utilise rules for assume-guarantee PA verification in order to derive synthesis rules for games.

When designing control systems, one is often interested in trade-offs between objectives (for example, minimising time to arrival and fuel consumption at the same time), and hence we consider multi-objective specifications expressed as Boolean combinations of objectives. We develop a series of transformations of the objectives, and so our primary problem reduces to that of synthesising strategies that maintain a vector of mean-payoffs almost surely above a threshold. We solve this strategy synthesis problem via a multi-dimensional dynamic programming characterisation, which is a popular method in optimal control.

Finally, to evaluate the effectiveness of our methods, we implement our assume-guarantee synthesis framework as an extension of the PRISM-games tool, adding multi-component and multi-objective strategy synthesis functionality. Using our tool, we subsequently study a collection of case studies relevant to controller design for autonomous systems.

Contributions

We develop a framework for the assume-guarantee synthesis of strategies for turn-based two-player zero-sum stochastic games, and focus on providing novel algorithms and tools for the design of controllers for autonomous systems. We are interested in strategies that satisfy quantitative multi-objective properties, which are defined using rewards accumulated during system execution. The main contributions are presented in Chapters 4–8, which we briefly summarise.

- In Chapter 4, we introduce our assume-guarantee framework. We define a synchronising composition of stochastic games, which is based on the composition operations of PAs and interface automata [54]. A fundamental property of our game composition is that strategies of the individual components can be composed to a strategy of the composed game, while preserving the winning conditions. More concretely, we develop a set of *assume-guarantee strategy synthesis*

rules, which deduce global winning conditions of the composed game under the assumption that local winning conditions are satisfied in the component games. Our framework is independent of the synthesis methods used for the individual components, and is applicable to general winning conditions that are invariant under interleaving of actions, such as ratio or total rewards.

- In Chapter 5, we discuss the properties of multi-objective queries for stochastic games, where we focus on *long-run objectives*, specifically, maintaining the mean-payoff or the ratio of rewards above a threshold almost surely, or maintaining the expected mean-payoff or the ratio of expectations above a threshold. For expectation objectives we consider arbitrary Boolean combinations, while for satisfaction objectives we consider conjunctions, and we develop a set of transformations of these multi-objective queries to conjunctions of almost sure mean-payoff objectives. We further discuss the approximation of Pareto sets, which is instrumental in our framework to derive the local specifications.

- In Chapter 6, we develop a synthesis method for strategies achieving conjunctions of almost sure mean-payoff objectives, and we establish that the corresponding achievability problem, that is, deciding whether a winning strategy exists, is in co-NP. To synthesise strategies, we show that the mean-payoff is almost surely non-negative if the strategy ensures at every step that the expected total reward is above a finite shortfall. We characterise these shortfall vectors via a Bellman operator, and from its fixpoint we can then construct strategies; while infinite memory is necessary in general, we can obtain finite strategies that are ε-optimal. Further, the strategies we construct use internal memory that is stochastically updated, enabling a more compact representation.

- In Chapter 7, we present our implementation of the framework, an extension of the PRISM-games tool. We explain the extensions required for modelling of multi-component stochastic games, discuss the data structures that we use in our algorithms, highlight heuristics and approximations we use to increase efficiency of our implementation, and explain the graphical user interface.

- In Chapter 8, we apply our methods to a set of case studies, demonstrating the viability of our framework and tool implementation for synthesising controllers for autonomous systems modelled as stochastic games. Particularly in order to illustrate the modelling paradigms of stochastic games for controller synthesis, we study the problem of safely steering an autonomous car in an urban context,

and path planning for an unmanned aerial vehicle (UAV) interacting with a human operator. We use conjunctions of total expected rewards based on prior work in these two case studies, which can also be synthesised with our tool. We then employ our assume-guarantee synthesis framework in order to find strategies for the control of electrical power distribution on an aircraft, and for regulating the temperature in a building with multiple rooms. To reflect the continual nature of the modelled processes, we use long-run (ratio) rewards in these latter two case studies.

Thesis Outline

In Chapter 2 we review the related work of this thesis, and in Chapter 3 we give the technical background and known results. We present the main theoretical contributions in Chapters 4–6. We describe the tool implementing our assume-guarantee framework in Chapter 7, and give four case studies in Chapter 8. We conclude with Chapter 9 summarising our work and highlighting possible future directions.

Publications

Some of the work presented in this thesis has been previously published in jointly authored papers. In [11] the assume-guarantee strategy synthesis framework was introduced, which forms the basis of Chapter 4. I developed the game composition framework using PA rules, and the compositional Pareto set computation. Nicolas Basset contributed the proof via functional simulations and helped in formalising the results. The main synthesis results for long-run properties are published in [10], and presented here in Chapter 6. I contributed the strategy construction via the reduction to expected energy, and Nicolas Basset contributed the completeness proofs. I also developed the aircraft control case study, featured in Section 8.3, while visiting Ufuk Topcu at the University of Pennsylvania (UPenn), who provided the key direction. The reductions in Chapter 5 are developed in collaboration with Nicolas Basset, and are part of a journal paper in preparation [12]. While at UPenn, I collaborated on the study of controller synthesis for human-UAV interaction published in [63]. The study was led by Lu Feng, who performed the main modelling task using Markov decision processes. I contributed the extension to stochastic games, which is the focus of Section 8.2. Earlier in my doctoral study, I was part of the collaboration [41] concerning stochastic games with multiple total expected rewards. I was the main driver of the

reduction theorem from Boolean combinations of objectives to conjunctions, as this is a key component for contracts between components, see Section 4.2. The complexity results were contributed by my collaborators. In [43] we present strategy synthesis for total expected rewards. I implemented the Pareto set computations and strategy synthesis algorithm. I also authored the autonomous urban driving case study, featured in Section 8.1. The strategy construction and reduction from linear temporal logic objectives was contributed by Aistis Šimaitis.

In [10, 11, 43, 63], the experimental results are obtained via my implementation of the synthesis algorithms, presented in Chapter 7, which extends the PRISM-games tool [39] to allow compositional modelling and analysis. The tool, developed with support from David Parker, is released as PRISM-games 2.0 at `http://www.prismmodelchecker.org/games/`, and is described as a tool paper [86].

This research was supported by ERC Advanced Grant VERIWARE and EPSRC.

Chapter 2

Related Work

Contents

Games have found versatile applications in areas ranging from artificial intelligence, through modelling and analysis of financial markets, to control system design and verification. The game model intuitively consists of an arena with a number of positions, and two or more players that move a token between positions. The rules of the game determine the allowed moves between positions, and a player's winning condition captures which positions or sequences of positions are desirable for the player. The rigorous treatment of games as an independent field began with the seminal work of John von Neumann on zero-sum games [100], and the characterisation of equilibria and their existence in non-zero-sum games goes back to John Nash [99], who defined the eponymous *Nash equilibrium*, which prescribes a strategy for each player such that no player can improve its outcome by unilaterally changing its strategy. Solving non-zero-sum games is beyond the scope of this thesis, and we give a brief discussion of their relation to the winning conditions we consider in Section 3.7. Solution concepts for games vary depending on the types of games and specifications used, and we give in this chapter an overview of the research that is related to this thesis.

We start by exploring, in Section 2.1, the strategy synthesis problem and traditional synthesis techniques, focusing on strategies that optimise a single quantitative objective. In Section 2.2, we discuss existing work on extensions toward multi-

objective verification and synthesis, which is a key element in our assume-guarantee framework. In Section 2.3, we review formalisms for the compositional modelling of systems, as well as existing methods for compositional verification and synthesis. Finally, in Section 2.4, we summarise applications of strategy synthesis to the design and control of autonomous systems, and existing tools developed for this purpose.

2.1 Strategy Synthesis

Strategy synthesis for a system is an automatic process to construct a *strategy*, which is a mapping from past observations about the system state to decisions about the next state, such that the behaviour of the system satisfies a given *specification* when the strategy is implemented. Thus, a strategy controls *nondeterministic* choices in a system, and, since the systems we consider are modelled as games with two *players*, each player is responsible for controlling its own nondeterministic choices and has its own strategy. We consider *stochastic games*, introduced by Shapley [121] (sometimes called Markov games or competitive Markov decision processes), whose behaviours are determined not just by the strategies but also by probability distributions on certain transitions between states. Thus, strategy synthesis for Player 1 in a stochastic game amounts to finding a strategy against all strategies of Player 2, so that the resulting probability distribution over behaviours satisfies the specification. Specifically, we are interested in turn-based two-player games, where Player 1 and Player 2 take turns in making choices. If there is only one player present in a stochastic game, the system is a *Markov decision process* (MDP) [110]. We make these concepts precise in Chapter 3. We remark that *robust MDPs* [101] are an alternative stochastic model that allows controllable and uncontrollabe nondeterminism, where the environment can choose from a compact set of transition probabilities; each choice represents Player 2 selecting a distribution in a stochastic game, similar to the approach discussed in Section 8.4.4. Stochastic games allow more structure than robust MDPs since the choices of Player 2 can be discrete, but since we consider stochastic games with finite branching only, we cannot encode, for example, elliptically constrained sets of distributions for Player 2.

Qualitative Specifications. In non-stochastic systems, a *qualitative* specification can be seen as a mapping $\Phi : \Omega \to \{\texttt{true}, \texttt{false}\}$ from behaviours Ω to truth values. For non-stochastic systems, the strategy synthesis problem for qualitative specifications expressed over infinite sequences of events was stated by Church in [47], and subsequently solved by Büchi and Landweber in 1969 [23], establishing the link between specification logics and game theory. Synthesis of strategies, represented as

finite-state automata, was shown to be 2EXPTIME-complete for the temporal logic CTL* in [83], which is able to express properties on the temporal succession of events during system execution, for example "all executions have recurring safety checks, and there is an execution where the task is completed." For the class of linear temporal logic (LTL) formulae called GR1, where a *guarantee* must be satisfied under an *assumption*, efficient, polynomial-time algorithms were developed [107]. GR1 formulae are used, for example, to specify autonomous systems [104, 140], and their versatile assume-guarantee format is one motivation to consider Boolean combinations of objectives in this thesis.

In stochastic systems, qualitative specifications typically require that almost all behaviours satisfy a property $\Phi : \Omega \to \{\texttt{true}, \texttt{false}\}$, that is, $\mathbb{P}(\Phi) = 1$, where $\mathbb{P}$ is a measure defined over behaviours Ω of the system. The presence of probabilities in the system means that the specification still holds even if a set of paths with probability measure zero violates the property. Synthesis in stochastic systems from qualitative LTL and CTL* specifications was studied in [117], and for parity objectives in [32]. For qualitative specifications, it is only relevant which states are accessible from each other, while the precise probability distributions in the system do not change the outcome for finite systems. For the results in this thesis, strategies that reach a set of states with probability one (that is, almost surely), are particularly relevant, as we use such strategies in our completeness proof for expectation objectives.

Quantitative Specifications. Attaching rewards and probabilities to transitions allows one to evaluate the strategies not just according to whether they achieve an objective or not, but also by how well they achieve an objective [16]. The traditional objectives in strategy synthesis are for a player to maximise a reward, or payoff, that it accumulates during the executions of the game, evaluated as total reward or mean-payoff (average reward). We call such objectives *quantitative*, since the specification requires the strategy to optimise a numerical value. Objectives requiring to maintain a payoff above a given threshold are also considered quantitative.

Our main focus is on properties that are determined entirely by the system behaviour in the long run, and are independent of finite prefixes, such as mean-payoff. The mean-payoff on a path is defined by summing up the rewards along the path, dividing by the length, and letting the length go to infinity. Hence, two versions of mean-payoff are studied, depending on whether the limit is defined via the limit inferior or the limit superior. Games with long-run objectives, such as mean-payoff, are qualitatively determined [71], that is, either one player has a winning strategy, or the other player has a strategy to spoil. Qualitative determinacy differs from quan-

titative determinacy [94], according to which the players may only have strategies to approach cumulated rewards arbitrarily close.

Strategy synthesis for mean-payoff objectives in non-stochastic games has been studied in [59, 144], and for MDPs and stochastic games in [64, 95, 110]. A key characteristic for non-stochastic games with mean-payoff objectives is that strategies for neither player have to remember the history to win, that is, they are *memoryless* [59]. While memoryless strategies are sufficient for single objectives, this is no longer the case in our multi-objective setting, as we explore in Section 2.2. In [25], it is shown that a strategy achieves a mean-payoff in a non-stochastic game if and only if it achieves a corresponding energy objective, requiring that the total cumulative reward never falls below zero along any path compatible with the strategy. We utilise this observation to develop strategy synthesis for our mean-payoff objectives.

Further, for stochastic games, quantitative ω-regular objectives were studied [33, 56], as well as a reduction from parity to mean-payoff objectives [29]. While these objectives are relevant for our study of strategy synthesis for autonomous systems, to our knowledge no such reduction for multiple long-run objectives has been explored so far. Another instance of long-run objectives are *ratio objectives*, defined by assigning to each path the ratio of two total rewards, and taking, for example, expectations [135]. A ratio between rewards r and c in the numerator and denominator, respectively, expresses a trade-off between maximising r and minimising c, and can thus be used to evaluate the robustness or efficiency of systems [17]. Specifications defined using ratios of rewards are particularly applicable to our assume-guarantee framework, since systems working together (that is, components of a larger system) can form a consensus on which quantities are averaged over by sharing the same denominator c.

2.2 Towards Multiple Objectives

Often it is not sufficient to consider a single objective. For example, one may require a controller for a power plant that maximises power output, minimises wear, minimises the downtime, maximises economic feasibility, and so on, all at the same time. To realise trade-offs between potentially conflicting objectives, multi-objective optimisation has been widely studied in the operations research community [49], with applications, for example, to engineering [92] and the medical field [130].

Pareto Sets. When optimising several quantitative objectives in a multi-objective problem, not all objectives can be independently optimised. Rather, we have to

consider the set of achievable trade-offs, called the *Pareto set* [105]. Strategy synthesis for multiple objectives therefore requires to select a realisable trade-off from the Pareto set, and to construct a strategy specific to this selection. Existing algorithms for computing Pareto sets include, for example, the weighted sum method [93, 142], or sandwich algorithms that compute successive under- and over-approximations of the Pareto set [113]. However, these methods typically rely on properties related to determinacy in the game-theoretic sense, which limits their applicability in our setting (as we explain in Section 3.5). Visualising the Pareto front for high-dimensional settings was studied in [88], and we employ similar ideas in the graphical user interface of our tool implementation.

MDPs. Multi-objective problems for stochastic systems have been studied mostly in the context of MDPs. This included expected discounted total rewards [35], ω-regular properties in [60], and undiscounted total rewards combined with ω-regular properties [67, 68]. Mean-payoff in stochastic systems has been studied in two variants, both for objectives defined using the expected mean-payoff, and objectives requiring the multi-dimensional mean-payoff to be above a threshold with a given probability (satisfaction) [18]. A further generalisation of satisfaction objectives are *percentile* objectives, where each objective requires the mean-payoff to be above a threshold with a different probability [112]. Solutions for MDPs are typically based on formulating a linear program (LP) characterising the achievable trade-offs, and extracting a strategy from the resulting solution.

Non-Stochastic Games. In games, attention has mostly been directed at the study of conjunctions of objectives. In the case of MDPs, and, more generally, systems with a single, controllable, type of nondeterminism, strategy synthesis for Boolean combinations of objectives in disjunctive normal form can be performed as a series of synthesis problems for conjunctions, by considering each disjunct separately. In games, however, the universal quantification over the adversary strategies prevents this transformation, and so the synthesis problem for Boolean combinations is in general harder than that for conjunctions. Non-stochastic games with conjunctions of objectives have been studied for total rewards, for example, in [82, 143]. More recently, conjunctions of mean-payoff and energy objectives were studied in [26] for finite strategies, and deciding the winner was shown to be co-NP complete; the corresponding strategy synthesis problem was subsequently studied in [36], where also incremental symbolic approaches are investigated. The difference of the definitions of mean-payoff via the limit inferior and limit superior are discussed in a series of

papers that establish decidability for subclasses of Boolean combinations in [37, 133], and undecidability for general Boolean combinations in [132].

Stochastic Games. The problem of precisely achieving an expected total reward can be expressed as a conjunction of a minimisation and maximisation of the same expected total reward, and is the subject of [40]. The work in [41, 43, 124] broadens the discussion to Boolean combinations of expected rewards in stopping games. Non-zero-sum stochastic games for more than two players, where each player has a single expected discounted total reward objective, are discussed in [91]. The probabilistic branching time logic PCTL also gives rise to quantitative specifications, and is the subject of [6, 19], where non-determinacy as well as several (un-)decidability results are established. We note that the synthesis problem for reaching multiple target sets in the class of stopping games is a special case of the synthesis problems for both expected total rewards and expected mean-payoffs.

For multi-dimensional mean-payoffs, strategies have been studied that ensure that the value accumulated along the infinite duration the game is played approaches a given set via some distance metric. This concept was introduced for repeated stochastic games in [15], where it is also established that these games are not determined, that is, in general, there are targets that neither player can force the value of the mean-payoff to converge to. Based on this work, [123] defines winning strategies in stochastic games where at least one state is recurrent under all considered strategies, but their suggested algorithm does not provide guarantees on the approximate strategies. To the best of our knowledge, multi-dimensional mean-payoff under satisfaction semantics has received little attention. We note that in [71] a satisfaction objective is combined with a Büchi condition, both in conjunction and disjunction, where the qualitative nature of the Büchi condition is exploited, and hence this approach is not directly applicable to our setting with multiple quantitative objectives.

2.3 Compositional Reasoning

While synthesis is attractive due to its potential to automatically deliver correct-by-construction controllers, the high computational complexity of even restricted problem classes has so far prevented application to large-scale systems, where formal verification has already shown promising results [85, 111]. One approach in verification to address the challenge of scalability is that of *compositional reasoning*, where a system is considered as a composition of several components, and local properties

are established for each component, which, using *assume-guarantee rules*, allow one to infer properties about the composed system.

Compositional Modelling. Compositional reasoning is based on modelling a system as a set of components that may interact according to mechanisms such as variable sharing, or synchronisation of actions. Composing stochastic systems is studied for Segala's probabilistic automata (PAs) in [118], which are labelled probabilistic transition systems, that generalise MDPs by allowing that at each state several outgoing moves carry the same label. Composition has also been studied for Interactive Markov chains (IMCs) in [74], which allow action-labelled transitions as well as random delays similar to continuous-time Markov chains. However, both PAs and IMCs do not capture the distinction between players required for composing stochastic games. Several notions of parallel composition of non-stochastic games have been proposed. In [70] extensive form games are considered. In [69], the strategies of the components have to agree in order for the composed game not to deadlock, leading to a complex strategy composition operation. In [55], probabilistic systems are composed, where variables are synchronised and partitioned into controllable (internal and interface) variables, as well as uncontrollable (external) variables, which can be interpreted as Player 1 and Player 2 actions, respectively. Similarly, Input/Output Automata (IOAs) contain an input and an output alphabet in order to model interaction between components [90]. Adding probabilities results in probabilistic IOAs (PIOAs) [46, 139], where schedulers for controlling the inputs and outputs can be seen as strategies in stochastic games. Scheduling decisions between several composed PIOAs are made locally, with a token passed between components by an arbiter to maintain a unique active component, and we adopt a similar scheduling scheme in the game composition that we develop. A major restriction of (P)IOAs is their reliance on the *input-enabledness* assumption, requiring that, in each state, every input action must be enabled. Hence, when scheduling actions, both inputs and outputs may be decided on simultaneously and independently, which in a game interpretation corresponds to a concurrent game between the output scheduler (Player 1) and the input scheduler (Player 2). This assumption is relaxed in Interface Automata (IAs) [54], but at the cost of a more expensive composition operation that is no longer purely syntactic. In the same paper, (non-stochastic) single-threaded IAs are defined, where the set of states is partitioned into *running* and *waiting* states, which can be seen as Player 1 and Player 2 states in a turn-based game. We are not aware of a study of single-threaded IAs with probabilities that would allow an interpretation as turn-based games. Our composition of games is most closely related to that of PAs, but also draws ideas from IAs.

Assume-Guarantee Verification. The main body of literature on compositional reasoning is concerned with *verification*, that is, checking whether all behaviours of a componentised system satisfy a specification by just checking all the behaviours of the subsystems. Early approaches to assume-guarantee reasoning include the temporal logic based frameworks [48, 109]. More recent work combines the assume-guarantee paradigm with automated learning of the assumptions [44, 106]. For probabilistic systems, an early approach to assume-guarantee reasoning has been discussed in [55]. Based on multi-objective model checking, assume-guarantee verification for PAs has been studied in [85], and learning of assumptions via the L^* algorithm has been applied in this framework [62]. In these frameworks, however, the assume-guarantee rules are incomplete, which is of particular concern for fully automated approaches, as then termination is not guaranteed; a complete approach for probabilistic systems is presented in [80], where properties are specified using a strong simulation relation.

Assume-Guarantee Synthesis. An assume-guarantee framework requires defining a composition of components, so that composing winning strategies of the components yields a winning strategy for the composition. One approach is to synthesise a specification such as $\varphi_1 \wedge \varphi_2 \wedge \ldots$ by first considering only φ_1, then augmenting the resulting strategy by also taking φ_2 into account, and so on [8, 65], which can be seen as iteratively over-approximating the allowable behaviour (to ensure liveness) and under-approximating the undesirable behaviour (to ensure safety). On the other hand, rather than considering composition of specifications as in the conjunction above, another approach is to consider a composition of individual games, synthesise controllers for each such game, and derive from them a controller for the composition. In [28], an assume-guarantee rule is discussed for non-stochastic systems, where the synthesis problem is transformed to solving a 3-player game, where the third player is considered a fair scheduler. However, this approach is not compositional in the sense that the synthesis problem can be solved as a series of smaller, local, sub-problems, but all components of the composed system must be considered at the same time; scalability is achieved via abstraction refinement, which is beyond the scope of this thesis and challenging due to our use of multi-objective stochastic games. While [69] discusses synthesis of a global strategy from local strategies, the strategy composition requires an additional composer to be computed, which is polynomial in the size of the full composed game. A further related approach is finding a composition of components taken from a given library, so that it satisfies a global LTL specification [89, 98]. This approach is in contrasts to our setting, where we consider a fixed parallel composition of components.

The problem of synthesising strategies for components whose composition according to a fixed architecture satisfies a given global LTL specification is undecidable [108], since strategies in the components need to accumulate sufficient knowledge in order to make choices that are consistent globally, while only being able to view the local history, as discussed in [78]. We expect synthesis of local strategies directly from a global quantitative specification to also be undecidable. However, in our setting, each strategy is synthesised on a single component, considering all other components as black boxes, and hence adversarial. To impose dependencies between components, assume-guarantee synthesis is a convenient way of encoding assumptions on other components and the overall environment in the local specifications.

2.4 Applications and Tools

We conclude this chapter by reviewing existing applications of strategy synthesis for the control of (autonomous) systems, and by discussing existing tools.

Applications. Strategy synthesis for an autonomous system can be interpreted as finding a controller that maintains the system state in a desirable condition. Inputs are received from *sensors*, and the strategy produces commands to control the *actuators*. Classical control theory is typically concerned with the continuous dynamics (in time and/or space) of systems, but since we are working with finite-state games we are mainly concerned with the discrete dynamics. The continuous dynamics can be discretised and thus approximated in the discrete setting, which has been employed for the control of autonomous systems, allowing one to synthesise controllers that satisfy temporal logic specifications in a non-stochastic setting [13, 81], as well as for stochastic systems [127]. In the spirit of hybrid systems [73], operating in the discrete domain means that we can model higher-level dynamics such as transitions between operating regimes. The errors introduced in the discretisation of continuous space models can be formally bounded, see, for example [126, 128] for stochastic systems, an approach that we discuss in a case study in Section 8.4.

The use of techniques such as formal verification, validation and synthesis from specifications for autonomous vehicles has been advocated, for example, in [24, 131], resulting from observations during the DARPA Urban Challenge 2007 [51]. Formal analysis, including the automatic synthesis of controllers from formal systems models, has also attracted interest for industrial applications such as aerospace, energy and industrial automation, for example in the iCyPhy consortium [66], where a strong case is made for the compositional design of controllers. In the context of security and

defence, games with stochasticity have been applied to compute strategies that are deployed to support decision making procedures, for example in patrol planning [141], port defence [122], and infrastructure protection [22].

Tools. Several tools for synthesis of strategies have been developed, and we highlight tools relevant to the topics that we focus on in this thesis. QUASY [31] is a tool for the synthesis of strategies for MDPs and non-stochastic games with mean-payoff objectives. An implementation of strategy synthesis methods for single-dimensional expected ratio reward objectives is presented in [134]. MultiGain synthesises strategies for MDPs from multi-dimensional mean-payoff objectives, both for expectations and satisfaction semantics [21]. For stochastic games, GIST is a tool synthesising strategies with qualitative ω-regular properties, that is, with almost sure and positive satisfaction. Moreover, the tool GAVS+ offers algorithms for value- and policy iteration in stochastic games for reachability objectives [45]. The PRISM-games 1.0 tool implements synthesis of one-dimensional expected total reward and reachability objectives for stochastic games [39], and is itself an extension of PRISM [84], a tool for the verification and synthesis of probabilistic systems. The MOCHA tool implements model checking and synthesis for alternating-time temporal logic (ATL) specifications for non-stochastic games (formulated as reactive modules [1]), as well as automatic checking of assume-guarantee queries formulated using refinement relations [2]. Finally, the TuLiP toolbox [138] provides synthesis for GR1 specifications via finite-state abstractions of systems with linear (continuous) dynamics.

2.5 Summary

In this chapter we discussed previous work in the areas of strategy synthesis and compositional analysis, which form a foundation upon which we build the contributions of this thesis. Our work on strategy synthesis for multi-objective stochastic games generalises three branches in strategy synthesis: it augments the multi-objective MDP branch, the non-stochastic game branch, and the single-objective stochastic game branch, and each extension is the source of new challenges. Further, our assume-guarantee framework encompasses ideas from several compositional frameworks, both for systems with stochasticity and with uncontrollable environment, but, in order to provide the capability for assume-guarantee strategy synthesis, we have to fine-tune the game composition operator. To validate our results, we implement our framework as PRISM-games 2.0, an extension of the PRISM-games tool, and draw on the previous case studies to justify the models that we analyse.

Chapter 3

Background

Contents

In this chapter we introduce the technical background material, including known results that we employ in this thesis. We first give notations that we use throughout in Section 3.1. In Section 3.2, we review the models of stochastic systems that we use (discrete-time Markov chains, probabilistic automata and stochastic games). In Section 3.3, we define our formulation of strategies, which use stochastically updated internal memory, and we explain how to apply them in order to control systems. In Section 3.4, we discuss the specifications that we employ throughout the thesis, which are Boolean combinations of objectives defined using mean-payoffs and ratios of rewards under expectation and almost sure satisfaction semantics. Section 3.5 is dedicated to reviewing previously known results on strategy synthesis in relevant non-stochastic and stochastic systems, as well as approaches to multi-objective optimisation problems. Finally, in Section 3.6, we introduce a running example.

3.1 Notation

We first introduce notations that we use throughout.

Tuples and Sequences. Given sets $Q_1, Q_2, \ldots$, we define the *Cartesian product* $Q_1 \times Q_2 \times \cdots$ to be the set of *tuples* $\{(q_1, q_2, \ldots) \,|\, \forall i \,.\, q_i \in Q_i\}$. For a tuple $q = (q_1, q_2, \ldots)$, we write q_i for the ith component; sometimes, for clarity, we write $[q]_i$. Given a set Q, we denote by Q^* and Q^ω the sets of finite and infinite *sequences* over Q, respectively. We denote by ϵ the empty sequence (note that $\epsilon \in Q^*$ but $\epsilon \notin Q^\omega$). Concatenation of two sequences κ and ρ is written $\kappa\rho$. Given a finite sequence ρ, we write ρ^n for the sequence that n times repeats ρ, and we write ρ^ω for the infinite sequence repeating ρ ad infinitum. Given a finite sequence $\rho = q_0 q_1 \ldots q_n$, we denote by $|\rho| \stackrel{\text{def}}{=} n + 1$ its *length*, and define $\mathsf{first}(\rho) \stackrel{\text{def}}{=} q_0$ and $\mathsf{last}(\rho) \stackrel{\text{def}}{=} q_n$. A *prefix* of a sequence ρ is a finite sequence κ such that $\rho = \kappa\rho'$ for some sequence ρ'. We sometimes use the notation $(q_i)_{i \geq 0}$ to abbreviate the sequence $q_0 q_1 q_2 \ldots$ (or simply an ordered or unordered collection $\{q_0, q_1, \ldots\}$).

Relations and Functions. A *relation* R between sets Q and E is a subset $R \subseteq Q \times E$, and it is a *function* from Q to E, written $f : Q \to E$, if no $q \in Q$ has more than one $e \in E$ such that $R(q, e)$. A function is *total* if $f(q)$ is defined for all q, and it is *partial* if there can be unassigned $q \in Q$. The notation $g \subseteq f$ is defined by recalling that the functions g and f are relations.

Probability. Denote by $\mathcal{P}(Q)$ the *powerset* of a set Q. A *measurable space* $(Q, \mathcal{Q})$ is such that $\mathcal{Q} \subseteq \mathcal{P}(Q)$ is a *sigma-algebra* of Q, that is, $Q \in \mathcal{Q}$ and $\mathcal{Q}$ is closed under complement and countable unions. Given measurable spaces $(Q, \mathcal{Q})$ and $(E, \mathcal{E})$, a function $f : Q \to E$ is $(\mathcal{Q}, \mathcal{E})$-*measurable* (or simply *measurable* if the context is clear) if, for all $e \in \mathcal{E}$, $f^{-1}(e) \in \mathcal{Q}$, that is, f preserves measurability. A (*discrete*) *probability space* is a tuple $(Q, \mathcal{Q}, \mathbb{P})$, where Q is a (countable) set of *outcomes*, the *event space* $\mathcal{Q}$ is a sigma-algebra of Q, and $\mathbb{P} : \mathcal{Q} \to [0, 1]$ is a *probability measure*, that is, for any countable collection $(Q_i)_{i \geq 0}$ of disjoint elements of $\mathcal{Q}$, $\mathbb{P}(\bigcup_{i \geq 0} Q_i) = \sum_{i \geq 0} \mathbb{P}(Q_i)$. Given a (discrete) probability space $(Q, \mathcal{Q}, \mathbb{P})$, and a measurable space $(E, \mathcal{E})$, a (discrete) *random variable* (RV) is a $(\mathcal{Q}, \mathcal{E})$-measurable function $X : Q \to E$. Given a discrete RV X, its *expectation* is $\mathbb{E}[X] \stackrel{\text{def}}{=} \int_{q \in Q} X(q)\mathbb{P}(q)$; and we define the *discrete probability distribution* (or *distribution* for short) μ over E by $\mu(e) \stackrel{\text{def}}{=} \mathbb{P}(X^{-1}(e))$ for all $e \in E$. The *support* of μ is $\mathsf{supp}(\mu) \stackrel{\text{def}}{=} \{e \in E \,|\, \mu(e) > 0\}$. We denote by $\mathsf{D}(E)$ the set of all distributions over E with finite support. A distribution $\mu \in \mathsf{D}(E)$ is *Dirac* if $\mu(e) = 1$ for some $e \in E$, and if the context is clear we just write e to denote such a distribution. We denote by $\mu^1 \times \mu^2 \in \mathsf{D}(E^1 \times E^2)$ the *product distribution* of the distributions $\mu^1 \in \mathsf{D}(E^1)$ and $\mu^2 \in \mathsf{D}(E^2)$, defined by $\mu^1 \times \mu^2(e^1, e^2) \stackrel{\text{def}}{=} \mu^1(e^1) \cdot \mu^2(e^2)$ for all $e^1 \in E^1$ and $e^2 \in E^2$.

Vector and Matrix Algebra. Synthesis for a multi-objective query with n objectives can be seen as a maximisation along n dimensions, one for each objective. Hence, we operate in the vector space $\mathbb{R}^n$. We use the notation $[\vec{v}]_s$ to refer to the sth component v_s of a vector $\vec{v}$, and $[A]_{s,t}$ to refer to the sth row and tth column $A_{s,t}$ of a matrix A. We use the standard vector dot product and matrix multiplication. Further, given vectors $\vec{u}, \vec{v}$ with equal dimensions, $\vec{u} \bullet \vec{v}$ is the vector $\vec{w}$ with $[w]_i \stackrel{\text{def}}{=} u_i \cdot v_i$ for all i. Given a vector $\vec{x}$, its *Euclidean norm* is $\|\vec{x}\| \stackrel{\text{def}}{=} \sqrt{\vec{x} \cdot \vec{x}}$, and its *supremum norm* is $\|\vec{x}\|_\infty \stackrel{\text{def}}{=} \sup_i x_i$. Correspondingly, the *induced matrix norm* of A is $\|A\|_\infty \stackrel{\text{def}}{=} \sup_i \sum_j |A_{ij}|$. This norm is sub-multiplicative, that is, $\|AB\|_\infty \leq \|A\|_\infty \|B\|_\infty$ for matrices A and B. Given a vector $\vec{v}$ with entries indexed by a set S, we denote by $\vec{v}_E$ the vector with entries indexed by the subset $E \subseteq S$, such that $[v_E]_s = v_s$ for all $s \in E$. Similarly, given a matrix A with entries indexed by a set S, we denote by A_E the $|E| \times |E|$ submatrix of A with entries indexed by $E \subseteq S$, such that $[A_E]_{s,t} = A_{s,t}$ for $s, t \in E$. We denote by I_S the $|S| \times |S|$ *identity matrix*. A square matrix A with non-negative entries is *(right) stochastic* if $\sum_t A_{s,t} = 1$ for all rows s of A.

Topology. We recall topological concepts from [114]. A subset $X \subseteq \mathbb{R}^n$ is *open* if, for any $\vec{x} \in X$, there is a $\epsilon > 0$ such that, for all $\vec{y} \in \mathbb{R}^n$ with $\|\vec{x} - \vec{y}\| < \epsilon$, we have $\vec{y} \in X$. A set X is *closed* if its complement, $\mathbb{R}^n \backslash X$, is open. The *closure* $\mathsf{cl}(X)$ of a set X is the smallest closed set containing X. A set X is *bounded* if there is a finite $R \in \mathbb{R}$ such that $\|\vec{x} - \vec{y}\| < R$ for all $\vec{x}, \vec{y} \in X$. A set is *compact* if it is closed and bounded. The *interior* of a set X is the largest open set contained in X, and the *boundary* of a set X is the closure of X less its interior. A set $X \subseteq \mathbb{R}^n$ is *convex* if, for all $\vec{x}_1, \vec{x}_2 \in X$, and all $0 \leq \alpha \leq 1$, the *convex combination* $\alpha\vec{x}_1 + (1-\alpha)\vec{x}_2$ is in X. The *convex hull* $\mathsf{conv}(X)$ of a set X is the intersection of all convex sets containing X. Given a set X, its *downward closure* $\mathsf{dwc}(X)$ is $\{\vec{y} \in \mathbb{R}^n \mid \exists \vec{x} \in X \,.\, \vec{y} \leq \vec{x}\}$, and its *upward closure* $\mathsf{upc}(X)$ is $\{\vec{y} \in \mathbb{R}^n \mid \exists \vec{x} \in X \,.\, \vec{x} \leq \vec{y}\}$. Let $\mathsf{C}(X)$ be the set of *extreme points* of $\mathsf{dwc}(X)$ for a closed convex set X. A *half-space* for a given vector $\vec{a} \in \mathbb{R}^n$ and scalar $b \in \mathbb{R}$ is a set $\{\vec{x} \in \mathbb{R}^n \mid \vec{x} \cdot \vec{a} \leq b\}$, and its boundary is the *hyperplane* $\{\vec{x} \in \mathbb{R}^n \mid \vec{x} \cdot \vec{a} = b\}$. A convex *polytope* is an intersection of a finite number of half-spaces. Given a set X, and a scalar α, we denote by $\alpha \times X$ the set $\{\alpha \cdot \vec{x} \mid \vec{x} \in X\}$. Given sets $X, Y \subseteq \mathbb{R}^n$, their *Minkowski sum* is the set $X + Y \stackrel{\text{def}}{=} \{\vec{x} + \vec{y} \mid \vec{x} \in X \wedge \vec{y} \in Y\}$; we also define $X + \vec{x} \stackrel{\text{def}}{=} X + \{\vec{x}\}$. Given a vector $\vec{v} \in \mathbb{R}^n$ and tuple $Y \in (\mathcal{P}(\mathbb{R}^n))^S$, we let $[Y + \vec{y}]_s \stackrel{\text{def}}{=} Y_s + \vec{y}$. We define, for v and $\epsilon \ll v$, the *rounding* $\lfloor v \rfloor_\epsilon \stackrel{\text{def}}{=} \lfloor v/\epsilon \rfloor \cdot \epsilon$.

3.2 Stochastic Models

In this section we review the models under consideration in the thesis.

3.2.1 Discrete-Time Markov Chains

The simplest model we are using in this thesis is a *discrete-time Markov chain* (DTMC). A DTMC models a system as a set of states, each representing a configuration of the system, together with transitions between states, which are determined via probability distributions depending only on the current state. We define DTMCs with *labels*, which we use later in our assume-guarantee framework to synchronise components. Action labels $\mathcal{A}$ are placed on states, and model observable behaviours. We additionally introduce a dedicated label τ, which can be seen as internal: it is not synchronised in the composition, and cannot be used in specifications.

Definition 3.1. *A* discrete-time Markov chain $\mathcal{D}$ *is a tuple* $\langle S, \varsigma, \mathcal{A}, \chi, \Delta \rangle$*, where*

- S *is a nonempty, countable set of* states*;*
- $\varsigma \in \mathsf{D}(S)$ *is an* initial distribution*;*
- $\mathcal{A}$ *is a set of* actions*;*
- $\chi : S \to \mathcal{A} \cup \{\tau\}$ *is a partial* labelling function*; and*
- $\Delta : S \times S \to [0,1]$ *is a* transition function*, such that* $\sum_{t \in S} \Delta(s,t) = 1$ *for all* $s \in S$*.*

We use the initial distribution ς rather than an initial state as it simplifies some definitions below. Define the *successors* of $s \in S$ as $\Delta(s) \stackrel{\text{def}}{=} \{t \in S \mid \Delta(s,t) > 0\}$. If a label is defined for a state s, that is, $\chi(s)$ is defined, then we sometimes write $s = (a, \mu)$, where $a \stackrel{\text{def}}{=} \chi(s)$ and $\mu(t) \stackrel{\text{def}}{=} \Delta(s,t)$ for all $t \in S$. We call such an action-distribution pair (a, μ) a *move*. The behaviours of the system modelled by a DTMC are represented by the executions, or paths, through the DTMC, where a *path* $\lambda = s_0 s_1 s_2 \ldots$ is a (possibly infinite) sequence of states S such that, for all $i \geq 0$, $\Delta(s_i, s_{i+1}) > 0$. We denote the set of finite (infinite) paths of a DTMC $\mathcal{D}$ by $\Omega_{\mathcal{D}}^{\mathsf{fin}}$ ($\Omega_{\mathcal{D}}$). Note that paths do not have to start in the support of the initial distribution ς. A finite path $\lambda \in \Omega_{\mathcal{D}}^{\mathsf{fin}}$ *reaches* a subset of states $T \subseteq S$ if $\mathsf{last}(\lambda) \in T$, and we denote by $\mathsf{F}\, T \stackrel{\text{def}}{=} \{\lambda \in \Omega_{\mathcal{D}}^{\mathsf{fin}} \mid \mathsf{last}(\lambda) \in T\}$ the set of paths reaching T. If there is a path starting at a state s that reaches T, we say that T is *reachable* from s. We use the notion of *traces* in order to reason about the sequences of actions along executions of

the DTMC. Given a path $\lambda = s_0 s_1 s_2 \ldots$, its trace $\mathsf{trace}(\lambda)$ is the sequence of actions $a_0 a_1 \ldots$ along λ with τ projected out. Formally, we let $a_0 a_1 a_2 \stackrel{\text{def}}{=} \chi(s_{i_0})\chi(s_{i_1})\chi(s_{i_2})\ldots$ along λ, where i_0 is the least index such that $\chi(s_{i_0})$ is defined and, for all $j \geq 0$, i_{j+1} is the least index such that $\chi(s_{i_{j+1}})$ is defined, but $\chi(s_k)$ is not defined for all $i_j < k < i_{j+1}$. Then $\mathsf{trace}(\lambda) \stackrel{\text{def}}{=} \mathsf{proj}_{\{\tau\}}(a_0 a_1 \ldots)$, where, for $\alpha \subseteq \mathcal{A} \cup \{\tau\}$, proj_α is the morphism defined by $\mathsf{proj}_\alpha(a) = a$ if $a \notin \alpha$, and ϵ (the empty trace) otherwise. A trace $w \in \mathcal{A}^* \cup \mathcal{A}^\omega$ is also called an $\mathcal{A}$-*trace*. Given a finite trace w, $\mathsf{paths}(w)$ denotes the set of minimal finite paths with trace w, that is, $\lambda \in \mathsf{paths}(w)$ if $\mathsf{trace}(\lambda) = w$ and there is no path $\lambda' \neq \lambda$ with $\mathsf{trace}(\lambda') = w$ and λ' being a prefix of λ.

Probability Measures. To reason about the probabilistic behaviour of a DTMC $\mathcal{D}$, we define a probability measure over its infinite paths, using the sigma-algebra generated by the cylinder sets of finite paths $\Omega^{\mathsf{fin}}_{\mathcal{D}}$, where the *cylinder set* of a finite path $\lambda \in \Omega^{\mathsf{fin}}_{\mathcal{D}}$ is the set of infinite paths with prefix λ. Thus, given a finite path $\lambda = s_0 s_1 s_2 \ldots s_k$ of $\mathcal{D}$, and a distribution $\vartheta \in \mathsf{D}(S)$, we define the *path distribution* $\mathbb{P}_{\mathcal{D},\vartheta}(\lambda)$, the measure of its cylinder set, by $\mathbb{P}_{\mathcal{D},\vartheta}(\lambda) \stackrel{\text{def}}{=} \vartheta(s_0)\prod_{l=0}^{k-1}\Delta(s_l, s_{l+1})$. We can use ϑ to evaluate the probability measure starting with an arbitrary initial distribution. If $\vartheta = \varsigma$, we sometimes write just $\mathbb{P}_{\mathcal{D}}$, instead of $\mathbb{P}_{\mathcal{D},\varsigma}$. We also define the *trace distribution* $\tilde{\mathbb{P}}_{\mathcal{D},\vartheta}$, of $\mathcal{D}$ by $\tilde{\mathbb{P}}_{\mathcal{D},\vartheta}(w) \stackrel{\text{def}}{=} \sum_{\lambda \in \mathsf{paths}(w)} \mathbb{P}_{\mathcal{D},\vartheta}(\lambda)$, for traces $w \in \mathcal{A}^*$, and similarly write $\tilde{\mathbb{P}}_{\mathcal{D}} \stackrel{\text{def}}{=} \tilde{\mathbb{P}}_{\mathcal{D},\varsigma}$. The path and trace distributions uniquely extend to infinite paths and traces, respectively, due to Carathéodory's extension theorem (Theorem 1.53 of [79]). The *expectation* $\mathbb{E}_{\mathcal{D},\vartheta}[\rho]$ of a measurable function ρ over infinite paths in a DTMC $\mathcal{D}$ is $\int_{\lambda \in \Omega_{\mathcal{D}}} \rho(\lambda) d\mathbb{P}_{\mathcal{D},\vartheta}(\lambda)$. Given a measurable set $H_{\mathcal{D}} \subseteq \Omega_{\mathcal{D}}$ of paths, we define the *conditional expectation* $\mathbb{E}_{\mathcal{D},\vartheta}[\rho \,|\, H_{\mathcal{D}}]$ by $\int_{\lambda \in H_{\mathcal{D}}} \rho(\lambda) d\mathbb{P}_{\mathcal{D},\vartheta}(\lambda) / \mathbb{P}_{\mathcal{D},\vartheta}(H_{\mathcal{D}})$.

We recall at this point some results from probability theory. First, we state the following immediate consequence of Markov's inequality (Lemma 1.7.1 in [115]).

Lemma 3.1. *Let X be a bounded RV such that $\mathbb{P}(X \geq v) = 1$. Then $\mathbb{E}[X] \geq v$.*

We also make use of the Lebesgue dominated convergence theorem.

Lemma 3.2 (Lemma 8.10.5 in [110])**.** *Let μ be a distribution, let $(f_n)_{n\geq 0}$ be measurable functions such that $f = \lim_{n\to\infty} f_n$ exists, and let g be an integrable function bounding the absolute value of f_n, for all $n \geq 0$. Then $\int f d\mu = \lim_{n\to\infty} \int f_n d\mu$.*

Bottom Strongly Connected Components. A *bottom strongly connected component* (BSCC) of a DTMC $\mathcal{D}$ is a nonempty maximal subset of states $\mathcal{B} \subseteq S$ such that every state in $\mathcal{B}$ is reachable from any other state in $\mathcal{B}$, and no state outside $\mathcal{B}$ is reachable. A state $s \in S$ of a DTMC $\mathcal{D}$ is called *recurrent* if it is in some BSCC

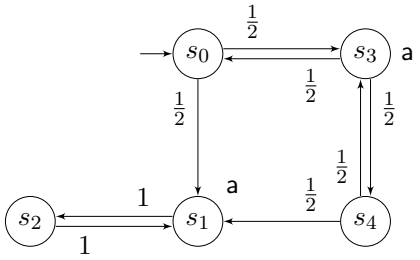


Figure 3.1: Example of a DTMC. The states are shown as circles, labelled by actions, with probabilities annotating the arrows.

$\mathcal{B}$ of $\mathcal{D}$. A state which is not recurrent is called *transient*. A DTMC $\langle S, \varsigma, \mathcal{A}, \chi, \Delta \rangle$ is *irreducible* if S is a BSCC. Conversely, a BSCC endowed with an initial state and a labelling function can be interpreted as an irreducible DTMC. The *stationary distribution* $\mu_{\mathcal{B}} \in \mathsf{D}(S)$ of a BSCC $\mathcal{B}$ is defined as the solution to the equations $\sum_{s \in \mathcal{B}} \mu_{\mathcal{B}}(s) \cdot \Delta(s,t) = \mu_{\mathcal{B}}(t)$ for all $t \in \mathcal{B}$, and is characterised by the following lemma.

Lemma 3.3 (Proposition M.2 in [64])**.** *Given a BSCC $\mathcal{B}$, its stationary distribution $\mu_{\mathcal{B}}$ exists and is unique.*

Example 3.1. *Consider the DTMC shown in Figure 3.1. It consists of five states $S = \{s_0, s_1, s_2, s_3, s_4\}$, and the initial distribution is the Dirac distribution s_0. The transition function is given by $\Delta(s_0, s_1) = \Delta(s_0, s_3) = \frac{1}{2}$, $\Delta(s_1, s_2) = 1$, $\Delta(s_2, s_1) = 1$, $\Delta(s_3, s_0) = \Delta(s_3, s_4) = \frac{1}{2}$, and $\Delta(s_4, s_3) = \Delta(s_4, s_1) = \frac{1}{2}$. The action set is $\mathcal{A} = \{\mathsf{a}\}$, and the labelling function is given by $\chi(s_1) = \chi(s_3) = \mathsf{a}$. Note that τ is not in $\mathcal{A}$, and that no labels are assigned to s_0, s_2 and s_4. Since s_1 and s_3 are labelled, we have the move $s_1 = (\mathsf{a}, s_2)$, where s_2 stands for a Dirac μ_1 with $\mu_1(s_2) = 1$, and the move $s_3 = (\mathsf{a}, \mu_3)$, where $\mu_3(s_0) = \mu_3(s_4) = \frac{1}{2}$.*

An example path through the DTMC is $\lambda_1 = s_0(s_1 s_2)^\omega$, and its probability is $\mathbb{P}_{\mathcal{D}}(\lambda_1) = \frac{1}{2}$, which is just the probability of the cylinder set of $s_0 s_1$. Another example path is $\lambda_2 = (s_0 s_3)^\omega$, and its probability is $\mathbb{P}_{\mathcal{D}}(\lambda_2) = \lim_{n \to \infty} (\frac{1}{2})^{2 \times n} = 0$. Yet another example path is $\lambda_3 = (s_1 s_2)^\omega$, which does not start at s_0, but we can evaluate the probability of λ_3 when starting at s_1, namely $\mathbb{P}_{\mathcal{D},s_1}(\lambda_3) = 1$. The traces $\mathsf{trace}(\lambda_1)$, $\mathsf{trace}(\lambda_2)$ and $\mathsf{trace}(\lambda_3)$ all evaluate to a^ω. Starting at s_0, the trace a^ω has probability $\tilde{\mathbb{P}}_{\mathcal{D}}(\mathsf{a}^\omega) = 1$, since all paths from s_0 contain a infinitely often. There is one BSCC in the DTMC, consisting of $\{s_1, s_2\}$. The states $\{s_0 s_3\}$ do not constitute a BSCC, since the transitions from s_0 to s_1 and from s_3 to s_4 have non-zero probability. We note that the BSCC $\{s_1, s_2\}$ is reached with probability one.

3.2.2 Probabilistic Automata

To model systems that exhibit both probabilistic and nondeterministic behaviour, we use *probabilistic automata* (PAs) as introduced by Segala [118]. The state space S of a PA is partitioned into $S_{\bigcirc}$ and $S_{\Box}$, where the successors are chosen according to a probability distribution (as in DTMCs) and nondeterministically, respectively. We mention several interpretations of nondeterminism when modelling systems. Firstly, we can model that the successor can be determined by a controller that we are interested in finding. Secondly, we can model that the successor is determined by an environment, and we are interested in the behaviours of the system under any environment. Finally, we can also model that the successors are determined by some unknown distribution, which we are not willing, or able, to fix at the time the model is analysed. In this thesis we consider nondeterminism in PAs arising from an environment or unknown distributions, and later, in Section 3.2.3, when defining games, return to interpreting nondeterminism as controlled by strategies.

Definition 3.2. *A* probabilistic automaton $\mathcal{M}$ *is a tuple* $\langle S, (S_{\Box}, S_{\bigcirc}), \varsigma, \mathcal{A}, \chi, \Delta \rangle$*, where*

- S *is a nonempty, countable set of* states*, which is partitioned into the* control states $S_{\Box}$ *and the* stochastic states $S_{\bigcirc}$*;*
- $\varsigma \in D(S_{\Box})$ *is an* initial distribution*;*
- $\mathcal{A}$ *is a set of* actions*;*
- $\chi : S_{\bigcirc} \to \mathcal{A} \cup \{\tau\}$ *is a (total)* labelling function*; and*
- $\Delta : S \times S \to [0,1]$ *is a* transition function*, such that* $\Delta(s,t) = 0$ *for all* $s,t \in S_{\Box}$*,* $\Delta(s,t) \in \{0,1\}$ *for all* $s \in S_{\Box}$ *and* $t \in S_{\bigcirc}$*, and* $\sum_{t \in S_{\Box}} \Delta(s,t) = 1$ *for all* $s \in S_{\bigcirc}$*.*

In stochastic states, the transition function assigns probabilities in the same way as in a DTMC. For control states, however, the transition relation merely indicates which successors are available to choose from. Paths through a PA $\mathcal{M}$ are defined as for DTMCs, and are sequences alternating between control states and stochastic states. We denote by $\Omega^{\mathsf{fin}}_{\mathcal{M}}$ ($\Omega_{\mathcal{M}}$) the sets of finite (infinite) paths of $\mathcal{M}$. In order to define a probability measure for a PA, we have to first resolve the nondeterministic choices at control states, which is done by applying a strategy to the PA to induce a DTMC. We defer the discussion of strategies and their application to Section 3.3. A

PA $\langle S, (S_\Box, S_\bigcirc), \varsigma, \mathcal{A}, \chi, \Delta\rangle$ where $|\Delta(s)| = 1$ for all $s \in S_\Box$ can be seen as a DTMC $\langle S, \varsigma, \mathcal{A}, \chi, \Delta\rangle$, where the control states of the PA are seen as stochastic states of the DTMC with Dirac distributions, and the labelling function χ is the partial function assigning the same labels to $S_\bigcirc \subseteq S$ as in the PA, and no labels to $S_\Box \subseteq S$. In the rest of the thesis, if not explicitly stated otherwise, we assume that PAs have no deadlocks, that is, $|\Delta(s)| \geq 1$ for all $s \in S$.

As in DTMCs, we sometimes write a stochastic state $s \in S_\bigcirc$ as a move (a, μ), where $a = \chi(s)$ and $\mu(t) = \Delta(s, t)$ for all $t \in S_\Box$. Since the labelling function of a PA is total, each stochastic state corresponds to a move. If $\Delta(s, (a, \mu)) > 0$, we write $s \xrightarrow{a} \mu$ for the transition labelled by $a \in \mathcal{A} \cup \{\tau\}$, called an *a-transition*. Conversely, if $s \xrightarrow{a} \mu$, we have $\Delta(s, (a, \mu)) > 0$ and $\chi(a, \mu) = a$. A move (a, μ) is *incoming* to a control state t if $\mu(t) > 0$, and is *outgoing* from a control state s if $s \xrightarrow{a} \mu$. An action is *enabled* in a control state s if there is an outgoing a-transition from s.

We note that several moves with the same label might be outgoing at a control state. A *Markov decision process* (MDP) is a special case of a PA, where in each state, all outgoing moves have distinct labels, but PAs allow more flexible development of compositional models due to the relaxation of this requirement. However, since strategies in PAs can select moves, not just actions, properties for PAs and MDPs are equivalent in our case.

End Components. An *end component* (EC) is a substructure of a PA that is closed under the transition relation and strongly connected, and thus analogous to BSCCs for DTMCs [7]. Formally, an EC $\mathcal{E}$ of a PA $\mathcal{M}$ is a pair $(S_\mathcal{E}, \Delta_\mathcal{E})$ with $\emptyset \neq S_\mathcal{E} \subseteq S$ and $\emptyset \neq \Delta_\mathcal{E} \subseteq \Delta$, such that (i) for all $s \in S_\mathcal{E} \cap S_\bigcirc$, $\sum_{t \in S_\Box} \Delta_\mathcal{E}(s, t) = 1$; (ii) for all $s \in S_\mathcal{E}$, $\Delta_\mathcal{E}(s, t) > 0$ only if $t \in S_\mathcal{E}$; and (iii) for all $s, t \in S_\mathcal{E}$, there is a finite path $s_0 s_1 \dots s_l \in \Omega^{\mathsf{fin}}_\mathcal{M}$ within $\mathcal{E}$ (that is, $s_i \in S_\mathcal{E}$ for all $0 \leq i \leq l$), such that $s_0 = s$ and $s_l = t$. An end component is a *maximal end component* (MEC) if it is maximal with respect to the pointwise subset ordering. A PA $\langle S, (S_\Box, S_\bigcirc), \varsigma, \mathcal{A}, \chi, \Delta\rangle$ is *irreducible* if (S, Δ) is a MEC. Conversely, a MEC endowed with an initial distribution and a labelling function can be interpreted as an irreducible PA. States of a PA that are in some MEC are called *recurrent*, otherwise they are called *transient*.

Example 3.2. *Consider the PA $\mathcal{M}$ shown in Figure 3.2. It consists of six states $S = \{s_0, s_1, s_2, s_3, s_4, s_5\}$, which are partitioned into control states $S_\Box = \{s_0, s_2, s_4\}$ and stochastic states $\{s_1, s_3, s_5\}$. The initial distribution is the Dirac distribution s_0, and the transition function is given by $\Delta(s_0, s_1) = \Delta(s_0, s_3) = 1$, $\Delta(s_1, s_2) = 1$, $\Delta(s_2, s_1) = \Delta(s_2, s_5) = 1$, $\Delta(s_3, s_0) = \Delta(s_3, s_4) = \frac{1}{2}$, $\Delta(s_4, s_1) = \Delta(s_4, s_3) = 1$, and*

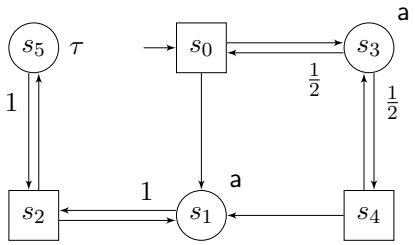


Figure 3.2: Example of a PA. Stochastic states and control states are shown as circles and squares, respectively. Probabilities are attached to the arrows, and labels annotate the stochastic states.

$\Delta(s_5, s_2) = 1$, *and so alternates between control and stochastic states. The action set is* $\mathcal{A} = \{\mathsf{a}\}$, *and the (total) labelling function is given by* $\chi(s_1) = \chi(s_3) = \mathsf{a}$ *and* $\chi(s_5) = \tau$. *Note that from* s_0 *there are two moves with the same action label,* $s_1 = (\mathsf{a}, s_2)$ *and* $s_3 = (\mathsf{a}, \mu_3)$ *with* $\mu_3(s_0) = \mu_3(s_4) = \frac{1}{2}$.

The states s_0, s_2 *and* s_4 *exhibit nondeterminism, which can be resolved by a strategy. The DTMC* $\mathcal{D}$ *in Figure 3.1 can be derived from* $\mathcal{M}$ *by applying a strategy that picks* s_1 *and* s_3 *with uniform probability* $\frac{1}{2}$ *at* s_0 *and* s_4, *and picks* s_1 *with probability one at* s_2. *The probability measure of* $\mathcal{M}$ *under this strategy is precisely the probability measure of* $\mathcal{D}$. *Note that* $\mathcal{D}$ *does not contain all states of* $\mathcal{M}$, *since not all states are reachable under this strategy.* $\mathcal{M}$ *has two maximal end components, namely* $\mathcal{E}_1 = (\{s_0, s_3, s_4\}, \{(s_0, s_3, 1), (s_3, s_0, \frac{1}{2}), (s_3, s_4, \frac{1}{2}), (s_4, s_3, 1)\})$ *and* $\mathcal{E}_2 = (\{s_1, s_2, s_5\}, \{(s_1, s_2, 1), (s_2, s_1, 1), (s_2, s_5, 1), (s_5, s_2, 1)\})$. *We note that* $(\{s_1, s_2\}, \{(s_1, s_2, 1), (s_2, s_2, 1)\})$ *and* $(\{s_2, s_5\}, \{(s_2, s_5, 1), (s_5, s_2, 1)\})$ *are also end components, but not maximal, since they are subsumed by* $\mathcal{E}_2$. *In contrast to BSCCs, end components may have outgoing transitions, as long as these are originating from control states, as is the case in* s_0 *and* s_4 *in* $\mathcal{E}_2$. *The PA* $\mathcal{M}$ *is not irreducible. If we change the initial distribution to the Dirac* $\varsigma = s_1$, $\mathcal{M}$ *becomes irreducible, since it consists entirely of the MEC* $\mathcal{E}_2$ *(under this modification* s_0, s_3 *and* s_4 *become unreachable).*

3.2.3 Stochastic Games

The model of primary interest in this thesis is that of *stochastic games*. A stochastic game, in contrast to a PA, can differentiate between more than one type of nondeterminism, each controlled by a separate player. We are concerned with turn-based two-player games, where at each state, at most one player can make a choice, and where Player 1 represents the controllable part for which we want to synthesise a strategy, while Player 2 represents an uncontrollable environment.

Definition 3.3. *A* turn-based two-player stochastic game $\mathcal{G}$ *(henceforth simply called* game*) is a tuple* $\langle S, (S_\Diamond, S_\Box, S_\bigcirc), \varsigma, \mathcal{A}, \chi, \Delta \rangle$*, where*

- S *is a nonempty, countable set of* states*, which is partitioned into the* Player 1 states $S_\Diamond$*,* Player 2 states $S_\Box$*, and* stochastic states $S_\bigcirc$*;*
- $\varsigma \in D(S_\Diamond \cup S_\Box)$ *is an* initial distribution*;*
- $\mathcal{A}$ *is a set of* actions*;*
- $\chi : S_\bigcirc \to \mathcal{A} \cup \{\tau\}$ *is a (total)* labelling function*; and*
- $\Delta : S \times S \to [0,1]$ *is a* transition function*, such that* $\Delta(s,t) = 0$ *for all* $s, t \in S_\Diamond \cup S_\Box$*,* $\Delta(s,t) \in \{0,1\}$ *for all* $s \in S_\Diamond \cup S_\Box$ *and* $t \in S_\bigcirc$*, and* $\sum_{t \in S_\Diamond \cup S_\Box} \Delta(s,t) = 1$ *for all* $s \in S_\bigcirc$*.*

We call $S_\Diamond \cup S_\Box$ the *player states*. The transition function assigns probabilities at stochastic states, as for DTMCs, and at player states the transition function defines the available choices, as for PAs. However, the Player 1 and Player 2 states are controlled by separate players, and hence the nondeterminism is resolved by separate strategies, as we explain in Section 3.3 below. A game is *non-stochastic* if $\Delta(s,t) \in \{0,1\}$ for all $s, t \in S$. The set of *terminal states* Term contains all states $s \in S$ whose only outgoing transitions are self-loops $s \xrightarrow{a} s$. A game is *stopping* if Term is reached with probability one under any strategy pair.

Paths through a game $\mathcal{G}$ are defined as for PAs, and we denote by $\Omega_\mathcal{G}^{\mathsf{fin}}$ ($\Omega_\mathcal{G}$) the sets of finite (infinite) paths of $\mathcal{G}$. Note that paths of games alternate between player states and stochastic states. All definitions and properties of games carry over to PAs, since a game without Player 1 states, that is, $S_\Diamond = \emptyset$, is equivalent to the PA $\langle S, (S_\Box, S_\bigcirc), \varsigma, \mathcal{A}, \chi, \Delta \rangle$. To enable this equivalence, Definition 3.3 allows a countable state space and a distribution over initial states, but throughout we assume that games with both types of nondeterminism present, that is, $S_\Diamond \neq \emptyset$ and $S_\Box \neq \emptyset$, are finite, and have a single initial state s_{init}, that is, $\varsigma(s_{\mathsf{init}}) = 1$. Assuming a single initial state is without loss of generality, since we can always add an initial state and a single outgoing move with the initial distribution. Further, a game where $|\Delta(s)| = 1$ for all player states $s \in S_\Diamond \cup S_\Box$ can be seen as a DTMC $\langle S, \varsigma, \mathcal{A}, \chi, \Delta \rangle$, where the player states of the game are seen as stochastic states of the DTMC. As for PAs, in the rest of the thesis, if not explicitly stated otherwise, we assume that games have no deadlocks, that is, $|\Delta(s)| \geq 1$ for all $s \in S$.

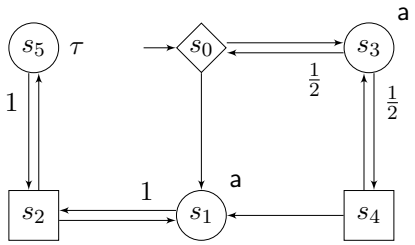


Figure 3.3: Example of a stochastic game. Stochastic states, and states of Player 1 and Player 2, are shown as $\bigcirc$, $\Diamond$ and $\Box$, respectively.

Example 3.3. *Consider the game $\mathcal{G}$ in Figure 3.3. It consists of six states $\{s_0, s_1, s_2, s_3, s_4, s_5\}$, partitioned into Player 1 states $S_\Diamond = \{s_0\}$, Player 2 states $S_\Box = \{s_2, s_4\}$, and stochastic states $S_\bigcirc = \{s_1, s_3, s_5\}$. Note that, apart from this state partition, the game $\mathcal{G}$ is identical to the PA $\mathcal{M}$ in Figure 3.2. The Player 2 states s_2 and s_4 represent uncontrollable nondeterminism, while the Player 1 state s_0 is controllable.*

3.3 Strategies

Nondeterminism in a game $\mathcal{G}$ is resolved by strategies, one for each player; hence, for PAs, a single strategy is sufficient. A strategy maps the states along a path seen so far (the *history*) to a distribution over available moves. We represent a strategy as a state machine with a (possibly infinite) state space $\mathfrak{M}$ representing the memory elements, where transitions between memory elements, triggered by transitions in the game, can be stochastic. This formulation is also known as the stochastic policy graph [96].

Definition 3.4. *A* strategy π *of Player 1 is a tuple $\langle \mathfrak{M}, \pi_c, \pi_u, \pi_d \rangle$, where*

- $\mathfrak{M}$ *is a nonempty, countable set of* memory elements*;*
- $\pi_c : S_\Diamond \times \mathfrak{M} \to D(S)$ *is a* choice function *such that $\pi_c(s, \mathfrak{m})(t) > 0$ only if $t \in \Delta(s)$;*
- $\pi_u : \mathfrak{M} \times S \to D(\mathfrak{M})$ *is a* memory update function*; and*
- $\pi_d : S \to D(\mathfrak{M})$ *is an* initial distribution *on the memory.*

A strategy $\sigma = \langle \mathfrak{N}, \sigma_c, \sigma_u, \sigma_d \rangle$ of Player 2 is defined symmetrically.

As the game proceeds, in a Player 1 state $s \in S_\Diamond$, the next move (a, μ) is picked according to the distribution $\pi_c(s, \mathfrak{m})$, based on the current memory $\mathfrak{m} \in \mathfrak{M}$ of Player 1 at state s. Symmetrically, in Player 2 states $s \in S_\Box$, the next move is picked according

to the distribution $\sigma_{\mathsf{c}}(s, \mathfrak{n})$, based on the memory $\mathfrak{n} \in \mathfrak{N}$ of Player 2 at s. Then both players update their memory based on the current memory and the selected move (a, μ), that is Player 1 picks the next memory $\mathfrak{m}'$ with probability $\pi_{\mathsf{u}}(\mathfrak{m}, (a, \mu))(\mathfrak{m}')$, and symmetrically for Player 2. The successor t of the move (a, μ) is then sampled from μ, and both players again update their memory, that is, Player 1 picks the next memory $\mathfrak{m}''$ with probability $\pi_{\mathsf{u}}(\mathfrak{m}', t)(\mathfrak{m}'')$, and symmetrically for Player 2.

The memory of a strategy is internal to the player owning the strategy, and cannot be observed directly by the other player. In our two-player game setting, Player 1 applies its strategy before Player 2, and so Player 2 can only infer the distribution over the memory of Player 1 given the history, which is observed by both players. The strategy of Player 2 is fixed after Player 1 already fixed a strategy, so the memory of Player 2 is never visible to Player 1. Stochastically updated memory is therefore only beneficial to Player 1; however, in games with more than two players, a player can infer the memory distribution of all other players that pick their strategies beforehand, and so stochastic memory update is beneficial for all but the last player to pick its strategy. We explain more about stochastic memory update in Sections 3.3.1 and 5.1.1.

A strategy is *deterministic (memory) update* (DU) if its memory update functions map to Dirac distributions. DU strategies can be thought of as keeping the prefixes of paths $\Omega_{\mathcal{G}}$ as their memory, and we sometimes write a DU strategy π for Player 1 as a function $\pi : \Omega_{\mathcal{G}}^{\mathsf{fin}} \to \mathsf{D}(S_{\bigcirc})$, where $\pi(\lambda)(t) > 0$ only if $t \in \Delta(\mathsf{last}(s))$ and $\mathsf{last}(s) \in S_{\Diamond}$ (and symmetrically for Player 2). To highlight that memory might not be deterministically updated, we sometimes speak of *stochastic (memory) update* (SU) strategies. If a strategy can be represented without updating memory, that is, $\pi_{\mathsf{u}}(\mathfrak{m}, s) = \mathfrak{m}$ for all $s \in S$, it is called *memoryless*. If the choice functions of a DU strategy are Dirac distributions, that is, $\pi_{\mathsf{c}}(s, \mathfrak{m}) = t$ for some $t \in S$, the strategy is called *deterministic*. A memoryless deterministic strategy is called MD.

3.3.1 Applying Strategies Consecutively

Applying a strategy for a given player in a game resolves the nondeterminism controlled by that player. When applying strategies one after the other, from a two-player game we obtain a PA by applying the Player 1 strategy, and from a PA we obtain a DTMC by applying the Player 2 strategy. Applying a SU strategy π for Player 1 in a game yields a *partially observable* system for Player 2, since the internal memory of the applied Player 1 strategy is hidden from Player 2 [75]. Hence, given a strategy π and a path λ, we let the *belief* $\mathfrak{d}_{\lambda}^{\pi}$ be the distribution of Player 1 memory seen by

Player 2 after λ; formally, we inductively define

$$\mathfrak{d}^\pi_s(\mathfrak{m}) \stackrel{\text{def}}{=} \pi_{\mathsf{d}}(s)(\mathfrak{m}) \qquad \mathfrak{d}^\pi_{\lambda s}(\mathfrak{m}) \stackrel{\text{def}}{=} \sum_{\mathfrak{n}\in\mathfrak{M}} \mathfrak{d}^\pi_\lambda(\mathfrak{n}) \cdot \pi_{\mathsf{u}}(\mathfrak{n}, s)(\mathfrak{m}),$$

for all memory elements $\mathfrak{m} \in \mathfrak{M}$ and all paths $\lambda s \in \Omega^{\mathsf{fin}}_{\mathcal{G}}$ (note that s can be a player state or a move). If the strategy π is clear from context, we write $\mathfrak{d}_\lambda = \mathfrak{d}^\pi_\lambda$. We call the set $\{\mathfrak{d}^\pi_\lambda \mid \lambda \in \Omega^{\mathsf{fin}}_{\mathcal{G}}\}$ of possible distributions over memory elements the *belief space* from π. Note that the belief space can be infinite even if $\mathfrak{M}$ is finite.

The PA $\mathcal{G}^\pi$ induced by a Player 1 strategy π does not contain memory elements but distributions over memory in the states, which are the most Player 2 is allowed to observe about Player 1. The induced PA has states of the form $(s, \mathfrak{d}_\lambda)$ and $(s, (a, \mu), \mathfrak{d}_\lambda)$, where the belief from the strategy is $\mathfrak{d}_\lambda$. A state $(s, \mathfrak{d}_\lambda)$ means that the game is in the Player 2 state s, which remains in control of Player 2 in the induced PA. A state $(s, (a, \mu), \mathfrak{d}_\lambda)$ means that the game is in the Player 1 state s, and the next move, (a, μ) is already decided by the Player 1 strategy. We observe that, if the belief space from the applied strategy is finite, then the induced PA is finite. In order to induce the PA $\mathcal{G}^\pi$, we derive the transition function of the PA from the transition function of the game, which we do by defining the distributions that occur in the moves of $\mathcal{G}^\pi$. Given a move $(a, \mu) \in S_{\bigcirc}$ in $\mathcal{G}$, and a path $\lambda(a, \mu) \in \Omega^{\mathsf{fin}}_{\mathcal{G}}$, we define the distribution $\mu^\pi_{\mathfrak{d}_{\lambda(a,\mu)}}$, which we use below to define the moves $(a, \mu^\pi_{\mathfrak{d}_{\lambda(a,\mu)}})$ in the induced PA $\mathcal{G}^\pi$, by

$$\begin{aligned}
&\mu^\pi_{\mathfrak{d}_{\lambda(a,\mu)}}(t, \quad \mathfrak{d}_{\lambda(a,\mu)t}) \stackrel{\text{def}}{=} \mu(t) && \text{if } t \notin S_\Diamond \\
&\mu^\pi_{\mathfrak{d}_{\lambda(a,\mu)}}(t, (b, \nu), \mathfrak{d}_{\lambda(a,\mu)t}) \stackrel{\text{def}}{=} \mu(t) \cdot \sum_{\mathfrak{m}\in\mathfrak{M}} \mathfrak{d}_{\lambda(a,\mu)t}(\mathfrak{m}) \cdot \pi_{\mathsf{c}}(t, \mathfrak{m})(b, \nu) && \text{if } t \in S_\Diamond,
\end{aligned}$$

for all $t \in S_\Diamond \cup S_\Box$ and $(b, \nu) \in S_{\bigcirc}$. For the initial distribution of the induced PA, we define $\mu^\pi \stackrel{\text{def}}{=} \mu^\pi_{\mathfrak{d}_\epsilon}$ in the same way, where we use the empty path ϵ instead of $\lambda(a, \mu)$.

Definition 3.5. *Let π be a Player 1 strategy, and let $\mathcal{G} = \langle S, (S_\Diamond, S_\Box, S_{\bigcirc}), \varsigma, \mathcal{A}, \chi, \Delta\rangle$ be a game. The* induced PA *$\mathcal{G}^\pi$ is $\langle S', (S'_\Box, S'_{\bigcirc}), \varsigma^\pi, \mathcal{A}, \chi', \Delta'\rangle$, where $S'_\Box \subseteq (S \cup S \times S_{\bigcirc}) \times D(\mathfrak{M})$ is defined inductively as the reachable control states, where χ' and Δ' are such that, for each finite path $\lambda(a, \mu) \in \Omega^{\mathsf{fin}}_{\mathcal{G}}$, we have the transitions*

$$\begin{aligned}
&(\mathit{last}(\lambda), \quad \mathfrak{d}_\lambda) \xrightarrow{a}{}' \mu^\pi_{\mathfrak{d}_{\lambda(a,\mu)}} \\
&(\mathit{last}(\lambda), (a, \mu), \mathfrak{d}_\lambda) \xrightarrow{a}{}' \mu^\pi_{\mathfrak{d}_{\lambda(a,\mu)}}.
\end{aligned}$$

After applying a Player 1 strategy to a game to induce a PA, we need to apply a Player 2 strategy to the PA to obtain an induced DTMC. Applying a Player 2

strategy σ to a PA $\mathcal{M} = \langle S, (S_\Box, S_\bigcirc), \varsigma, \mathcal{A}, \chi, \Delta\rangle$ is defined symmetrically, namely, we apply σ to the equivalent game $\langle S, (\emptyset, S_\Box, S_\bigcirc), \varsigma, \mathcal{A}, \chi, \Delta\rangle$, yielding an induced PA $\mathcal{M}^\sigma = \langle S', (S'_\Box, S'_\bigcirc), \varsigma^\sigma, \mathcal{A}, \chi', \Delta'\rangle$ with $|\Delta'(s)| = 1$ for all $s \in S'$, and so we get the *induced DTMC* $\langle S', \varsigma^\sigma, \mathcal{A}, \chi', \Delta'\rangle$. If no deadlocks are in $\mathcal{M}$, $|\Delta'(s)| = 1$ holds, and otherwise imposing a fairness condition on σ is sufficient, as we explain below in Section 4.2.3. Note that applying a Player 2 strategy resolves all nondeterminism, and so we can include the actual memory of Player 2 in the states of the DTMC, and do not have to use the belief. Thus, for a Player 2 strategy σ, we assume without loss of generality that the belief $\mathfrak{d}^\sigma_\lambda$ for any path λ is a Dirac for some memory element of σ.

To recover a path for $\mathcal{G}$ from a path in $\mathcal{G}^\pi$, we define the mapping $\mathsf{path}_\mathcal{G} : \Omega^{\mathsf{fin}}_{\mathcal{G}^\pi} \to \Omega^{\mathsf{fin}}_\mathcal{G}$ inductively by

$$\mathsf{path}_\mathcal{G}((s,\mathfrak{d})) \stackrel{\text{def}}{=} s, \qquad \mathsf{path}_\mathcal{G}((s,\mathfrak{d})(a,\mu^\pi_{\mathfrak{d}'})\lambda) \stackrel{\text{def}}{=} s(a,\mu)\mathsf{path}_\mathcal{G}(\lambda),$$
$$\mathsf{path}_\mathcal{G}((s,(a,\mu),\mathfrak{d})) \stackrel{\text{def}}{=} s, \qquad \mathsf{path}_\mathcal{G}((s,(a,\mu),\mathfrak{d})(a,\mu^\pi_{\mathfrak{d}'})\lambda) \stackrel{\text{def}}{=} s(a,\mu)\mathsf{path}_\mathcal{G}(\lambda).$$

To apply strategies of both players one after the other, we need to define the strategy σ' of Player 2 for the PA $\mathcal{G}^\pi$ induced by a Player 1 strategy π, given a strategy σ of Player 2 for the game $\mathcal{G}$: we define $\sigma'(\lambda) = (a, \mu^\pi_{\kappa(a,\mu)})$, where $(a,\mu) = \sigma(\kappa)$, for any path $\lambda \in \Omega^{\mathsf{fin}}_{\mathcal{G}^\pi}$ with $\kappa = \mathsf{path}_\mathcal{G}(\lambda)$ and $\mathsf{last}(\kappa) \in S_\Box$. Then, given a game $\mathcal{G}$ and strategies π and σ, we define the induced DTMC $\mathcal{G}^{\pi,\sigma} \stackrel{\text{def}}{=} (\mathcal{G}^\pi)^{\sigma'}$, where σ' is σ mapped to $\mathcal{G}^\pi$ as defined above. We also define the path mapping for $\lambda \in \mathcal{G}^{\pi,\sigma}$ by $\mathsf{path}_\mathcal{G}(\lambda) = \mathsf{path}_\mathcal{G}(\mathsf{path}_{\mathcal{G}^\pi}(\lambda))$. We then define the probability measure for a PA $\mathcal{M}$ under strategy σ by $\mathbb{P}^\sigma_\mathcal{M}(\lambda) \stackrel{\text{def}}{=} \sum\{\mathbb{P}_{\mathcal{M}^\sigma}(\lambda') \mid \mathsf{path}_\mathcal{M}(\lambda') = \lambda\}$ for any path $\lambda \in \Omega^{\mathsf{fin}}_\mathcal{M}$, and for a game $\mathcal{G}$ under a strategies π and σ by $\mathbb{P}^{\pi,\sigma}_\mathcal{G}(\lambda) \stackrel{\text{def}}{=} \sum\{\mathbb{P}_{\mathcal{G}^{\pi,\sigma}}(\lambda') \mid \mathsf{path}_\mathcal{G}(\lambda') = \lambda\}$ for any path $\lambda \in \Omega^{\mathsf{fin}}_\mathcal{G}$.

Example 3.4. *We illustrate strategy application in Figure 3.4. Consider the game on the left, which consists of one Player 1 state, one Player 2 state, and two stochastic states. We apply the following Player 1 strategy π: the memory is $\mathfrak{M} = \{\mathfrak{m}_0, \mathfrak{m}_1\}$, the choice function is defined as $\pi_c(s_2, \mathfrak{m}_0) = s_3$ and $\pi_c(s_2, \mathfrak{m}_1) = s_1$; the memory update function is defined as $\pi_u(\mathfrak{m}_0, s_0)(\mathfrak{m}_1) = 1$ (going from s_1 to s_0), $\pi_u(\mathfrak{m}_1, s_0)(\mathfrak{m}_1) = 1$ (going from s_2 to s_0), $\pi_u(\mathfrak{m}_0, s_1)(\mathfrak{m}_0) = \pi_u(\mathfrak{m}_1, s_1)(\mathfrak{m}_0) = 1$ (going from s_0 or s_2 to s_1), $\pi_u(\mathfrak{m}_0, s_2)(\mathfrak{m}_0) = \pi_u(\mathfrak{m}_0, s_2)(\mathfrak{m}_1) = \frac{1}{2}$ (going from s_1 to s_2), $\pi_u(\mathfrak{m}_1, s_2)(\mathfrak{m}_1) = 1$ (going from s_3 to s_2), and $\pi_u(\mathfrak{m}_0, s_3)(\mathfrak{m}_1) = \pi_u(\mathfrak{m}_1, s_3)(\mathfrak{m}_1) = 1$ (going from s_0 or s_2 to s_3); and the initial distribution is $\pi_d(s_0) = \mathfrak{m}_1$ (we only need to give here*

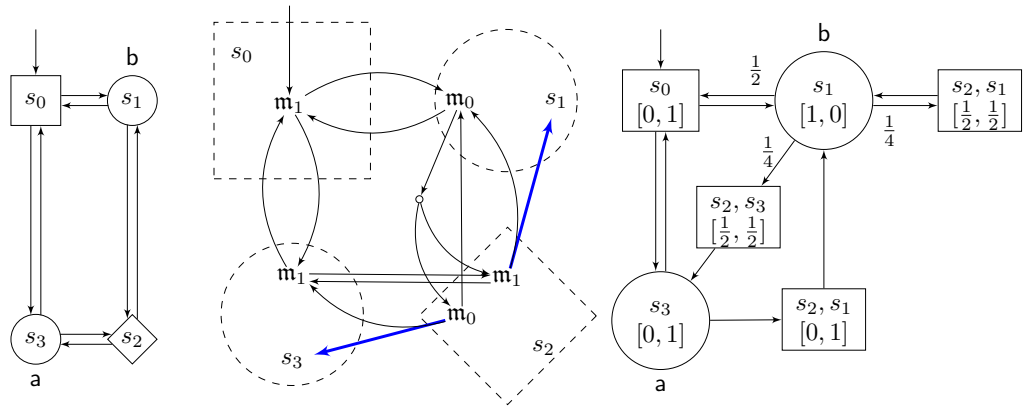


Figure 3.4: Stochastic game $\mathcal{G}$ (left) and PA $\mathcal{G}^\pi$ (right), induced by the strategy π (centre) of Example 3.4. Distributions are uniform if not otherwise indicated. The labels $[p_0, p_1]$ show the belief $\mathfrak{d} \in \mathsf{D}(\mathfrak{M})$, where p_i is the probability of $\mathfrak{m}_i$.

the initial memory for the initial state of the game). The strategy is illustrated in Figure 3.4 (centre). We note that the correspondence between memory updates and the transition function of the game is not always possible, since during the update only the next state and current memory is known. However, we can always augment the strategy by taking $S \times \mathfrak{M}$ as the memory, where the first component is the current state; we show the original states as dashed shapes in the figure. The solid arrows between memory elements represent memory updates, where the distribution is shown as a small unlabelled circle. Note that an update needs to be defined for every outgoing transition of Player 2 and stochastic states, and so there is nondeterminism in the memory updates. For Player 1 states, only memory updates for transitions used by the choice function need to be defined (here both outgoing transitions of s_2 are used). The thick (blue) arrows going from memory elements to states represent the choice function, which is non-stochastic in this case.

The PA $\mathcal{G}^\pi$ induced by the strategy is shown in Figure 3.4 (right). We use the notation $[p_0, p_1]$ for the belief $\mathfrak{d}$ with $p_i = \mathfrak{d}(\mathfrak{m}_i)$ for $i \in \{1, 2\}$. First, note that the belief space from π consists of only three elements, since for any path λ we have $\mathfrak{d}_{\lambda s_0} = \mathfrak{d}_{\lambda s_3} = \mathfrak{d}_{\lambda s_3 s_2} = [0, 1]$, $\mathfrak{d}_{\lambda s_1} = [1, 0]$, and $\mathfrak{d}_{\lambda s_1 s_2} = [\frac{1}{2}, \frac{1}{2}]$. where $[p_0, p_1]$ denotes the distribution assigning probability p_i to $\mathfrak{m}_i$. To determine the transition relation, we explore the reachable states in the induced PA. First, the initial distribution is clearly the Dirac to $(s_0, [0, 1])$. From $(s_0, [0, 1])$, we have two outgoing moves. One

move is $(\mathsf{a}, \mu_{\mathfrak{d}_{\lambda s_1}})$*, where*

$$\mu_{\mathfrak{d}_{\lambda s_1}}(s_0, \overbrace{\mathfrak{d}_{\lambda s_1 s_0}}^{[0,1]}) = \frac{1}{2}, \qquad \mu_{\mathfrak{d}_{\lambda s_1}}(s_2, s_1, \overbrace{\mathfrak{d}_{\lambda s_1 s_2}}^{[\frac{1}{2},\frac{1}{2}]}) = \frac{1}{4}, \qquad \mu_{\mathfrak{d}_{\lambda s_1}}(s_2, s_3, \overbrace{\mathfrak{d}_{\lambda s_1 s_2}}^{[\frac{1}{2},\frac{1}{2}]}) = \frac{1}{4},$$

which we write in the figure as $(s_1, [1, 0])$ *to capture that the memory is* $\mathfrak{d}_{\lambda s_1} = [1, 0]$*. The other move is* $(\mathsf{a}, \mu_{\mathfrak{d}_{\lambda s_3}})$*, where*

$$\mu_{\mathfrak{d}_{\lambda s_3}}(s_0, \overbrace{\mathfrak{d}_{\lambda s_3 s_0}}^{[0,1]}) = \frac{1}{2}, \qquad \mu_{\mathfrak{d}_{\lambda s_3}}(s_2, s_1, \overbrace{\mathfrak{d}_{\lambda s_3 s_2}}^{[0,1]}) = \frac{1}{2},$$

which we write in the figure as $(s_3, [0, 1])$*. The new states accessed from the state* $(s_0, [0, 1])$ *are thus* $(s_2, s_1, [\frac{1}{2}, \frac{1}{2}])$ *and* $(s_2, s_3, [\frac{1}{2}, \frac{1}{2}])$ *via histories ending in* s_1*, and* $(s_2, s_1, [0, 1])$ *via histories ending in* s_3*. In each of these states, there is only one outgoing move, already determined by the strategy, and so* $(s_2, s_1, [\frac{1}{2}, \frac{1}{2}])$ *and* $(s_2, s_1, [0, 1])$ *go to* $(s_1, [1, 0])$*, while* $(s_2, s_1, [\frac{1}{2}, \frac{1}{2}])$ *goes to* $(s_3, [0, 1])$*. Both these states are already added, and so we have added all states reachable from the initial state.*

3.3.2 Applying Strategies Simultaneously

An alternative, equivalent, way of inducing a DTMC from a game when given both strategies is to apply the strategies at the same time, see [40]. We use this formulation in some of our proofs, since, for a pair of finite SU strategies applied to a finite game, this definition yields a finite induced DTMC. Note that, when both strategies are applied at the same time, no player is able to see the other player's memory, and so we do not have to employ the belief to emulate memory hiding.

Definition 3.6. *Given a game* $\mathcal{G} = \langle S, (S_\Diamond, S_\Box, S_\bigcirc), s_{\mathsf{init}}, \mathcal{A}, \chi, \Delta \rangle$*, a strategy pair* (π, σ) *induces a DTMC* $\mathcal{G}^{(\pi,\sigma)} = \langle S', \varsigma', \mathcal{A} \cup S, \chi', \Delta' \rangle$*, where* $S' \subseteq S \times \mathfrak{M} \times \mathfrak{N}$ *is defined as the set of reachable states,* $\varsigma'(s_{\mathsf{init}}, \mathfrak{m}, \mathfrak{n}) = \pi_{\mathsf{d}}(s_{\mathsf{init}})(\mathfrak{m}) \cdot \sigma_{\mathsf{d}}(s_{\mathsf{init}})(\mathfrak{n})$*, and* χ' *and* Δ' *are such that, for each* $s \xrightarrow{a} \mu$ *in* $\mathcal{G}$*, we have both* $(s, \mathfrak{m}, \mathfrak{n}) \xrightarrow{s} \nu'_s$ *and* $((a, \mu), \mathfrak{m}, \mathfrak{n}) \xrightarrow{a} \nu'_{(a,\mu)}$ *in* $\mathcal{G}^{(\pi,\sigma)}$*, where, for all* $\tilde{\mathfrak{m}} \in \mathfrak{M}$*,* $\tilde{\mathfrak{n}} \in \mathfrak{N}$*, and* $t \in S_\Diamond \cup S_\Box$*,*

$$\begin{array}{lll}
\nu'_s & ((a,\mu),\tilde{\mathfrak{m}},\tilde{\mathfrak{n}}) \stackrel{\text{def}}{=} \pi_{\mathsf{c}}(s,\mathfrak{m})(a,\mu) \cdot \pi_{\mathsf{u}}(\mathfrak{m},(a,\mu))(\tilde{\mathfrak{m}}) \cdot \sigma_{\mathsf{u}}(\mathfrak{n},(a,\mu))(\tilde{\mathfrak{n}}) & \text{if } s \in S_\Diamond \\
\nu'_s & ((a,\mu),\tilde{\mathfrak{m}},\tilde{\mathfrak{n}}) \stackrel{\text{def}}{=} \sigma_{\mathsf{c}}(s,\mathfrak{n})(a,\mu) \cdot \pi_{\mathsf{u}}(\mathfrak{m},(a,\mu))(\tilde{\mathfrak{m}}) \cdot \sigma_{\mathsf{u}}(\mathfrak{n},(a,\mu))(\tilde{\mathfrak{n}}) & \text{if } s \in S_\Box \\
\nu'_{(a,\mu)}(t, & \tilde{\mathfrak{m}},\tilde{\mathfrak{n}}) \stackrel{\text{def}}{=} \mu(t) \cdot \pi_{\mathsf{u}}(\mathfrak{m},t)(\tilde{\mathfrak{m}}) \cdot \sigma_{\mathsf{u}}(\mathfrak{n},t)(\tilde{\mathfrak{n}}). &
\end{array}$$

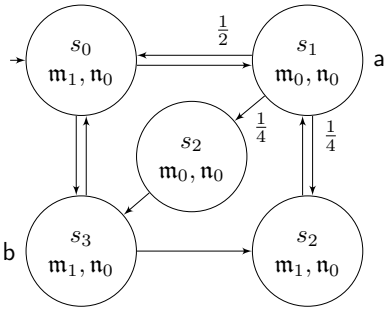


Figure 3.5: The induced DTMC $\mathcal{G}^{(\pi,\sigma)}$ obtained from the game $\mathcal{G}$ from Figure 3.4 by simultaneously applying the Player 1 strategy π (from Figure 3.4) and the Player 2 strategy σ randomising uniformly at s_0.

We define the mapping $\mathsf{path}_{\mathcal{G}} : \Omega^{\mathsf{fin}}_{\mathcal{G}^{(\pi,\sigma)}} \to \Omega^{\mathsf{fin}}_{\mathcal{G}}$ inductively by $\mathsf{path}_{\mathcal{G}}((s,\mathfrak{m},\mathfrak{n})) \stackrel{\text{def}}{=} s$, and $\mathsf{path}_{\mathcal{G}}((s,\mathfrak{m},\mathfrak{n})(a,\nu)\lambda) \stackrel{\text{def}}{=} s\,\mathsf{path}_{\mathcal{G}}(\lambda)$ for $s \in S$. Given a path $\lambda \in \Omega^{\mathsf{fin}}_{\mathcal{G}}$ in a game $\mathcal{G}$, we define $\mathbb{P}^{(\pi,\sigma)}_{\mathcal{G}}(\lambda) \stackrel{\text{def}}{=} \sum\{\mathbb{P}_{\mathcal{G}^{(\pi,\sigma)}}(\lambda') \mid \mathsf{path}_{\mathcal{G}}(\lambda') = \lambda\}$.

Example 3.5. *Consider again, in Figure 3.4, the game $\mathcal{G}$ (left) and corresponding Player 1 strategy π (right). We additionally consider a Player 2 strategy σ that has a single memory element $\mathfrak{n}_0$, and randomises uniformly in s_0. The induced DTMC $\mathcal{G}^{(\pi,\sigma)}$ is shown in Figure 3.5.*

The path distributions of the two ways of inducing DTMCs are equivalent. The proof of the following lemma is technical and given in Appendix A.1.

Lemma 3.4. *Given a game $\mathcal{G}$ with strategies π and σ, it holds that $\mathbb{P}^{\pi,\sigma}_{\mathcal{G}} = \mathbb{P}^{(\pi,\sigma)}_{\mathcal{G}}$.*

3.4 Specifications

To synthesise a strategy for a model, a specification is used to define the desirable behaviours. We consider specifications based on rewards that annotate states. During an execution of a game, rewards are accumulated, and we study properties based on mean-payoffs and ratio rewards, and recall some results for total rewards.

Specifications. A *specification* φ is a predicate on the path distribution of a DTMC. We write $\mathcal{D} \models \varphi$ if $\varphi(\mathbb{P}_{\mathcal{D}})$ holds for a DTMC $\mathcal{D}$, and write $\mathcal{M} \models \varphi$ for a PA $\mathcal{M}$ if, for all σ, $\mathcal{M}^{\sigma} \models \varphi$. We call π *winning* for φ in a game $\mathcal{G}$, if $\mathcal{G}^{\pi} \models \varphi$, and say that φ is *achievable* if such a winning strategy exists, written $\mathcal{G} \models \varphi$. In the assume-guarantee framework, we use specifications defined on traces, since the actions abstract from the states that are specific to the components, and hence can be synchronised. Formally, DTMCs $\mathcal{D}$ and $\mathcal{D}'$, with respective action alphabets $\mathcal{A}_{\mathcal{D}}$ and $\mathcal{A}_{\mathcal{D}'}$, are $\mathcal{A}$-*trace equiv-*

alent if $\mathcal{A} \subseteq \mathcal{A}_{\mathcal{D}} \cap \mathcal{A}_{\mathcal{D}'}$ and $\tilde{\mathbb{P}}_{\mathcal{D}}(w) = \tilde{\mathbb{P}}_{\mathcal{D}'}(w)$ for all $\mathcal{A}$-traces $w \in \mathcal{A}^*$. A specification φ is *defined on $\mathcal{A}$-traces* if $\varphi(\tilde{\mathbb{P}}_{\mathcal{D}}) = \varphi(\tilde{\mathbb{P}}_{\mathcal{D}'})$ for all $\mathcal{A}$-trace equivalent DTMCs $\mathcal{D}$ and $\mathcal{D}'$; if $\mathcal{A}$ is clear from context, we just say φ is defined on traces.

3.4.1 Rewards

We introduce reward structures, and the three types of rewards on paths that we consider in this thesis. A *reward structure* of a game $\mathcal{G}$ with state space S is a function $r : S \to \mathbb{R}$, and we let $r(s) = r(\mathsf{path}_{\mathcal{G}}(s))$ for states s of an induced PA or induced DTMC of $\mathcal{G}$. Note that rewards can be negative. A reward structure r is *defined on* $\mathcal{A}$ if $r(a,\mu) = r(a,\mu')$ for all moves $(a,\mu),(a,\mu') \in S_{\bigcirc}$ such that $a \in \mathcal{A}$, and $r(s) = 0$ otherwise, in which case we sometimes just write $r(a)$ for $r(a,\mu)$. Given reward structures r and r', define the reward structure $r+r'$ by $(r+r')(s) \stackrel{\text{def}}{=} r(s)+r'(s)$ for all $s \in S$; given $v \in \mathbb{R}$, define $r+v$ by $(r+v)(s) \stackrel{\text{def}}{=} r(s)+v$ for all $s \in S$. Given a reward structure r, $N \geq 0$ and a path $\lambda = s_0 s_1 \ldots$, we define $\mathsf{rew}^N(r)(\lambda) \stackrel{\text{def}}{=} \sum_{i=0}^{N} r(s_i)$.

Mean-Payoffs. Mean-payoffs allow one to specify how a system behaves in the long-run. Given a reward structure r, we define the *mean-payoff* (or *average reward*) by $\mathsf{mp}(r)(\lambda) \stackrel{\text{def}}{=} \underline{\lim}_{N\to\infty} \frac{1}{N+1}\mathsf{rew}^N(r)(\lambda)$, where $\underline{\lim}$ is the limit inferior defined by $\underline{\lim}_{N\to\infty} x_N \stackrel{\text{def}}{=} \lim_{N\to\infty}(\inf_{k\geq N} x_k)$ for any sequence $(x_k)_{k\geq 0}$. The limit inferior corresponds naturally to our interpretation of Player 1 operating against an adverse environment in a control scenario. Note that we count both player and stochastic states of a game in the denominator $(N+1)$ of the mean-payoff.

We show below, in Lemma 5.2, that, if we operate with finite DTMCs, the limit inferior can be replaced by the true limit, as it is almost surely defined. We utilise the following characterisation of the long-run behaviour in finite irreducible DTMCs.

Lemma 3.5 (Theorem 4.16 of [87])**.** *Let $\mathcal{D}$ be an irreducible DTMC, where its unique BSCC $\mathcal{B}$ has a stationary distribution $\mu_{\mathcal{B}}$, and let r be a reward structure. The limit $\lim_{N\to\infty} \frac{1}{N+1}\mathsf{rew}^N(r)(\lambda)$ almost surely converges to $\sum_{s\in\mathcal{B}} \mu_{\mathcal{B}}(s)\cdot r(s)$, where $\lambda \in \Omega_{\mathcal{D}}$.*

Remark 3.6. *From Lemma 3.5, the mean-payoff in a BSCC $\mathcal{B}$ is the same at every state $s \in \mathcal{B}$, and we define, for a reward structure r, $\mathsf{mp}(r)(\mathcal{B}) \stackrel{\text{def}}{=} \sum_{s\in\mathcal{B}} r(s)\mu_{\mathcal{B}}(s)$.*

Ratio Rewards. Ratio rewards can express relative performance criteria, such as the fuel consumed by a car per unit distance driven. Given reward structures r and c, we define the *ratio reward* by $\mathsf{ratio}(r/c)(\lambda) \stackrel{\text{def}}{=} \underline{\lim}_{N\to\infty} \mathsf{rew}^N(r)(\lambda)/(1+\mathsf{rew}^N(c)(\lambda))$, where we require that $c(s) \geq 0$ for all $s \in S$, and, for all π and σ, $\mathbb{P}^{\pi,\sigma}_{\mathcal{G}}(\mathsf{mp}(c) > 0) = 1$.

The condition on c can be checked as a single-objective query in a PA, and is satisfied in a game $\mathcal{G}$ precisely if, for every $\mathsf{Player\ 1}$ strategy π, for each EC $(S_\mathcal{E}, \Delta_\mathcal{E})$ of the induced PA $\mathcal{G}^\pi$, there is a state $s \in S_\mathcal{E}$ such that $c(s) > 0$. Ratio rewards $\mathsf{ratio}(r/c)$ generalise average rewards, since, to express $\mathsf{mp}(r)$, we can let $c(s) = 1$ for all $s \in S$.

Total Rewards. Total rewards allow us to model the total payoff received or resources consumed over the time horizon of interest. The *total reward* is defined by $\mathsf{rew}(r)(\lambda) \stackrel{\text{def}}{=} \lim_{N\to\infty} \mathsf{rew}^N(r)(\lambda)$, where we require that either $r(s) \leq 0$ for all $S^\infty_\mathcal{G}$ or $r(s) \leq 0$ for all $S^\infty_\mathcal{G}$, with $S^\infty_\mathcal{G}$ being the set of states occurring infinitely often on any path of $\mathcal{G}$. This condition on r is natural for stopping games, as it is satisfied if $r(s) = 0$ for all $s \in \mathsf{Term}$. Note that the rewards we are considering are undiscounted (that is, it is immaterial how many steps back a reward was acquired).

Transient and Long-Run Behaviour. If we model a control scenario with a distinct start and finish, such as steering a vehicle from one geographical location to another (see for example Section 8.1), we are interested in the transient behaviour of the DTMC induced by the players' strategies. Total rewards and reachability are examples of objectives typically used to specify transient behaviour. On the other hand, scenarios where we are interested in the long-term trend, for instance maintaining the continual safe operation of an aircraft (see Section 8.3), do not have a start or finish, and so we are interested in the recurrent behaviour of the induced DTMC. Average rewards and linear temporal logic (LTL) recurrence properties (of the form "always eventually satisfy a condition") are typically used in this case. Note that ratio rewards as defined above also specify long-run behaviour due to the condition on the denominator c. While temporal logics such as LTL allow one to mix transient and long-run behaviour in the same specification, we are not aware of any work for quantitative specifications towards this direction, which permit combinations of, for example, total and average rewards in one multi-objective specification.

3.4.2 Objectives

An objective is a specification concerned with optimising a one-dimensional reward.

Expectation and Satisfaction Semantics. We consider two types of semantics for achieving an objective. The *satisfaction semantics* requires that the paths whose reward is above a target value together have probability above a threshold. The *expectation semantics* requires that the expectation of the reward over all paths is above a target value. Satisfaction semantics is appropriate if it is sufficient to guarantee a target with a certain probability, and no condition is required with the residual prob-

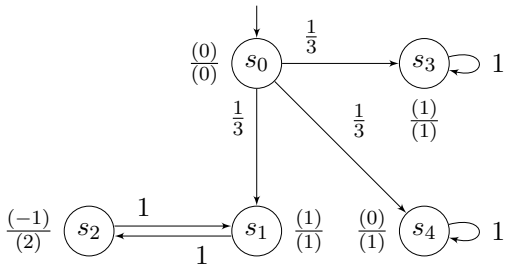


Figure 3.6: Example of objectives for a DTMC. We annotate each state s with $\frac{(r(s))}{(c(s))}$ to indicate that we consider ratios with the reward structures r and c.

ability. Expectation semantics is appropriate if we want to make a statement over all possible executions of a system. An illustration is wanting to position an autonomous vehicle at a certain location: under satisfaction semantics we may require that with probability 0.99 the vehicle is at the location, but with probability 0.01 is may be arbitrarily far away; under expectation semantics we may require the expected position of the vehicle to be at the desired location (and yet the actual position may never be attained). In this thesis, for satisfaction semantics, we only consider probability one, and briefly discuss potential extensions towards arbitrary thresholds in Section 6.5.

Objectives. We consider the following *objectives*, where r and c are reward structures, and $v \in \mathbb{R}$ is a *target*.

Semantics	Reward Type	Symbol	Definition
satisfaction	average	$\mathsf{Pmp}(r)(v)$	$\mathbb{P}_{\mathcal{D}}(\mathsf{mp}(r) \geq v) = 1$
satisfaction	ratio	$\mathsf{Pratio}(r/c)(v)$	$\mathbb{P}_{\mathcal{D}}(\mathsf{ratio}(r/c) \geq v) = 1$
expectation	average	$\mathsf{Emp}(r)(v)$	$\mathbb{E}_{\mathcal{D}}[\mathsf{mp}(r)] \geq v$
expectation	ratio	$\mathsf{Eratio}(r/c)(v)$	$\mathbb{E}_{\mathcal{D}}[\mathsf{ratio}(r/c)] \geq v$
expectation	ratio	$\mathsf{ratioE}(r/c)(v)$	$\mathbb{E}_{\mathcal{D}}[\mathsf{mp}(r)]/\mathbb{E}_{\mathcal{D}}[\mathsf{mp}(c)] \geq v$
expectation	total	$\mathsf{Erew}(r)(v)$	$\mathbb{E}_{\mathcal{D}}[\mathsf{rew}(r)] \geq v$

$\mathsf{Pmp}(-r)(-v)$ means $\mathsf{mp}(r)$ has to be below v almost surely, which we write as $\mathsf{Pmp}^{\leq}(r)(v)$; similarly for $\mathsf{Pratio}^{\leq}$, $\mathsf{Emp}^{\leq}$, $\mathsf{Eratio}^{\leq}$, $\mathsf{ratioE}^{\leq}$ and $\mathsf{Erew}^{\leq}$. Since the condition $\mathsf{mp}(r) \geq v$ is equivalent to $\mathsf{mp}(r - v) \geq 0$, that is, with the rewards r shifted by $-v$, we may assume without loss of generality that mean-payoff objectives have target 0, and we write $\mathsf{Pmp}(r) \stackrel{\text{def}}{=} \mathsf{Pmp}(r)(0)$ and $\mathsf{Emp}(r) \stackrel{\text{def}}{=} \mathsf{Emp}(r)(0)$.

Example 3.6 (Objectives)**.** *Consider the DTMC $\mathcal{D}$ in Figure 3.6, with the two reward structures r and c annotated on each state s as $\frac{(r(s))}{(c(s))}$. Note that $c(s_4) = 1$, in order to fulfil the condition for ratio rewards. We evaluate the mean-payoff using the*

stationary distributions of the BSCCs, which we formally justify in Lemma 5.2 below. The expected mean-payoff is $\mathbb{E}_{\mathcal{D}}[\mathsf{mp}(r)] = \frac{1}{3}\cdot(\frac{1}{2}\cdot(-1)+\frac{1}{2}\cdot 1)+\frac{1}{3}\cdot 0+\frac{1}{3}\cdot 1 = \frac{1}{3}$*. The ratio of expectations is* $\mathbb{E}_{\mathcal{D}}[\mathsf{mp}(r)]/\mathbb{E}_{\mathcal{D}}[\mathsf{mp}(c)] = \frac{1}{3}/(\frac{1}{3}\cdot(\frac{1}{2}\cdot 2+\frac{1}{2}\cdot 1)+\frac{1}{3}\cdot 1+\frac{1}{3}\cdot 1) = \frac{2}{7}$*. The expected total reward* $\mathbb{E}_{\mathcal{D}}[\mathsf{rew}(r)]$ *is not defined, due to the oscillating sequence of rewards* $1,-1,1,\ldots$ *on* s_1 *and* s_2*. The highest mean-payoff* $\mathsf{mp}(r)$ *achievable almost surely is* 0*, since in the BSCCs* $\{s_1,s_2\}$ *and* $\{s_4\}$*, only* 0 *is achievable. Likewise, the highest ratio reward* $\mathsf{ratio}(r/c)$ *achievable almost surely is* 0*: in the BSCC* $\{s_3\}$*, we get* 1*, in the BSCC* $\{s_1,s_2\}$*, we get* $\frac{1}{4}$ *but in the BSCC* $\{s_4\}$*, we get only* 0*. Finally, we can also evaluate the expected ratio reward* $\mathbb{E}_{\mathcal{D}}[\mathsf{ratio}(r/c)] = \frac{1}{3}\cdot(\frac{1}{2}\cdot\frac{-1}{2}+\frac{1}{2}\cdot\frac{1}{1})+\frac{1}{3}\cdot\frac{0}{1}+\frac{1}{3}\cdot\frac{1}{1} = \frac{5}{12}$*, which differs from the ratio of expectations,* $\frac{2}{7}$*.*

ε-Optimality. An objective with target v is *ε-achievable* if, for all $\varepsilon > 0$, it is achievable with target $v-\varepsilon$ by some strategy, and we call such a strategy *ε-optimal*. In an ε-achievable objective, for example of the form $\mathbb{E}[\mathsf{mp}(r)] \geq v$, we can replace the non-strict inequality $\geq$ by a strict inequality $>$, and retain ε-achievability: for example, if $\mathsf{Emp}(r)(v)$ is ε-achievable, then, for any $\varepsilon > 0$, there exists a strategy π achieving $\mathsf{Emp}(r)(v-\frac{\varepsilon}{2})$, and so π achieves $\mathsf{Emp}(r)(v-\varepsilon)$. We hence define $\neg\mathsf{Emp}(r)(v) \stackrel{\text{def}}{=} \mathsf{Emp}^{\leq}(r)(v)$, $\neg\mathsf{Eratio}(r/c)(v) \stackrel{\text{def}}{=} \mathsf{Eratio}^{\leq}(r/c)(v)$, $\neg\mathsf{ratioE}(r/c)(v) \stackrel{\text{def}}{=} \mathsf{ratioE}^{\leq}(r/c)(v)$, and $\neg\mathsf{Erew}(r)(v) \stackrel{\text{def}}{=} \mathsf{Erew}^{\leq}(r)(v)$. However, negating almost sure satisfaction requires positive satisfaction, which we do not consider (see Section 5.6 for a brief discussion).

3.4.3 Multi-Objective Queries

In order to specify trade-offs between objectives, we introduce specifications consisting of several independent objectives. A *multi-objective query* (MQ) φ is a Boolean combination of objectives and its truth value is defined inductively on its syntax. Our focus is on MQs φ with syntax

$$\begin{aligned} & \varphi ::= \psi_{\mathbb{P}} \mid \psi_{\mathbb{E}} \\ \text{satisfaction:}\quad & \psi_{\mathbb{P}} ::= \psi_{\mathbb{P}} \wedge \psi_{\mathbb{P}} \mid \mathsf{Pmp}(r)(v) \mid \mathsf{Pratio}(r/c)(v) \\ \text{expectation:}\quad & \psi_{\mathbb{E}} ::= \psi_{\mathbb{E}} \wedge \psi_{\mathbb{E}} \mid \neg\psi_{\mathbb{E}} \mid \mathsf{Emp}(r)(v) \mid \mathsf{ratioE}(r/c)(v) \mid \mathsf{Erew}(r)(v), \end{aligned}$$

where r and c are reward structures, and $v \in \mathbb{R}$ is a target, and where we allow each MQ to contain objectives only of one kind. The semantics of an MQ is defined on its structure using the canonical interpretation of the logical operators $\wedge$ (conjunction) and $\neg$ (negation). We also use the standard definitions for the operators $\vee$ (disjunction) and $\rightarrow$ (implication) for $\psi_{\mathbb{E}}$. Thus, an MQ can be a conjunction of objectives,

called a *conjunctive query* (CQ), or a general Boolean combination of expectation objectives. Given an MQ φ with targets $v_1, v_2, \ldots$, we denote by $\varphi[\vec{x}]$ the MQ φ, where, for all i, the target v_i is replaced by x_i. For CQs we sometimes use vector notation, for example, we write $\mathsf{Pmp}(\vec{r})(\vec{v})$ for $\bigwedge_{i=1}^{n} \mathsf{Pmp}(r_i)(v_i)$. MQs can be converted into conjunctive or disjunctive normal form (CNF and DNF, respectively), and so we can operate on MQs in either form without loss of generality.

Pareto Sets. When interpreting an MQ as a multi-dimensional maximisation, the optimum in all dimensions at the same time may not be achievable, but only certain *trade-offs*. A trade-off is considered optimal if the value in no dimension can be increased without decreasing the remaining dimensions. Only considering maximisations is without loss of generality, since we can invert signs to obtain minimisations.

If clear from context, we write ε for the vector $\vec{\varepsilon} \stackrel{\text{def}}{=} (\varepsilon, \varepsilon, \ldots, \varepsilon)$. Given an MQ φ with n objectives, the vector $\vec{v} \in \mathbb{R}^n$ is a *Pareto vector* if and only if, for all $\varepsilon > 0$, $\varphi[\vec{v} - \varepsilon]$ is achievable by some finite DU strategy, and $\varphi[\vec{v} + \varepsilon]$ is not achievable by any finite DU strategy. The set of all Pareto vectors (the optimal trade-offs) of a specification φ in a game $\mathcal{G}$ is the *Pareto frontier*, and its downward closure is the *Pareto set*, written $\mathsf{Pareto}(\mathcal{G} \,|\, \varphi)$. Pareto sets are closed, but need not be convex (see Example 3.7 below). Any point in the interior of a Pareto set is achievable, but points on the Pareto frontier itself may not be achievable, see for example Figure 3.7 (d). We quantify the quality of an approximation to a Pareto set P by defining, for $\varepsilon > 0$, an ε-approximation of P to be a set of vectors Q such that for any $\vec{q} \in Q$ there is a $\vec{p} \in P$ such that $\|p - q\| \leq \varepsilon$, and for any $\vec{p} \in Q$ there is a $\vec{q} \in P$ such that $\|p - q\| \leq \varepsilon$. Note that the ε-approximation of a Pareto set is not necessarily unique.

Example 3.7 (Pareto Sets)**.** *We show four games with their corresponding specifications and Pareto sets in Figure 3.7. For game (a), we consider the conjunction of expectations* $\mathsf{Emp}(r_1)(v_1) \wedge \mathsf{Emp}(r_2)(v_2)$*, and obtain a convex Pareto set. For game (b), we consider the disjunction of expectations* $\mathsf{Emp}(r_1)(v_1) \vee \mathsf{Emp}(r_2)(v_2)$*, for which the Pareto set is not convex, but its complement is. Consider, for example, the point* $(v_1, v_2) = (x, \frac{1}{2})$ *with* $x \in \mathbb{R}$*: it is achieved by the strategy that picks* s_2 *at* s_0*, and since the second objective is satisfied,* x *can be arbitrary. Below, in Section 5.4, we further show that, the specifications for games (a) and (b) are equivalent, for* ε*-optimality, to* $\mathsf{Pmp}(r_1)(v_1) \wedge \mathsf{Pmp}(r_2)(v_2)$ *and* $\mathsf{Pmp}(r_1)(v_1) \vee \mathsf{Pmp}(r_2)(v_2)$*, respectively.*

Game (c) comes with the CQ $\mathsf{Pmp}(r_1)(v_1) \wedge \mathsf{Pmp}(r_2)(v_2)$*, and the Pareto set is the non-convex set shown below. Note that also its complement is non-convex, and the specification is not equivalent to* $\mathsf{Emp}(r_1)(v_1) \wedge \mathsf{Emp}(r_2)(v_2)$*. Game (d) comes with the*

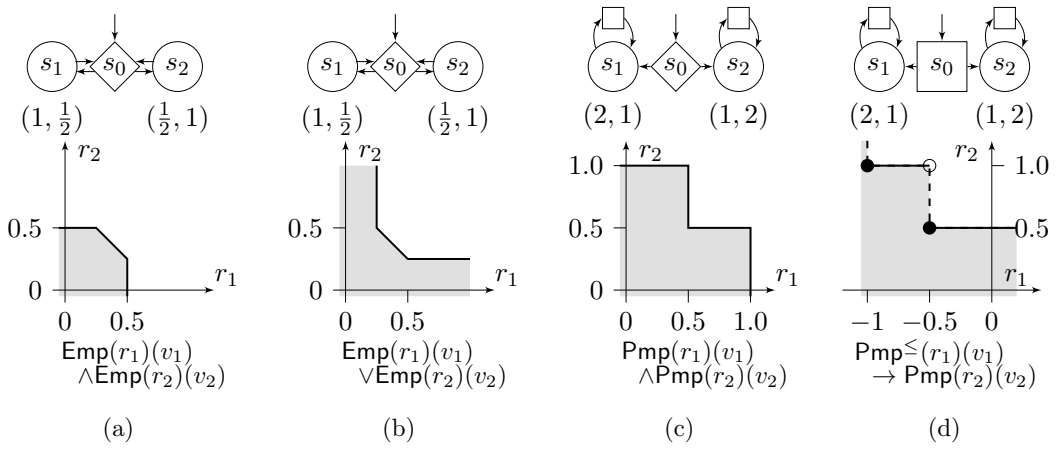


Figure 3.7: Games with corresponding specifications and Pareto sets, shown in the grey shaded areas. Points on the boundary of the Pareto set may not be achievable.

specification $\mathsf{Pmp}^{\leq}(r_1)(v_1) \rightarrow \mathsf{Pmp}(r_2)(v_2)$*, and its Pareto set is shown below. Note that this specification is equivalent to requiring the existence of a strategy* π *such that, for all* σ*,* $\mathbb{P}^{\pi,\sigma}_{\mathcal{G}}(\mathsf{mp}(r_1) > v_1) > 0 \vee \mathbb{P}^{\pi,\sigma}_{\mathcal{G}}(\mathsf{mp}(r_2) \geq v_2) = 1$*; we express the specification in this way here to emphasise that we compute Pareto sets by maximising in several dimensions. The Pareto set is closed by definition, but we indicate in the figure by a dashed line and the hollow circle points that cannot be achieved by finite DU strategies, and yet belong to the Pareto set.*

3.5 Approaches to Synthesis

The main focus of this thesis is on automated strategy synthesis, which we can separate into three problems. Firstly, deciding whether a Player 1 strategy π exists, such that, for all Player 2 strategies σ, a specification φ is satisfied, is called the *achievability problem*. Secondly, constructing a winning strategy, if it exists, is called the *synthesis problem*. Thirdly, computing the set of targets $\vec{v}$ such that a Player 1 strategy π exists that achieves $\varphi[\vec{v}]$ is called the *Pareto set problem*. We treat in this thesis variants of these general problems under various restrictions of memory, and in the context of ε-optimality for strategies. We next review results from [71, 75, 110] for models and specifications related to those we consider in this thesis.

Player 1 Strategies in PAs. Synthesising a strategy in a PA or game achieving a single objective has received considerable attention in the past. A key concern is to

classify the types of strategies that are sufficient for the respective players to achieve an objective. For one-dimensional Emp objectives in PAs, MD strategies suffice.

Lemma 3.7 (Theorem 9.1.8 in [110])**.** *In finite PAs, MD strategies suffice to achieve one-dimensional Emp objectives.*

For reachability in games, we obtain a similar result, as a consequence of the attractor properties of Section 2.1.1 of [75].

Lemma 3.8. *In a game $\mathcal{G}$ with state space S, the set of states from which Player 1 can reach a set $T \subseteq S$ of states almost surely is computable in polynomial time, together with corresponding MD witness strategies.*

Proof. Let $\mathcal{G} = \langle S, (S_\Diamond, S_\Box, S_\bigcirc), \varsigma, \mathcal{A}, \chi, \Delta \rangle$ be a game. Let $T \subseteq S$. Define the attractor $\mathrm{Attr}_\Diamond(A)$ for Player 1 to $A \subseteq S$ as the limit of the non-decreasing sequence of sets $A_0, A_1, \ldots$ defined recursively by $A_0 = A$ and $A_{i+1} = A_i \cup \{s \in S_\Box \mid \Delta(s) \subseteq A_i\} \cup \{s \in S_\bigcirc \cup S_\Diamond \mid \Delta(s) \cap A_i \neq \emptyset\}$; define the attractor $\mathrm{Attr}_\Box(A)$ for Player 2 symmetrically. Note that attractors can be computed in polynomial time. From Section 2.1.1 of [75] we have that $S \setminus \mathrm{Attr}_\Box(S \setminus \mathrm{Attr}_\Diamond(T))$ is the set of states from with Player 1 has a MD strategy to reach T almost surely. □

Determinacy. Given a game $\mathcal{G}$ and a specification φ, a *spoiling strategy* σ for Player 2 is such that, for all Player 1 strategies π, $\mathcal{G}^{\pi,\sigma} \models \neg\varphi$. A game $\mathcal{G}$ with specification φ is *qualitatively determined* if either Player 1 has a winning strategy, or Player 2 has a spoiling strategy. A *tail condition* $\Phi \subseteq \Omega_\mathcal{D}$ is a Borel set of paths (closed under countable unions, countable intersections and relative complement), such that $\forall \kappa \in \Omega_\mathcal{D}^{\mathsf{fin}} . \forall \lambda \in \Omega_\mathcal{D} . \phi \in \Phi \Leftrightarrow \kappa\lambda \in \Phi$, and defines an objective $\mathbb{P}_\mathcal{D}(\Phi) = 1$.

Lemma 3.9 (Theorem 7 of [71])**.** *Stochastic games with tail winning conditions are qualitatively determined.*

In particular, satisfaction semantics for mp and ratio defines tail conditions, and so games are qualitatively determined for one-dimensional Pmp and Pratio objectives.

3.5.1 Dynamic Programming

A popular method for optimising a function is dynamic programming [14], where a *Bellman operator* F characterises the set of optimal achievable targets X^* of an infinite horizon optimisation problem via the *Bellman equation* $X^* = F(X^*)$. The typical setting is that of one-dimensional optimisation, which imposes an ordering in

which any two strategies can be compared. Dynamic programming for objectives that are incomparable in terms of importance requires a multi-dimensional value function. As we are concerned with multi-dimensional queries, the optimal targets are vectors, and X^* characterises the set of such optimal, incomparable, vectors, one for each state in the game. The Bellman equation asserts that X^* is a fixpoint of the operator F. We use concepts from dynamic programming to characterise the strategies achieving Pmp objectives, and thus define background concepts from fixpoint theory.

Fixpoint Theory. We first recall concepts about fixpoints from [52]. A set $\mathcal{C}$ is *(partially) ordered* if it is endowed with an *order* $\preceq \subseteq \mathcal{C} \times \mathcal{C}$ such that, for all $x, y, z \in \mathcal{C}$, (i) $x \preceq y$ (*reflexivity*); (ii) $x \preceq y$ and $y \preceq x$ imply $x = y$ (*antisymmetry*); and (iii) $x \preceq y$ and $y \preceq z$ imply $x \preceq z$ (*transitivity*). Given an ordered set $\mathcal{C}$ and a set $Y \subseteq \mathcal{C}$, an element $x \in \mathcal{C}$ is an *upper bound* of Y if $y \preceq x$ for all $y \in Y$, and the *supremum* of Y is its least upper bound, written $\sup Y$. Given an ordered set $\mathcal{C}$ and a map $\Phi : \mathcal{C} \to \mathcal{C}$, we say that $x \in \mathcal{C}$ is a *fixpoint* of Φ if $\Phi(x) = x$. We write $\mathsf{fix}(\Phi)$ for the least fixpoint of Φ. A nonempty subset D of an ordered set $\mathcal{C}$ is *directed* if, for every finite subset $D' \subseteq D$, an upper bound of D' is in D. An ordered set $\mathcal{C}$ is a *complete partially ordered set* (CPO) if $\sup D$ exists for each directed subset D of $\mathcal{C}$, and if $\mathcal{C}$ has a *bottom element* $\bot$, which is the least element with respect to the order $\preceq$. A map $\Phi : \mathcal{C} \to \mathcal{C}$ over a CPO $\mathcal{C}$ is *(Scott) continuous* if, for every directed set D in $\mathcal{C}$, $\Phi(\sup D) = \sup \Phi(D)$. By Lemma 3.15 of [52], every continuous map is *order-preserving*, meaning that $\Phi(x) \preceq \Phi(y)$ for all $x, y \in \mathcal{C}$ such that $x \preceq y$. We make use of the Kleene fixpoint theorem, characterising the fixpoints of a map.

Lemma 3.10 (Theorem 4.5 (ii) in [52])**.** *Let $\mathcal{C}$ be a CPO, and let $\Phi : \mathcal{C} \to \mathcal{C}$ be a order-preserving map. The least fixpoint $\mathsf{fix}(\Phi)$ exists and is equal to $\sup_{k \geq 0} \Phi^k(\bot)$.*

3.5.2 Multi-Objective Synthesis in PAs

We discuss strategy synthesis in PAs for multi-objective properties of long-run objectives. Since we talk about synthesis in PAs, we use Player 1 strategies instead of Player 2. For PAs, we are interested in evaluating $\exists \pi \,.\, \bigvee_{j=1}^{m} \bigwedge_{i=1}^{n} \varphi_{i,j}$. Clearly, this statement can be analysed by pushing the quantification over π inside the disjunction, and hence getting, for each j, a conjunction $\exists \pi \,.\, \bigwedge_{i=1}^{n} \varphi_{i,j}$. For strategy synthesis in PAs, it is therefore sufficient to consider conjunctions (see also the discussion in [60]).

Weighted Sum Method. Consider the problem of finding a strategy π such that

$$\vec{f}(\pi) \geq \vec{v}, \tag{3.1}$$

where $\vec{f}$ maps strategies π to n-dimensional vectors, and is defined by the PA and the specification (for example, $\vec{f}$ maps a strategy π to $\mathbb{E}^{\pi}_{\mathcal{M}}[\mathsf{mp}(\vec{r})]$), and where $\vec{v} \in \mathbb{R}^n$ is the intended target. One way of finding π in (3.1), and for computing the Pareto set of achievable targets $\vec{v}$, is by repeatedly selecting *weights* $\vec{x} \in \mathbb{R}^n$ and instead solving

$$\max_{\pi} \vec{x} \cdot \vec{f}(\pi) \geq \vec{v}, \tag{3.2}$$

see [93]. This approach can also yield Pareto sets, see Algorithm 4 of [68].

The weighted sum method is justified by observing that, if π is a solution to (3.2), then it is also a solution to (3.1). However, this method is incomplete whenever there are strategies π^1 and π^2 such that $\vec{x} \cdot \vec{f}(\pi^1) = \vec{x} \cdot \vec{f}(\pi^2)$, but $\vec{f}(\pi^1) \neq \vec{f}(\pi^2)$. For example, for Erew objectives, both single- and multi-objective queries in PAs are solved by MD strategies [110], and so the weighted sum method is complete. When applying this method to multi-objective games, we note that MD strategies suffice for single-objective properties, but not for MQs, where memory is required in general, and so the weighted sum method is incomplete in this case. Yet, we can make use of a similar idea when converting MQs to CQs in Section 5.4.

Linear Programming. In PAs, the strategies for both Pmp CQs and Emp CQs can be constructed from the solutions to a linear program (LP), see for example the formulation in [18]. The idea is to split the PA into its MECs, and compute, for each state s, the frequency y_s that s is visited. In the linear program, the constraints maintain that the frequency of entering a state is the same as the frequency of exiting a state, and so the feasible solutions are those in which the frequencies y_s are equivalent to the stationary distribution. An additional constraint $\sum_s y_s \cdot \vec{r}(s) \geq \vec{v}$ ensures the specification is achieved. The strategy can be derived from y_s, giving the probabilities of choosing a move. For Emp CQs, the strategy consists of two phases: first, the strategy plays to reach each MEC with some probability, and then plays within the MECs. Both phases are achieved with memoryless strategies, and a single memory element is sufficient to keep track of which phase the strategy is in. For Pmp CQs, ε-optimal strategies are constructed similarly. The LP characterisation gives rise to a polynomial time algorithm for the achievability problem of Pmp and Emp CQs, as well as for the Pareto set problem for Emp CQs.

Lemma 3.11 (A.3, A.5 and B.3 of [18])**.** *In PAs, the achievability and Pareto set problems for Emp are in P; and the achievability problem for Pmp CQs is in P.*

3.5.3 Multi-Objective Synthesis in Games

In games, algorithms for the analysis of multi-objective queries have to take into account that, in general, finding a spoiler strategy for Player 2 is not the same problem as finding a strategy for Player 2 to achieve the negated objective. For Pmp and Pratio objectives, which are defined via tail conditions, determinacy holds by Lemma 3.9, but for Emp, Eratio, ratioE and Erew MQs stochastic games are not determined in general, see for example [41] for Erew objectives. Non-determinacy is also why sandwich algorithms, which accelerate the computation of the Pareto set by both under- and over-approximating it, cannot be applied, see [113].

Expected Total Rewards. We summarise our previous work on multi-objective synthesis for total expected rewards in stochastic games from [41, 42, 43]. We note at this point that the weighted sum method is incomplete in this case. MD strategies suffice for single-dimensional Erew objectives [64], and so the weights $\vec{x}$ are the same for every state. However, for MQs of Erew objectives, memory is required. Switching between memory elements corresponds to playing with different weights, depending on the history. During synthesis, the amount of memory needed is not known a-priori, and so, in the worst case, for every weight at each predecessor, up to n weights may be required, and this requirement propagates recursively through the game, potentially requiring an infinite number of weights. To compute the Pareto sets for an Erew CQ defined by an n-dimensional reward structure $\vec{r}$, we define in [41] a Bellman operator $F_{\mathsf{rew},\vec{r}}$ over $(\mathcal{P}(\mathbb{R}^n))^{|S|}$, where $S = S_\Diamond \cup S_\Box \cup S_\bigcirc$ is the set of states of the game, by

$$[F_{\mathsf{rew},\vec{r}}(Y)]_s \stackrel{\text{def}}{=} \begin{cases} \mathsf{dwc}(\mathsf{conv}(\bigcup_{t\in\Delta(s)} Y_t) + \vec{r}(s)) & \text{if } s \in S_\Diamond \\ \mathsf{dwc}(\bigcap_{t\in\Delta(s)} Y_t + \vec{r}(s)) & \text{if } s \in S_\Box \\ \mathsf{dwc}(\sum_{t\in\Delta(s)} \Delta(s,t) \times Y_t + \vec{r}(s)) & \text{if } s \in S_\bigcirc. \end{cases}$$

for all $s \in S$. Using the operator $F_{\mathsf{rew},\vec{r}}$, we can compute the sets of targets for Erew conjunctions that Player 1 can achieve after N steps.

Lemma 3.12 (Proposition 2 of [42])**.** *Let $\mathcal{G}$ be a game, and let $\vec{r}$ be an n-dimensional reward structure. For all $k \geq 0$, and all states s of $\mathcal{G}$, $[F^k_{\mathsf{rew},\vec{r}}(\mathsf{dwc}(\vec{0}))]_s = \{\vec{v} \in \mathbb{R}^n \mid \exists\pi\,.\,\forall\sigma\,.\,\mathbb{E}^{\pi,\sigma}_{\mathcal{G},s}[\mathsf{rew}^k(\vec{r})] \geq \vec{v}\}$.*

Under the stopping game assumption, the operator $F_{\mathsf{rew},\vec{r}}$ computes ε-approximations of the Pareto sets for Erew CQs in a bounded number of steps (Theorem 4 of [41]). Note that we do not give a fixpoint characterisation for $F_{\mathsf{rew},\vec{r}}$ in [41], since in the presence of negative rewards the classical subset inclusion order does not make $F_{\mathsf{rew},\vec{r}}$

monotonic in the powerset CPO $(\mathcal{P}(\mathbb{R}^n))^{|S|}$. In [43] we operate on non-negative rewards, obtaining a monotone sequence $(F^k_{\mathsf{rew},\vec{r}}(\mathsf{dwc}(\vec{0})))_{k\geq 0}$, and in Section 7.2.2 we demonstrate that by starting the iteration of $F_{\mathsf{rew},\vec{r}}$ at the lowest possible rewards under any strategies, we obtain a monotone sequence also for negative rewards.

Mean-Payoff in Non-Stochastic Games. We now review previous methods for multi-objective synthesis in non-stochastic games for long-run properties. Strategies for sure-satisfaction of mean-payoffs in non-stochastic games are constructed in [36] by considering *energy objectives*, which are closely related to the total reward. Given a reward structure r and an *initial credit* $c_0 \in \mathbb{R}$, a path λ satisfies an energy objective if $\forall N \geq 0 \,.\, \mathsf{rew}^N(r)(\lambda) + c_0 \geq 0$. For multi-objective rewards $\vec{r}$, Theorem 3 of [26] states that, in a non-stochastic game, there is an initial credit vector for which all paths satisfy the energy objective if and only if $\mathsf{mp}(\vec{r})(\lambda) \geq 0$ for all paths λ. In [36] a Bellman operator is given for non-stochastic games and integer rewards (called *controllable predecessor operator*), which we denote here by $F_{\mathsf{ens},\vec{r},M}$, where $\vec{r}$ is an n-dimensional reward structure defining the objective, and M is a bounded integer. $F_{\mathsf{ens},\vec{r},M}$ is a map between discrete sets $(\{0,1,\ldots,M\}^n)^{|S_\Diamond \cup S_\Box|}$, defined by

$$[F_{\mathsf{ens},\vec{r},M}(X)]_s \stackrel{\text{def}}{=} \begin{cases} \bigcup_{(a,t)\in\Delta(s)} \{\vec{e}_s \mid \exists \vec{e}_t \in X_t \,.\, \vec{e}_s \geq \vec{e}_t - \vec{r}(s,(a,t))\} & \text{if } s \in S_\Diamond \\ \bigcap_{(a,t)\in\Delta(s)} \{e_s \mid \exists \vec{e}_t \in X_t \,.\, \vec{e}_s \geq \vec{e}_t - \vec{r}(s,(a,t))\} & \text{if } s \in S_\Box, \end{cases}$$

for all $s \in S_\Diamond \cup S_\Box$, where $\vec{r}(s,(a,t)) \stackrel{\text{def}}{=} \vec{r}(s) + \vec{r}(a,t)$ is the reward accumulated from taking the move (s,t) from state s. Note that the sets are implicitly upward closed up to M, and contain no vectors with negative elements, even if $\vec{e}_t - \vec{r}(s,(a,t)) \not\geq \vec{0}$: in this case, the negative reward is *truncated*. We return to the concept of truncation in Section 6.2.3, where we define a similar Bellman operator for stochastic games.

3.6 Running Example

In this section we introduce a running example that we refer to throughout the thesis, indicated by the superscript re, as in "Example$^{\mathsf{re}}$ 4.3." Consider a plant producing widgets, with the objective to produce the maximum number of widgets, while minimising the resource requirements We consider a plant and a supervisor that operate in parallel, and communicate with each other over a channel. The communication channel is modelled via synchronisation of actions, and it allows serial communication, arbitrated by a scheduler. The environment of the plant can be considered as,

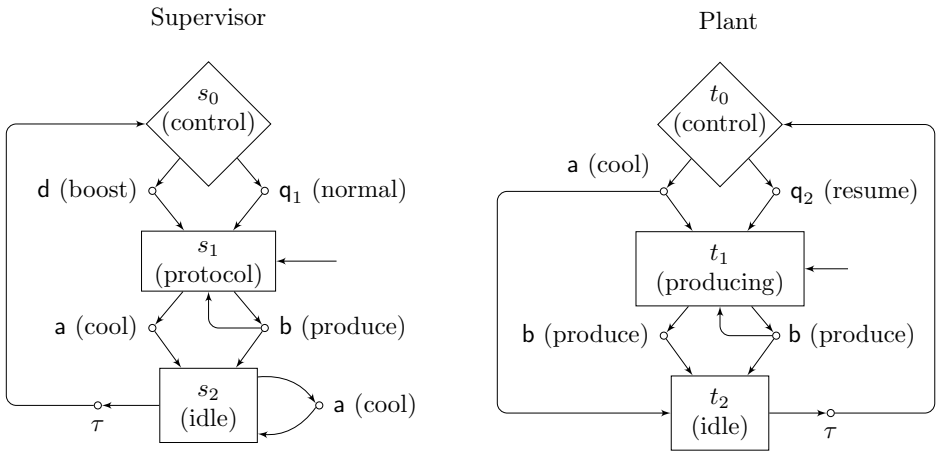


Figure 3.8: Example games $\mathcal{G}^1_{\mathsf{re}}$ (left) and $\mathcal{G}^2_{\mathsf{re}}$ (right). Distributions in stochastic states are uniform. The (ratios of) rewards are $\frac{(1,0,0)}{(1)}$ for a, $\frac{(0,1,1)}{(1)}$ for b, and $\frac{(0,1,0)}{(0)}$ for d.

for example, the quality of the raw materials available. The plant and supervisor are modelled as stochastic games, shown in Figure 3.8, and explained in the following.

We model the plant as $\mathcal{G}^2_{\mathsf{re}}$. In state t_1 the plant is producing widgets, since the action b is enabled. The raw materials might be of imperfect quality, and so, after a widget is produced, the plant may enter the idle state t_2, where additional post-processing on the widget is performed. With some probability, however, the widget is ok, and the plant returns to state t_1, and is ready to produce the next widget. The probability of low-quality raw materials is anywhere between $\frac{1}{2}$ and 1, and so we model this by Player 2 choosing a distribution by randomising between the moves in the t_1 state. Once the t_2 state is entered, there is a τ-transition present in the model to allow the scheduler to arbitrate the communication channel. Once the t_0 state is entered, the plant can either decide to resume widget production using the q_2 action, or it can decide to cool down with some probability, using the a action.

The supervisor is modelled as $\mathcal{G}^1_{\mathsf{re}}$. In the s_1 state, the protocol for widget production is enforced by allowing only certain sequences of cooling and production: cooling is always allowed, but a sequence of producing n widgets is allowed only with probability $\frac{1}{2^n}$, in order to prevent overheating of the plant. The s_2 state has the a action enabled in order to listen on the communication channel, and the outgoing τ-transition allows channel arbitration by the scheduler. Once in the s_0 state, the supervisor can decide to resume production normally using the q_1 action, or boost production by one widget by simply inserting a widget with the d action.

We define reward structures for our running example, based on the properties outlined above. The reward structure c used for ratios of rewards advances time

when cooling and producing, that is, $c(\mathsf{a}) = c(\mathsf{b}) = 1$. We define $\vec{r}$ to express the quantities of interest we want to optimise. To minimise the cooling time, we let $r_1(\mathsf{a}) = 1$; to maximise the number of widgets produced, we let $r_2(\mathsf{b}) = r_2(\mathsf{d}) = 1$; and to minimise the required resources during production, we let $r_3(\mathsf{b}) = 1$. We use $\mathsf{Eratio}(r_1/c)(v_1)$ in the local specifications, in particular, to establish a contract between the components. The global specification for $\mathcal{G}_{\mathsf{re}}$ is

$$\varphi_{\mathsf{re}} = \mathsf{Eratio}(r_2/c)(v_2) \wedge \mathsf{Eratio}^{\leq}(r_3/c)(v_3),$$

meaning that we want to maximise the amounts of widgets produced, and minimise the amount of resources required. Locally, we consider the specifications

$$\varphi^1_{\mathsf{re}} = \mathsf{Eratio}^{\leq}(r_1/c)(v_1) \rightarrow \mathsf{Eratio}(r_2/c)(v_2),$$
$$\varphi^2_{\mathsf{re}} = \mathsf{Eratio}^{\leq}(r_1/c)(v_1) \wedge \mathsf{Eratio}^{\leq}(r_3/c)(v_3),$$

for $\mathcal{G}^1_{\mathsf{re}}$ and $\mathcal{G}^2_{\mathsf{re}}$, respectively, where we use the objective to minimise cooling as a contract between the components.

3.7 Summary

In this chapter we presented the stochastic models that we discuss in this thesis. In particular, we reviewed stochastic games and their behaviour under strategies. Motivated by the need to induce both finite PAs and DTMCs from finite stochastic memory update (SU) strategies applied to games, we gave two definitions of the induced DTMC, which allows us to separate the steps of resolving nondeterminism one player at a time and study related PA problems after a Player 1 strategy has been fixed. Further, we discussed the specifications that we are interested in: we consider specifications consisting of Boolean combinations of objectives, which are defined via mean-payoff and average rewards, both with satisfaction and expectation semantics.

We consider the synthesis problem of strategies for Player 1 to achieve a specification φ_1, and we implicitly assume that Player 2 has the specification $\neg\varphi_1$. This setting is also known as a zero-sum game, since Player 1 wins if and only if Player 2 looses (note that this does not necessarily mean that the players can force a win, since the games may not be determined.) More generally, in a non-zero-sum game, one considers (potentially unrelated) specifications φ_1 and φ_2 for the respective players. A Nash equilibrium in this setting is a strategy profile (π, σ), such that both the

following conditions are satisfied:

$$\begin{aligned} &\text{(NZS1)} \qquad (\exists \pi' \,.\, \mathcal{G}^{\pi',\sigma} \models \varphi_1) \Rightarrow \mathcal{G}^{\pi,\sigma} \models \varphi_1, \quad \text{and} \\ &\text{(NZS2)} \qquad (\exists \sigma' \,.\, \mathcal{G}^{\sigma',\pi} \models \varphi_2) \Rightarrow \mathcal{G}^{\pi,\sigma} \models \varphi_2. \end{aligned}$$

In zero-sum games, where $\varphi_2 = \neg\varphi_1$, the problem reduces to deciding whether either of the following is satisfied:

$$\begin{aligned} &\text{(ZS1)} \qquad \exists \pi \,.\, \forall \sigma \,.\, \mathcal{G}^{\pi,\sigma} \models \varphi_1, \quad \text{or} \\ &\text{(ZS2)} \qquad \exists \sigma \,.\, \forall \pi \,.\, \mathcal{G}^{\pi,\sigma} \models \neg\varphi_1. \end{aligned}$$

We focus on the first query, (ZS1). For expectation objectives, deciding whether an ε-optimal strategy for (ZS1) exists is equivalent to the problem of deciding the second query (ZS2), but the same is not the case for satisfaction objectives.

We moreover discussed existing methods for the synthesis of multi-objective queries in MDPs, as well as non-stochastic and stochastic games. Finally, we introduced a running example in order to illustrate concepts discussed throughout the thesis.

Chapter 4

Assume-Guarantee Framework

Contents

In this chapter we develop the compositional features of our assume-guarantee framework for stochastic games. We discuss how to compose stochastic games, develop composition rules for inferring the global specifications that are achievable in the composed game, and show how to compose strategies so that they satisfy global specifications. Our assume-guarantee framework is enabled by the capability of synthesising strategies for individual components, which we discuss in Chapters 5 and 6.

Our game composition allows for the development of *assume-guarantee rules* for strategy synthesis, according to which winning strategies for individual components can be composed and are winning for the composition, as long as certain side conditions are satisfied. The main idea for obtaining such rules for synthesis is that applying strategies to games yields PAs, and hence any sound *verification* rule for PAs gives rise to a sound synthesis rule for games, under the same side conditions, for example, on action alphabets, and, in some cases, fairness conditions. We consider fairness to be a property guaranteed by the scheduler (that is, Player 2), and hence the synthesis method for individual, monolithic, components does not have to be modified. Furthermore, we show how to compute the Pareto set of target vectors that are achievable compositionally, by composing the Pareto sets computed for individual components. This allows us to instantiate given specifications for the components

with achievable targets, based on the required target of the global specification for the composed game. The compositionally computed Pareto set is, in general, an under-approximation of the Pareto set of the full system computed monolithically, since not all strategies are realisable as composed strategies. This is because our synthesis rules are not complete, even if the underlying PA verification rules are complete.

The game composition that we develop forms the product of the state spaces of the *component games*, and synchronises on shared actions, similarly to the PA composition [118]. However, in the case of the PA composition, no distinction between players is made, which is necessary for a true game composition, to preserve the competitive character between players. We hence augment our game composition with a *compatibility condition*, which reflects an interleaving semantics controlled by Player 2, that is, the environment acts like a scheduler between components. One key feature of our composition is that each Player 1 state $(s^1, s^2, \ldots)$ in the composition there is exactly one *controlling* component $\mathcal{G}^\iota$ such that s^ι is a Player 1 state in $\mathcal{G}^\iota$. Since games synchronise on actions, the local specifications we consider in our framework must allow for the interleaving of paths of one component with the paths of other components. Hence, we consider *specifications defined on traces*, formalised in Section 4.2.1, which only depend on traces over the local action alphabets.

The technical contributions in this chapter are mainly based on our previous work in [10, 11]. In Section 4.1 we introduce our game and strategy composition. In Section 4.2 we develop our synthesis rules by leveraging PA verification rules, and so we first show that winning in composed induced PAs is sufficient to win in the composed game with the composed strategy. To aid the system designer (or *decision maker*) in selecting achievable targets for the local specifications, in Section 4.3 we show how to compute Pareto sets compositionally. Finally, in Section 4.4 we summarise our method, giving a step-by-step recipe for assume-guarantee strategy synthesis.

4.1 Composing Systems

In this section we develop a composition for turn-based two-player stochastic games. Our composition is inspired by the composition of PAs, which can be considered as a special case of our composition, and by interface automata [54], which have a natural interpretation as (concurrent) games. We use superscripts, for example $\mathcal{G}^i$, to denote components and objects specific to the components, for instance, S^i is the state space of $\mathcal{G}^i$. If we want to explicitly indicate that a control state s is from a composed game we use vector notation $\vec{s} \in S_\Box$, and we denote by s^i the ith element

of $\vec{s}$. Furthermore, every probability distribution in the composition is a product distribution $\mu^1 \times \mu^2 \times \cdots$, for which we likewise use vector notation, $\vec{\mu}$. We say that a transition $\vec{s} \xrightarrow{a} \vec{\mu}$ *involves* the ith component if $s^i \xrightarrow{a}{}^i \mu^i$, where the infix operator $\xrightarrow{a}{}^i$ corresponds to the transition function Δ^i of $\mathcal{G}^i$. If a component $\mathcal{G}^i$ is not involved in a transition $\vec{s} \xrightarrow{a} \vec{\mu}$, it is required to remain in the same state, that is, $\mu^i(s^i) = 1$. We denote by I the finite *index set* of component indices. Each component game $\mathcal{G}^i$ is endowed with an alphabet of actions $\mathcal{A}^i$, where synchronisation on *shared actions* in $\bigcap_{i \in I} \mathcal{A}^i$ is viewed as a multi-way blocking communication over ports, as in interface automata, though for simplicity we do not distinguish inputs and outputs (that is, an action can label outgoing transitions of either player).

4.1.1 PA Composition

We start by recalling the composition of PAs [119], and develop our game composition as a generalisation thereof in Section 4.1.2. Given PAs $\mathcal{M}^i$, $i \in I$, with respective state spaces S^i, the set of control states $S_\Box$ of their composition is defined as the Cartesian product $\prod_{i \in I} S^i_\Box$ of the sets of control states $S^i_\Box$, and the moves of the composition are derived from synchronising on actions.

Definition 4.1. *Given PAs* $\mathcal{M}^i = \langle S^i, (S^i_\Box, S^i_\bigcirc), \varsigma^i, \mathcal{A}^i, \chi^i, \Delta^i \rangle$, $i \in I$, *their* composition *is the PA* $\|_{i \in I}\, \mathcal{M}^i \stackrel{\text{def}}{=} \langle S, (\prod_{i \in I} S^i_\Box, S_\bigcirc), \prod_{i \in I} \varsigma^i, \bigcup_{i \in I} \mathcal{A}^i, \chi, \Delta \rangle$, *where* $S_\bigcirc$, χ, *and* Δ *are defined via*

- $\vec{s} \xrightarrow{a} \vec{\mu}$ *for* $a \neq \tau$ *if and only if at least one component is involved and the involved components are exactly those with* a *in their action alphabet; and*
- $\vec{s} \xrightarrow{\tau} \vec{t}$ *if and only if exactly one component* $\mathcal{M}^i$ *is involved.*

In Definition 4.1 we make explicit that we do not synchronise on τ, which can be considered internal, but we could achieve the same effect by renaming τ in component $\mathcal{M}^i$ to τ^i and put τ^i in the actions $\mathcal{A}^i$. Within a PA, a strategy picks the moves outgoing from control states, that is, the strategy choices determine both the action and distribution by picking a stochastic state. However, when composing PAs, synchronisation is only on actions. Thus, from the point of view of a PA $\mathcal{M}^2$ that is composed with a PA $\mathcal{M}^1$, the specific moves chosen in $\mathcal{M}^1$ are hidden, and only the actions are visible, which is in contrast to composing MDPs, where actions uniquely determine the outgoing moves, and allows greater flexibility when modelling systems.

4.1.2 Game Composition

We now define our game composition, which generalises the PA composition, since there are two players and the alternating behaviour of turn-based games must be preserved. Hence, our game composition is also related to that of single threaded interface automata, which preserve alternation [54].

Normal Form. Since our games are turn-based, we have to ensure that it is clear which player controls each state, that is, whose turn it is to play. In particular, we ensure that actions controlled by Player 1 in the components are controlled by Player 1 in the composition. To this end, we transform games into a normal form, which does not affect the achievability of specifications defined on traces.

Definition 4.2. *A game is in* normal form *if every τ-transition $s \xrightarrow{\tau} \mu$ is from a Player 2 state s to a Player 1 state t with a Dirac distribution $\mu = t$; and every Player 1 state s can only be reached by an incoming move (τ, s).*

In particular, every distribution μ of a non-τ-transition, as well as the initial distribution, assigns probability zero to all Player 1 states. Given a game $\mathcal{G}$ without τ-transitions, one can construct its normal form by splitting every state $s \in S_\Diamond$ into a Player 2 state $\overline{s}$ and a Player 1 state $\underline{s}$, such that

- the incoming (outgoing) moves of $\overline{s}$ are precisely the incoming (outgoing) moves of s, where every move (a, μ) is replaced with (a, μ'), where $\mu'(t) = \mu(t)$ for all $t \in S_\Box$ and $\mu(\overline{t}) = \mu(t)$ for all $t \in S_\Diamond$; and
- the only incoming move of $\underline{s}$, and the only outgoing move of $\overline{s}$, is $(\tau, \underline{s})$.

Intuitively, at $\overline{s}$ the game is idle until Player 2 allows Player 1 to choose a move in $\underline{s}$. Hence, any strategy for a game carries over naturally to its normal form, and for specifications defined on traces we can operate without loss of generality with normal-form games. Moreover, a specification φ defined on traces is achievable in a game $\mathcal{G}$ if and only if it is also achievable in the corresponding normal form of $\mathcal{G}$.

Example 4.1 (Normal Form of Games)**.** *Consider the games in Figure 4.1. The two games on the left are transformed into normal form, indicated by the dashed double-arrows. The two games admit the traces $(\mathsf{ab})^\omega$ and $(\mathsf{ac})^\omega$ respectively, and, since τ is projected out, the normal form transformation leaves the traces unaltered.*

Game Composition. We now formally define our game composition. Given games $\mathcal{G}^i$, $i \in I$, in normal form with respective player states $S^i_\Diamond \cup S^i_\Box$, the set of player states

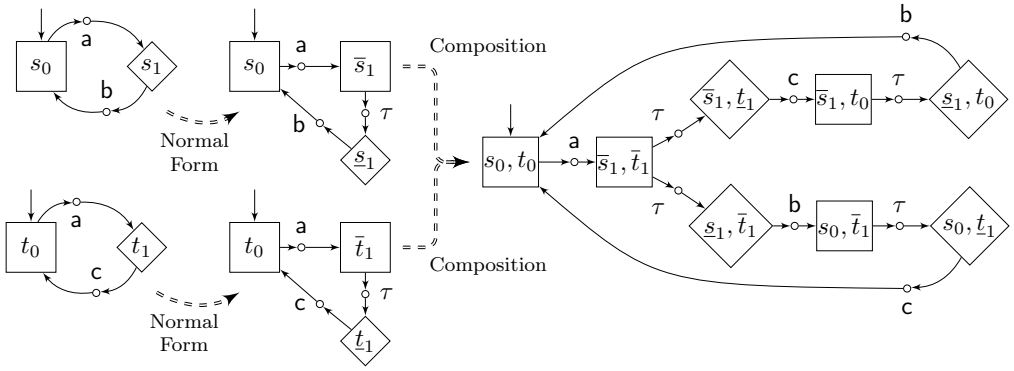


Figure 4.1: Example of normal form and scheduling via τ. Dashed double-arrows show the normal form transformation and game composition operations.

$S_\Diamond \cup S_\Box$ of the composition is a subset of the Cartesian product $\prod_{i\in I} S^i_\Diamond \cup S^i_\Box$. Due to the normal form, each state $\vec{s} \in S_\Diamond \cup S_\Box$ contains either no Player 1 component, in which case $\vec{s}$ is a Player 2 state of the composition, or $\vec{s}$ contains exactly one Player 1 component, in which case $\vec{s}$ is a Player 1 state of the composition. Another consequence of the normal form is that the initial distribution ς of the composed game is such that $\mathsf{supp}(\varsigma) \subseteq S_\Box$. We denote by $\mathsf{En}(s)$ the set of enabled actions in s.

Definition 4.3. *Given normal-form games* $\mathcal{G}^i = \langle S^i, (S^i_\Diamond, S^i_\Box, S^i_\bigcirc), \varsigma^i, \mathcal{A}^i, \chi^i, \Delta^i\rangle$, $i \in I$, *their* composition *is the game* $\|_{i\in I}\, \mathcal{G}^i \stackrel{\text{def}}{=} \langle S, (S_\Diamond, S_\Box, S_\bigcirc), \prod_{i\in I} \varsigma^i, \bigcup_{i\in I} \mathcal{A}^i, \chi, \Delta\rangle$, *where the sets of Player 1 and Player 2 states*

$$S_\Diamond \subseteq \{\vec{s} \in \prod_{i\in I}(S^i_\Diamond \cup S^i_\Box) \mid \exists! \iota \,.\, s^\iota \in S^\iota_\Diamond\} \qquad \text{and} \qquad S_\Box \subseteq \prod_{i\in I} S^i_\Box,$$

are defined to contain the reachable states, where $S_\bigcirc$, χ, *and* Δ *are defined via*

- $\vec{s} \xrightarrow{a} \vec{\mu}$ *for* $a \neq \tau$ *if at least one component is involved and the involved components are exactly those with* a *in their action alphabet, and if* $\vec{s}$ *is a Player 1 state then its only Player 1 component* $\mathcal{G}^\iota$ *is involved; and*
- $\vec{s} \xrightarrow{\tau} \vec{t}$ *if exactly one component* $\mathcal{G}^i$ *is involved,* $\vec{s} \in S_\Box$, *and* $\mathsf{En}(\vec{t}) \neq \emptyset$.

Our game composition is both associative and commutative, formalised in Proposition 4.1 below, facilitating model development. We define the equivalence $\simeq$ such that $\mathcal{G} \simeq \mathcal{G}'$ means that, for all specifications φ defined on traces, $\mathcal{G} \models \varphi$ if and only if $\mathcal{G}' \models \varphi$.

Proposition 4.1. *Given normal-form games $\mathcal{G}^1$, $\mathcal{G}^2$ and $\mathcal{G}^3$, we have*

- $\mathcal{G}^1 \parallel \mathcal{G}^2 \simeq \mathcal{G}^2 \parallel \mathcal{G}^1$ *(commutativity); and*
- $(\mathcal{G}^1 \parallel \mathcal{G}^2) \parallel \mathcal{G}^3 \simeq \mathcal{G}^1 \parallel (\mathcal{G}^2 \parallel \mathcal{G}^3)$ *(associativity).*

Proof. A direct consequence of Definition 4.3. □

The identity of the players must be preserved during composition, to enable synthesis. Thus, moves outgoing from Player 1 states in the individual components are controlled by a single Player 1 in the composition; similarly, the identity of Player 1 in the composition can be seen as the coalition of all the Player 1s in the components. The τ-transitions are controlled by Player 2, and can be considered as a scheduling choice. In the transformation to normal form, at most one such scheduling choice is introduced for each Player 2 state, but in the composition several scheduling choices may be present at a Player 2 state, so that Player 2 resolves nondeterminism arising from concurrency. Hence, Player 2 in the composition acts as a scheduler, controlling which component advances and, in Player 2 states, selecting among available actions, whether synchronised or not.

Example 4.2 (Scheduling by Player 2)**.** *Consider again the games in Figure 4.1. The game on the right is the composition of the two normal form games in the centre. In state $(\overline{s}_1, \overline{t}_1)$, Player 2 picks which game to advance via the two τ-transitions. Hence, the traces are $(\mathsf{a}(\mathsf{bc}|\mathsf{cb}))^\omega$ (using ω-regular expression notation), that is, all interleavings of $(\mathsf{ab})^\omega$ and $(\mathsf{ac})^\omega$ that synchronise on a.*

Our game composition is closely related to PA composition (Definition 4.1), with the added condition that in Player 1 states of the composition the unique Player 1 component must be involved. As PAs are just games without Player 1 states, the game composition applied to PAs is the same as classical PA composition. The condition that the states reached by τ-transitions have outgoing moves ($\mathsf{En}(\vec{t}) \neq \emptyset$) ensures that deadlocks introduced by the normal form transformation are not present in the composed game. Deadlocks that were present before the transformation are still present. In the composition of normal form games, τ-transitions are only enabled in Player 2 states, and Player 1 states are only reached by such transitions; hence, composing normal form games yields a game in normal form.

Example$^{\mathsf{re}}$ 4.3 (Game Composition)**.** *We return to our running example from Section 3.6. The games in Figure 3.8 are reproduced in Figure 4.2, with $\mathcal{G}^1_{re}$ and $\mathcal{G}^2_{re}$*

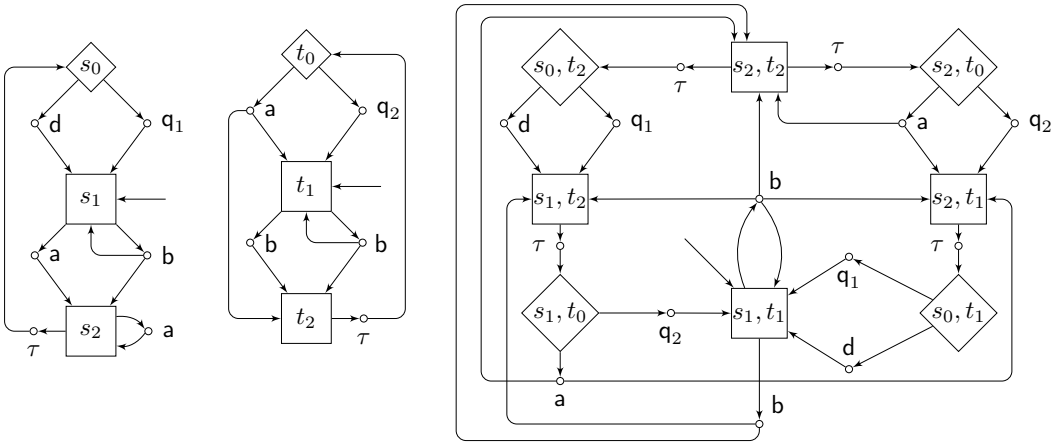


Figure 4.2: Example normal-form games $\mathcal{G}^1_{\mathsf{re}}$ (left) and $\mathcal{G}^2_{\mathsf{re}}$ (centre), with their composition $\mathcal{G}_{\mathsf{re}}$ (right). All distributions are uniform.

on the left and in the centre respectively. Note that $\mathcal{G}^1_{re}$ and $\mathcal{G}^2_{re}$ are already in normal form (the self-loop labelled a *in s_2 indicates that $\mathcal{G}^1_{re}$ was not derived via our automatic normal-form transformation). The game on the right is the composition $\mathcal{G}_{re} = \mathcal{G}^1_{re} \parallel \mathcal{G}^2_{re}$. Actions* a *and* b *are synchronised.* Player 2 *controls* b *in both s_1 and t_1, and so in the composition* Player 2 *controls* b *at (s_1, t_1).* Player 2 *controls* a *in s_1 and s_2, but* Player 1 *controls* a *in t_0, and so it is controlled by* Player 1 *in (s_1, t_0) and (s_2, t_0) in the composition. Note that actions are not necessarily exclusive to* Player 1 *or* Player 2*, since* a *is enabled in $s_1, s_2 \in S^1_\Box$ as well as in $t_0 \in S^2_\Diamond$.*

4.1.3 Compatibility

Composition must not disable or add any Player 1 choice, because strategies that are winning in the components need to make the same choices in the composed game in order for the composition to be winning. Therefore, for assume-guarantee strategy synthesis, we require that moves controlled by Player 1 in one game are enabled and fully controlled by Player 1 in the composition, which is analogous to the composability condition for single-threaded interface automata [54].

Definition 4.4. *Games $(\mathcal{G}^i)_{i\in I}$ are* compatible *if, for every* Player 1 *state $\vec{s} \in S_\Diamond$ in the composition with $s^\iota \in S^\iota_\Diamond$, if $s^\iota \xrightarrow{a}_\iota \mu^\iota$ then there is exactly one distribution $\vec{\nu}$, denoted by $\langle \mu^\iota \rangle_{\vec{s},a}$, such that $\vec{s} \xrightarrow{a} \vec{\nu}$ and $\nu^\iota = \mu^\iota$. (That is, for $i \neq \iota$ such that $a \in \mathcal{A}^i$, there exists exactly one a-transition enabled in s^i.)*

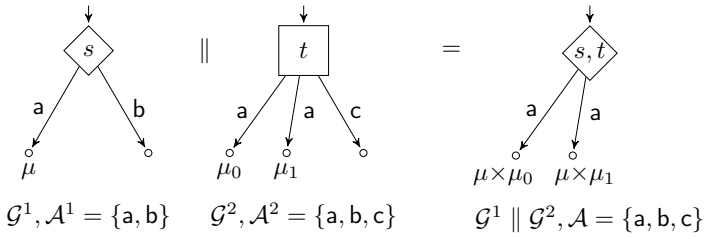


Figure 4.3: Game-fragments to illustrate the effects of incompatibility.

Example 4.4. *Consider the game fragments in Figure 4.3, illustrating the ramifications on the composed strategy if compatibility is violated. The games $\mathcal{G}^1$ and $\mathcal{G}^2$ synchronise on actions a and b. In $\mathcal{G}^1$, the state s has a transition labelled with b, which is blocked in the composition at state (s,t). If a Player 1 strategy synthesised for $\mathcal{G}^1$ needs to pick b in order to win, it is no longer able to do so in the composed game, and so we exclude such cases. Further, in $\mathcal{G}^2$, the state t has two transitions labelled with a that both synchronise with the move (a, μ) at s in $\mathcal{G}^1$. The intuition is that these Player 2 actions in $\mathcal{G}^2$ are actually controlled by Player 1 in $\mathcal{G}^1$. However, if a Player 1 strategy in $\mathcal{G}^1$ picks (a, μ), it is not clear which of the two moves $(\mathsf{a}, \mu \times \mu_0)$ or $(\mathsf{a}, \mu \times \mu_1)$ should be picked in the composition, and hence we prohibit this case in the compatibility condition.*

Checking compatibility requires the construction of the product game, whose state space size is exponential in the number of components. Note, however, that the computational complexity of strategy synthesis typically dominates that of constructing the product game. The explicit construction of the composition can be avoided by requiring that only Player 2 actions are synchronised. We make use of this observation in the case study of Section 8.3. Compatibility is required to ensure that the choices of local winning strategies are available in the composed game. We do not impose the input-enabledness condition of, for example, IO automata [46], as this would give rise to concurrent games.

4.1.4 Strategy Composition

We now show how to compose strategies. We synthesise SU strategies in Chapter 6, justified by their succinct representation, which is why we compose SU strategies so as not to sacrifice this succinctness. The memory update function of the composed SU strategy ensures that the memory in the composition is the same as if the SU strategies were applied to the games individually. According to the definition of strategies,

Definition 3.4, only the *next* state is available to the memory update function, and so, when moving to a state, it can no longer be recovered which components synchronised, unless the memory elements are unique for each state or move. We can, however, always augment the strategy memory $\mathfrak{M}$ by substituting it with $\mathfrak{M} \times S$, so that, for each memory element $(\mathfrak{m}, s)$, $\mathfrak{m}$ corresponds to the original memory, and s corresponds to the *current* state. We assume, therefore, without loss of generality, that we can recover the action a that is part of the current move from the current memory element $\mathfrak{m}$, which we denote by $\mathsf{act}(\mathfrak{m}) \stackrel{\text{def}}{=} a$. We let $\Gamma(a, \vec{t})$ be the set of indices of components that update their memory during an a-transition of the composed game to the state $\vec{s}$, formally defined by

$$\Gamma(a, \vec{t}) \stackrel{\text{def}}{=} \begin{cases} \{i \in I \mid a \in \mathcal{A}^i\} & \text{if } \vec{t} \in S_\Box \text{ or } \vec{t} = (a, \vec{\mu}) \in S_\bigcirc \text{ such that } a \neq \tau \\ \{\iota\} & \text{if } t^\iota \in S^\iota_\Diamond \text{ or } \vec{t} = (\tau, \vec{u}) \in S_\bigcirc \text{ such that } u^\iota \in S^\iota_\Diamond. \end{cases}$$

Thus, if $a \neq \tau$, $\Gamma(a, \vec{t})$ yields the components synchronising on the action a, while on τ-transitions $\Gamma(\tau, \vec{t})$ yields ι, the scheduled component, which can be obtained from $\vec{t}$, the next state, due to the normal form.

Definition 4.5. *The* composition *of Player 1 strategies $\pi^i = \langle \mathfrak{M}^i, \pi^i_{\mathsf{c}}, \pi^i_{\mathsf{u}}, \pi^i_{\mathsf{d}} \rangle$, $i \in I$, for compatible games is $\|_{i \in I}\, \pi^i \stackrel{\text{def}}{=} \langle \prod_{i \in I} \mathfrak{M}^i, \pi_{\mathsf{c}}, \pi_{\mathsf{u}}, \pi_{\mathsf{d}} \rangle$, where we define*

$$\begin{aligned} \pi_{\mathsf{c}}(\vec{s}, \vec{\mathfrak{m}})(a, \langle \mu^\iota \rangle_{\vec{s},a}) &\stackrel{\text{def}}{=} \pi^\iota_{\mathsf{c}}(s^\iota, \mathfrak{m}^\iota)(a, \mu^\iota) && \text{if } s^\iota \in S^\iota_\Diamond \\ \pi_{\mathsf{u}}(\vec{\mathfrak{m}}, \vec{t})(\vec{\mathfrak{n}}) &\stackrel{\text{def}}{=} \textstyle\prod_{i \in \Gamma(\mathsf{act}(\vec{\mathfrak{m}}), \vec{t})} \pi^i_{\mathsf{u}}(\mathfrak{m}^i, t^i)(\mathfrak{n}^i) && \text{if } \mathfrak{m}^i = \mathfrak{n}^i \text{ for } i \neq \Gamma(\mathsf{act}(\vec{\mathfrak{m}}), \vec{t}) \\ \pi_{\mathsf{d}}(\vec{s}) &\stackrel{\text{def}}{=} \textstyle\prod_{i \in I} \pi^i_{\mathsf{d}}(s^i). \end{aligned}$$

From this definition, strategy composition is commutative and associative.

Proposition 4.2. *Given compatible normal-form games $\mathcal{G}^1$, $\mathcal{G}^2$ and $\mathcal{G}^3$, with respective strategies π^1, π^2 and π^3, we have*

- $(\mathcal{G}^1 \parallel \mathcal{G}^2)^{\pi^1 \| \pi^2} \simeq (\mathcal{G}^2 \parallel \mathcal{G}^1)^{\pi^2 \| \pi^1}$ *(commutativity); and*
- $((\mathcal{G}^1 \parallel \mathcal{G}^2) \parallel \mathcal{G}^3)^{(\pi^1 \| \pi^2) \| \pi^3} \simeq (\mathcal{G}^1 \parallel (\mathcal{G}^2 \parallel \mathcal{G}^3))^{\pi^1 \| (\pi^2 \| \pi^3)}$ *(associativity).*

Proof. A direct consequence of Definitions 4.3 and 4.5. □

Example 4.5 (Composing Strategies)**.** *Consider the game fragments in Figure 4.4. The game $\mathcal{G}$ on the right is the composition of the two games $\mathcal{G}^1$ and $\mathcal{G}^2$ on the left.*

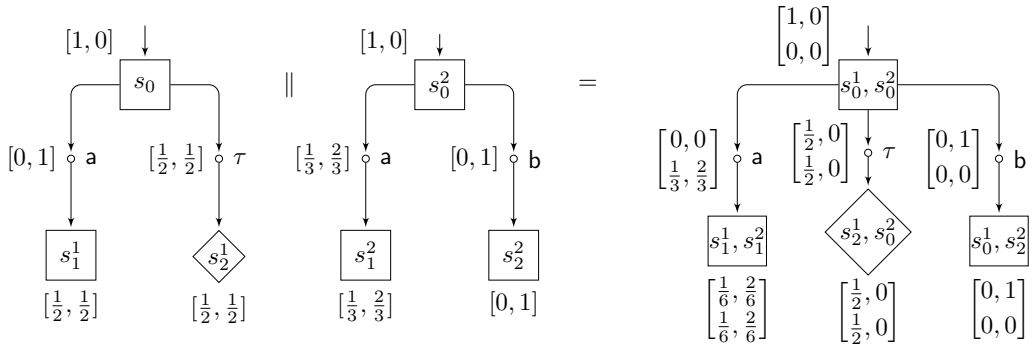


Figure 4.4: Composing SU Strategies. The arrays $[p_0^i, p_1^i]$ show the belief in game $\mathcal{G}^i$, where p_j^i is the probability of $\mathfrak{m}_j^i$. The matrices on the right show the belief from $\pi^1 \parallel \pi^2$, where the entry in the ith row and jth column, $p_{i,j}$ stands for the probability of $(\mathfrak{m}_i^1, \mathfrak{m}_j^2)$.

We define the SU strategies π^i for $\mathcal{G}^i$, with respective memory $\mathfrak{M}^i \stackrel{\text{def}}{=} \{\mathfrak{m}_0^i, \mathfrak{m}_1^i\}$. Let

$$\pi_u^1(\mathfrak{m}_0^1, (\mathsf{a}, s_1^1))(\mathfrak{m}_1^1) = 1, \qquad \pi_u^1(\mathfrak{m}_1^1, (\tau, s_2^1))(\mathfrak{m}_0^1) = \pi_u^1(\mathfrak{m}_1^1, (\tau, s_2^1))(\mathfrak{m}_1^1) = \tfrac{1}{2},$$
$$\pi_u^1(\mathfrak{m}_1^1, s_1^1)(\mathfrak{m}_0^1) = \pi_u^1(\mathfrak{m}_0^1, s_1^1)(\mathfrak{m}_1^1) = \tfrac{1}{2}, \qquad \pi_u^1(\mathfrak{m}_0^1, s_2^1)(\mathfrak{m}_0^1) = \pi_u^1(\mathfrak{m}_1^1, s_2^1)(\mathfrak{m}_1^1) = 1.$$

Supposing that the memory is $\mathfrak{m}_0^1$ in s_0^1, we show how the belief from π^1 evolves in Figure 4.4 (left). The choice function π_c^1 only needs to be specified at s_2^1 in the given game fragment. Further, let

$$\pi_u^2(\mathfrak{m}_0^2, (\mathsf{a}, s_1^2))(\mathfrak{m}_0^2) = \tfrac{1}{3},\ \pi_u^2(\mathfrak{m}_0^2, (\mathsf{a}, s_1^2))(\mathfrak{m}_1^2) = \tfrac{2}{3}, \qquad \pi_u^2(\mathfrak{m}_0^2, (\mathsf{b}, s_2^2))(\mathfrak{m}_1^2) = 1,$$
$$\pi_u^2(\mathfrak{m}_0^2, s_1^1)(\mathfrak{m}_0^2) = \pi_u^2(\mathfrak{m}_1^2, s_1^1)(\mathfrak{m}_1^2) = 1, \qquad \pi_u^2(\mathfrak{m}_1^2, s_2^2)(\mathfrak{m}_1^2) = 1.$$

In Figure 4.4 (centre), we show how the belief from π^2 evolves when starting with $\mathfrak{m}_0^2$ in s_0^2. The memory of the composed strategy $\pi = \pi^1 \parallel \pi^2$ is $\{(\mathfrak{m}_i^1, \mathfrak{m}_j^2) \mid i, j \in \{1, 2\}\}$, corresponding to the product of the individual memories. In Figure 4.4 (right) we show how the belief from π evolves, supposing that the memory is $(\mathfrak{m}_0^1, \mathfrak{m}_0^2)$ at (s_0^1, s_0^2). We show the belief via the matrices $\left[\begin{smallmatrix} p_{0,0}, & p_{0,1} \\ p_{1,0}, & p_{1,1} \end{smallmatrix}\right]$, where $p_{i,j}$ corresponds to the probability that the memory of π is in $(\mathfrak{m}_i^1, \mathfrak{m}_j^2)$. We observe that, on the τ-transition, only the memory of π^1 updates, while on the b-transition only the memory of π^2 updates, which is because $\Gamma(\tau, (s_2^1, s_0^2)) = \{1\}$ and $\Gamma(\mathsf{b}, (s_0^1, s_2^2)) = \{2\}$. On the a-transition, the memory of both local strategies is updated, since $\Gamma(\mathsf{a}, (s_1^1, s_1^2)) = \{1, 2\}$. Finally, we note that the memory of the local strategies can be obtained by marginals: for example, at (s_2^1, s_0^2) the belief from π^1 is given by the row marginals of $\left[\begin{smallmatrix} 0.5, 0 \\ 0.5, 0 \end{smallmatrix}\right]$, which is $[0.5, 0.5]$, and is thus the same as the local belief at s_2^1.

4.2 Assume-Guarantee Rules

In this section we develop assume-guarantee strategy synthesis rules. Synchronisation in Player 1 states means that Player 1 in one component may indirectly control some Player 2 actions in another component. In particular, we can impose local conditions on the components, so that Player 1 of different components can cooperate to achieve a common goal. A typical "contract" between components is to require that in one component Player 1 satisfies the goal B under an assumption A on its environment behaviour (that is, $A \to B$), while Player 1 in the other component ensures that the assumption is satisfied, against all Player 2 strategies.

We suppose in our framework that the designer supplies a game $\mathcal{G} = \|_{i \in I} \mathcal{G}^i$ composed of component games $\mathcal{G}^i$, $i \in I$, together with a *local specification* defined on traces φ^i for each component $\mathcal{G}^i$, and a *global specification* φ for the composed game $\mathcal{G}$. The requirement is that φ can be deduced from φ^i using PA composition rules. Ideally, only the component games and the global specification are required, and the local specifications are deduced automatically. Below, in Section 4.3, we show how by interpreting the targets of an MQ as parameters, we can find the set of achievable targets for the global specification by computing the Pareto sets of the local specifications, and, inversely, by fixing a target for the global specification, we can instantiate the targets of the local specifications.

4.2.1 Specifications Defined on Traces

Our assume-guarantee framework supports a general class of specifications, which is characterised by being invariant under interleaving actions, that is, we operate on specifications defined on traces. We illustrate in Example 4.6 that specifications not defined on traces are in general not supported in our framework. Specifically, Example 4.6 demonstrates that average rewards are not defined on traces in general, since the divisor $(N + 1)$ counts the transitions, irrespective of whether the specification takes them into account. In contrast, ratios of rewards are defined on traces, (as we show below in Proposition 5.4), since we can control which actions are counted in the denominator, and thus, in our game composition, both actions used in the numerator r and denominator c can be synchronised.

Example$^{\text{re}}$ 4.6 (Specifications Defined on Traces)**.** *We continue our running example of Section 3.6. We illustrate how our compositional analysis relies on specifications defined on traces (here we use ratios of rewards), by comparing with specifica-*

tions not defined on traces (here we use mean-payoffs). We investigate the Pareto sets in the component games and compare them with the composed game. We have

$$\mathsf{Pareto}(\mathcal{G}^1_{re} \,|\, \mathsf{Pmp}(r_1)) = \{v_1 \leq v_1^1 \stackrel{\text{def}}{=} 0\}$$
$$\mathsf{Pareto}(\mathcal{G}^2_{re} \,|\, \mathsf{Pmp}(r_1)) = \{v_1 \leq v_1^2 \stackrel{\text{def}}{=} \tfrac{1}{6}\}.$$

In the composed game, we have that $\mathsf{Pareto}(\mathcal{G}^1_{re} \parallel \mathcal{G}^2_{re} \,|\, \mathsf{Pmp}(r_1)) = \{v_1 \leq \frac{1}{12}\}$*, and so* $v_1^1 < \frac{1}{12} < v_1^2$*. However, there is no straightforward relationship between the values achievable for the components and the composition. For example, we have*

$$\mathsf{Pareto}(\mathcal{G}^1_{re} \,|\, \mathsf{Pmp}(r_2)) = \{v_2 \leq v_2^1 \stackrel{\text{def}}{=} 0\}$$
$$\mathsf{Pareto}(\mathcal{G}^2_{re} \,|\, \mathsf{Pmp}(r_2)) = \{v_2 \leq v_2^2 \stackrel{\text{def}}{=} \tfrac{1}{6}\},$$

but we have $\mathsf{Pareto}(\mathcal{G}^1_{re} \parallel \mathcal{G}^2_{re} \,|\, \mathsf{Pmp}(r_2)) = \{v_2 \leq \frac{3}{16}\}$*, where* $v_2^1 < v_2^2 < \frac{3}{16}$*. We can perform the same analysis with ratios of rewards. We obtain*

$$\mathsf{Pareto}(\mathcal{G}^1_{re} \,|\, \mathsf{Pratio}(r_1/c)) = \{v_1 \leq v_1^1 \stackrel{\text{def}}{=} 0\}$$
$$\mathsf{Pareto}(\mathcal{G}^2_{re} \,|\, \mathsf{Pratio}(r_1/c)) = \{v_1 \leq v_1^2 \stackrel{\text{def}}{=} \tfrac{1}{2}\}$$
$$\mathsf{Pareto}(\mathcal{G}^1_{re} \parallel \mathcal{G}^2_{re} \,|\, \mathsf{Pratio}(r_1/c)) = \{v_1 \leq \tfrac{1}{2}\}$$

where $\max\{v_1^1, v_1^2\} \leq \frac{1}{2}$*. Similarly, we obtain*

$$\mathsf{Pareto}(\mathcal{G}^1_{re} \,|\, \mathsf{Pratio}(r_2/c)(v_2)) = \{v_2 \leq v_2^1 \stackrel{\text{def}}{=} 0\}$$
$$\mathsf{Pareto}(\mathcal{G}^2_{re} \,|\, \mathsf{Pratio}(r_2/c)(v_2)) = \{v_2 \leq v_2^2 \stackrel{\text{def}}{=} 1\}$$
$$\mathsf{Pareto}(\mathcal{G}^1_{re} \parallel \mathcal{G}^2_{re} \,|\, \mathsf{Pratio}(r_2/c)) = \{v_2 \leq \tfrac{3}{2}\}$$

where $\max\{v_2^1, v_2^2\} \leq \frac{3}{2}$*. We observe that, for ratio objectives, the Pareto sets of the individual components are subsets of the Pareto sets of the composed game. This is formally shown in Theorem 4.6.*

The traces of a composed system are a subset of the interleavings of the traces of the individual components, and so we can view the composition as modifying the trace distribution of the components. Additionally, note that specifications over multi-dimensional reward structures are defined on traces over the union of the actions of the individual dimensions, and so MQs consisting of objectives defined on traces are likewise defined on traces.

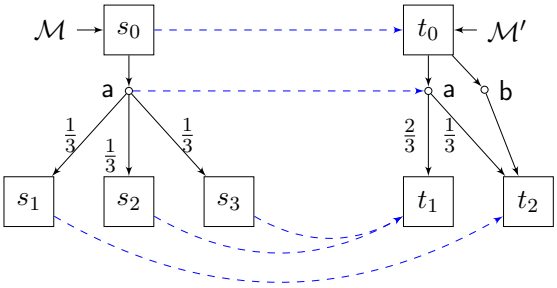


Figure 4.5: Functional simulation from the PA $\mathcal{M}$ to the PA $\mathcal{M}'$, shown as dashed arrows.

Proposition 4.3. *Let $(\varphi_j)_{j\in J}$ be specifications defined on $\mathcal{A}_j$-traces, respectively. Then any MQ φ defined using $(\varphi_j)_{j\in J}$ is defined on $\bigcup_{j\in J}\mathcal{A}_j$-traces.*

Proof. By a straightforward structural induction on the syntax of the MQ φ. □

4.2.2 Properties of the Composition

We now discuss properties of our game composition that allow us to show that synthesising strategies for compatible individual components is sufficient to obtain a composed strategy for the composed game.

Functional Simulations. We introduce functional simulations, which are a special case of classical PA simulations [118], and show that they preserve specifications over traces. Intuitively, a PA $\mathcal{M}'$ functionally simulates a PA $\mathcal{M}$ if all behaviours of $\mathcal{M}$ are present in $\mathcal{M}'$, and if strategies translate from $\mathcal{M}$ to $\mathcal{M}'$. Given a distribution μ, and a partial function $\mathcal{F} : S \to S'$ defined on the support of μ, we write $\overline{\mathcal{F}}(\mu)$ for the distribution defined by $\overline{\mathcal{F}}(\mu)(s') \stackrel{\text{def}}{=} \sum_{\mathcal{F}(s)=s'} \mu(s)$.

Definition 4.6. *A* functional simulation *from a PA $\mathcal{M} = \langle S, (S_\Box, S_\bigcirc), \varsigma, \mathcal{A}, \chi, \Delta\rangle$ to a PA $\mathcal{M}' = \langle S', (S'_\Box, S'_\bigcirc), \varsigma', \mathcal{A}', \chi', \Delta'\rangle$ is a partial function $\mathcal{F} : S \to S'$ such that*

(F1) $\overline{\mathcal{F}}(\varsigma) = \varsigma'$; and

(F2) if $s \xrightarrow{a} \mu$ in $\mathcal{M}$ then $\mathcal{F}(s) \xrightarrow{a}{}' \overline{\mathcal{F}}(\mu)$ in $\mathcal{M}'$.

Example 4.7 (Functional Simulations)**.** *Consider the two PA fragments $\mathcal{M}$ and $\mathcal{M}'$ in Figure 4.5. We define a functional simulation $\mathcal{F}$ from $\mathcal{M}$ to $\mathcal{M}'$ by $\mathcal{F}(s_0) = t_0$, $\mathcal{F}(s_1) = t_2$, and $\mathcal{F}(s_2) = \mathcal{F}(s_3) = t_1$. Consider the transition $s_0 \xrightarrow{a} \mu$ in $\mathcal{M}$, where $\mu(s_1) = \mu(s_2) = \mu(s_3) = \frac{1}{3}$. We have that $\nu = \overline{\mathcal{F}}(\mu)$ is the distribution*

$\nu(t_1) = \frac{2}{3}$ and $\nu(t_2) = \frac{1}{3}$. Since $s_0 \xrightarrow{a} \mu$ in $\mathcal{M}$, we have $t_0 \xrightarrow{a}{}' \overline{\mathcal{F}}(\mu)$ in $\mathcal{M}'$. Further, the initial distribution ς' of $\mathcal{M}'$ is simply the Dirac t_0, which is equivalent to $\overline{\mathcal{F}}(\varsigma)$. Note, however, that the functional simulation is only in one direction, and so the transition $t_0 \xrightarrow{b}{}' t_2$ in $\mathcal{M}'$ has no corresponding transition in $\mathcal{M}$.

If a functional simulation exists from a PA $\mathcal{M}$ to a PA $\mathcal{M}'$ then we can obtain a strategy σ' for $\mathcal{M}'$ from a strategy σ of $\mathcal{M}$. Whenever σ selects a move (a, μ) in state s of $\mathcal{M}$, the functional simulation guarantees that there is a corresponding move $(a, \overline{\mathcal{F}}(\mu))$ in state $\mathcal{F}(s)$ of $\mathcal{M}'$, which σ' can choose. We now formally show that a functional simulation from $\mathcal{M}$ to $\mathcal{M}'$ guarantees that a strategy winning in $\mathcal{M}$ can be mapped to a strategy winning in $\mathcal{M}'$ for specifications defined on traces. The need to considering specifications defined on traces, rather than on paths, is a consequence of the PAs $\mathcal{M}$ and $\mathcal{M}'$ having different state spaces.

Lemma 4.4. *Given a functional simulation from a PA $\mathcal{M}$ to a PA $\mathcal{M}'$ and a specification φ defined on traces, for every (finite) strategy σ there is a (finite) strategy σ' such that $(\mathcal{M}')^{\sigma'} \models \varphi \Leftrightarrow \mathcal{M}^{\sigma} \models \varphi$.*

Proof. Let $\mathcal{M} = \langle S, (S_\Box, S_\bigcirc), \varsigma, \mathcal{A}, \chi, \Delta \rangle$, $\mathcal{M}' = \langle S', (S'_\Box, S'_\bigcirc), \varsigma', \mathcal{A}', \chi', \Delta' \rangle$, and $\sigma = \langle \mathfrak{N}, \sigma_{\mathsf{c}}, \sigma_{\mathsf{u}}, \sigma_{\mathsf{d}} \rangle$. We construct an SU strategy σ' that simulates σ applied to $\mathcal{M}$ by keeping the current state in $\mathcal{M}$ and the memory of σ in its own memory. The functional simulation ensures that every path of $\mathcal{M}^\sigma$ corresponds to a path in $(\mathcal{M}')^{\sigma'}$, and so after seeing memory $(s, \mathfrak{m})$ the strategy σ' picks the next move that σ would pick in state s with memory $\mathfrak{m}$. Our aim is to show that the trace distributions of $(\mathcal{M}')^{\sigma'}$ and $\mathcal{M}^\sigma$ are equivalent. We formally let $\sigma' \stackrel{\text{def}}{=} \langle \mathfrak{N}', \sigma'_{\mathsf{c}}, \sigma'_{\mathsf{u}}, \sigma'_{\mathsf{d}} \rangle$, where we define $\mathfrak{N}' \stackrel{\text{def}}{=} \mathfrak{N} \times S$, and where, for all $(\mathfrak{m}, s), (\mathfrak{n}, (a, \mu)), (\mathfrak{o}, t) \in \mathfrak{N}'$ and all $s' \xrightarrow{a}{}' \mu'$ in $\mathcal{M}'$, such that $s' = \mathcal{F}(s)$, $\mu' = \overline{\mathcal{F}}(\mu)$, $t' = \mathcal{F}(t) \in \mathsf{supp}(\mu')$, we define

$$\sigma'_{\mathsf{d}}(s')((\mathfrak{m}, s)) \stackrel{\text{def}}{=} \sigma_{\mathsf{d}}(s)(\mathfrak{m}) \cdot \frac{\varsigma(s)}{\varsigma'(s')}$$

$$\sigma'_{\mathsf{u}}((\mathfrak{m}, s), (a, \mu'))((\mathfrak{n}, (a, \mu))) \stackrel{\text{def}}{=} \frac{\sigma_{\mathsf{u}}(\mathfrak{m}, (a, \mu))(\mathfrak{n})}{\sigma'_{\mathsf{c}}(s', (\mathfrak{m}, s))(a, \mu')} \tag{4.1}$$

$$\sigma'_{\mathsf{u}}((\mathfrak{n}, (a, \mu)), t')((\mathfrak{o}, t)) \stackrel{\text{def}}{=} \sigma_{\mathsf{u}}(\mathfrak{n}, t)(\mathfrak{o}) \cdot \frac{\mu(t)}{\mu'(t')} \tag{4.2}$$

$$\sigma'_{\mathsf{c}}(s', (\mathfrak{m}, s))(a, \mu') \stackrel{\text{def}}{=} \textstyle\sum_{\overline{\mathcal{F}}(\mu) = \mu'} \sigma_{\mathsf{c}}(s, \mathfrak{m})(a, \mu).$$

Denote by $\mathbb{P}_{\mathcal{D}}(\mathfrak{m}, \lambda) \stackrel{\text{def}}{=} \mathbb{P}_{\mathcal{D}}(\lambda) \cdot \mathfrak{d}_\lambda(\mathfrak{m})$ the probability of the path λ and the memory $\mathfrak{m}$ after seeing λ. A functional simulation $\mathcal{F}$ must be defined for the reachable states of $\mathcal{M}$, and so it extends inductively to a total function on paths of $\mathcal{M}$ by defining

$\mathcal{F}(\lambda(a,\mu)s) \stackrel{\text{def}}{=} \mathcal{F}(\lambda)(a,\overline{\mathcal{F}}(\mu))\mathcal{F}(s)$. We now show by induction on the length of paths that $\mathbb{P}^{\sigma'}_{\mathcal{M}'}((\mathfrak{m},s),\lambda') = \mathbb{P}^{\sigma}_{\mathcal{M}}(\mathfrak{m},\lambda)$ if $\mathcal{F}(\lambda)=\lambda'$, and $\mathbb{P}^{\sigma'}_{\mathcal{M}'}((\mathfrak{m},s),\lambda') = 0$ otherwise.

For the base case, for any $(\mathfrak{m},s) \in \mathfrak{N}'$ and $s' \in S'$ such that $s' = \mathcal{F}(s)$, we have that $\mathbb{P}^{\sigma'}_{\mathcal{M}'}((\mathfrak{m},s),s') = \varsigma'(s') \cdot \sigma'_{\mathsf{d}}(s',(\mathfrak{m},s)) = \sigma_{\mathsf{d}}(s)(\mathfrak{m}) \cdot \varsigma(s) = \mathbb{P}^{\sigma}_{\mathcal{M}}(\mathfrak{m},s)$; if, on the other hand, $s' \neq \mathcal{F}(s)$ then $\sigma'_{\mathsf{d}}(s',(\mathfrak{m},s)) = 0$, and so $\mathbb{P}^{\sigma'}_{\mathcal{M}'}((\mathfrak{m},s),s') = 0$.

For the induction step, assume the equality holds for $\lambda \in \Omega^{\mathsf{fin}}_{\mathcal{M}}$ and $\lambda' \in \Omega^{\mathsf{fin}}_{\mathcal{M}'}$, and we consider paths $\lambda(a,\mu)t \in \Omega^{\mathsf{fin}}_{\mathcal{M}}$ and $\lambda'(a,\mu')t' \in \Omega^{\mathsf{fin}}_{\mathcal{M}'}$. We have that

$$\mathbb{P}^{\sigma'}_{\mathcal{M}'}((\mathfrak{o},t),\lambda'(a,\mu')t') = \sum_{(\mathfrak{m},\mathsf{last}(\lambda)),(\mathfrak{n},(a,\mu))\in\mathfrak{N}'} \mathbb{P}^{\sigma'}_{\mathcal{M}'}((\mathfrak{m},s),\lambda') \cdot p_1 \cdot p_2,$$

where

$$p_1 = \sigma'_{\mathsf{c}}(\mathsf{last}(\lambda'),(\mathfrak{m},\mathsf{last}(\lambda)))(a,\mu') \cdot \sigma'_{\mathsf{u}}((\mathfrak{m},\mathsf{last}(\lambda)),(a,\mu'))((\mathfrak{n},(a,\mu)))$$
$$p_2 = \mu'(t') \cdot \sigma'_{\mathsf{u}}((\mathfrak{n},(a,\mu)),t')((\mathfrak{o},t')).$$

We consider first the case where $\mathcal{F}(\lambda(a,\mu)t) \neq \lambda'(a,\mu')t'$: if $\mathcal{F}(\lambda) \neq \lambda'$, then from the induction hypothesis $\mathbb{P}^{\sigma'}_{\mathcal{M}'}((\mathfrak{m},s),\lambda') = 0$; and if $\mathcal{F}((a,\mu)t) \neq (a,\mu')t'$, then $p_2 = 0$ from (4.2). Now suppose that $\mathcal{F}(\lambda(a,\mu)t) = \lambda'(a,\mu')t'$. From (4.1) we have that $p_1 = \sigma_{\mathsf{u}}(\mathfrak{m},(a,\mu))(\mathfrak{n})$ and from (4.2) we have that $p_2 = \mu(t)\cdot\sigma_{\mathsf{u}}(\mathfrak{n},t)(\mathfrak{o})$. Applying the induction hypothesis, we conclude the induction, since

$$\begin{aligned}\mathbb{P}^{\sigma'}_{\mathcal{M}'}((\mathfrak{o},t),\lambda'(a,\mu')t') &= \sum_{\mathfrak{m},\mathfrak{n}\in\mathfrak{N}} \mathbb{P}^{\sigma}_{\mathcal{M}}(\mathfrak{m},\lambda)\cdot\sigma_{\mathsf{u}}(\mathfrak{m},(a,\mu))(\mathfrak{n})\cdot\mu(t)\cdot\sigma_{\mathsf{u}}(\mathfrak{n},t)(\mathfrak{o}) \\ &= \mathbb{P}^{\sigma}_{\mathcal{M}}(\mathfrak{o},\lambda(a,\mu)t).\end{aligned}$$

We thus have

$$\tilde{\mathbb{P}}^{\sigma'}_{\mathcal{M}'}(w) = \sum_{\substack{\lambda'\in\mathsf{paths}(w)\\(\mathfrak{m},s)\in\mathfrak{N}'}} \mathbb{P}^{\sigma'}_{\mathcal{M}'}((\mathfrak{m},s),\lambda') = \sum_{\substack{\lambda'\in\mathsf{paths}(w)\\\mathcal{F}(\lambda)=\lambda'\\\mathfrak{m}\in\mathfrak{N}}} \mathbb{P}^{\sigma}_{\mathcal{M}}(\mathfrak{m},\lambda) \stackrel{*}{=} \sum_{\lambda\in\mathsf{paths}(w)} \mathbb{P}^{\sigma}_{\mathcal{M}}(\lambda) \stackrel{\text{def}}{=} \tilde{\mathbb{P}}^{\sigma}_{\mathcal{M}}(w).$$

where the equation marked with $*$ is a consequence of $\mathsf{trace}(\lambda) = \mathsf{trace}(\mathcal{F}(\lambda))$. Thus, σ' and σ induce the same trace distribution, and φ, which is defined on traces, satisfies $(\mathcal{M}')^{\sigma'} \models \varphi \Leftrightarrow \mathcal{M}^{\sigma} \models \varphi$. □

Exchanging Order of Composition. We are interested in compositionally synthesising strategies for the composed game $\mathcal{G} = \|_{i\in I}\, \mathcal{G}^i$, that is, find strategies π^i for the respective component games $\mathcal{G}^i$, and then compose them, to $\pi = \|_{i\in I}\, \pi^i$, in order

to obtain a winning strategy for $\mathcal{G}$. Below, we use compositional verification rules for PAs in order to develop our synthesis rules for games, and hence we define the semantics of strategies via inducing PAs (see Definition 3.5). When synthesising strategies π^i for the respective component games $\mathcal{G}^i$, we obtain induced PAs $(\mathcal{G}^i)^{\pi^i}$ by applying the strategies, and then compose them to the PA $\|_{i\in I}\ (\mathcal{G}^i)^{\pi^i}$. For a fully compositional framework for strategy synthesis in games, however, we are interested in the PA $(\|_{i\in I}\ \mathcal{G}^i)^{\|_{i\in I}\pi^i}$, which is constructed by first composing the individual component games $\mathcal{G}^i$, and only then applying the composed Player 1 strategy. The following lemma justifies, via the existence of a functional simulation, that composing Player 1 strategies preserves the trace distribution between such PAs.

Lemma 4.5. *Given compatible normal form games $(\mathcal{G}^i)_{i\in I}$, and Player 1 strategies $(\pi^i)_{i\in I}$, there is a functional simulation from $(\|_{i\in I}\ \mathcal{G}^i)^{\|_{i\in I}\pi^i}$ to $\|_{i\in I}\ (\mathcal{G}^i)^{\pi^i}$.*

Proof. We construct a functional simulation by viewing states in the induced PA $\mathcal{M} = (\|_{i\in I}\ \mathcal{G}^i)^{\|_{i\in I}\pi^i}$ as derived from the paths of the composed game $\mathcal{G} = (\|_{i\in I}\ \mathcal{G}^i)$. These paths are projected to components $\mathcal{G}^i$ and then assigned a corresponding state in the induced PA $(\mathcal{G}^i)^{\pi^i}$. Due to the structure imposed by compatibility, moves chosen at Player 1 states in $\mathcal{G}^i$ can be translated to moves in the composition $\mathcal{M}' = \|_{i\in I}\ (\mathcal{G}^i)^{\pi^i}$.

Denote the induced PA by $\mathcal{M} = \langle S, (S_\Box, S_\bigcirc), \varsigma, \mathcal{A}, \chi, \Delta\rangle$, and the composition of induced PAs by $\mathcal{M}' = \langle S', (S'_\Box, S'_\bigcirc), \varsigma', \mathcal{A}, \chi', \Delta'\rangle$. We define a partial function $\mathcal{F} : S \to S'$, and then show that it is a functional simulation. We use $\vec{\gamma}$ to stand for both Player 2 states $\vec{s}$ and Player 1 state-move tuples $(\vec{s}, (a, \vec{\mu}))$ of the game $\mathcal{G}$, as occurring in the induced PA $\mathcal{M}$ (see Definition 3.5). We write

$$[\vec{\gamma}]^i = \begin{cases} s^i & \text{if } \vec{\gamma} = \vec{s} \in S_\Box \\ (s^i, (a, \mu^i)) & \text{if } \vec{\gamma} = (\vec{s}, a, \vec{\mu}) \text{ and } \mathcal{G}^i \text{ is involved in } \vec{s} \xrightarrow{a} \vec{\mu} \\ s^i & \text{if } \vec{\gamma} = (\vec{s}, (a, \vec{\mu})) \text{ and } \mathcal{G}^i \text{ is not involved in } \vec{s} \xrightarrow{a} \vec{\mu}, \end{cases}$$

and let $[\mathfrak{d}]^i(\mathfrak{m}^i) \stackrel{\text{def}}{=} \sum_{\mathfrak{m}^j, j\neq i} \mathfrak{d}(\vec{\mathfrak{m}})$ for all $\vec{\mathfrak{m}} \in \mathfrak{M} = \prod_{i\in I} \mathfrak{M}^i$. We define $\mathcal{F}$ by $[\mathcal{F}(\vec{\gamma}, \mathfrak{d})]_i = ([\vec{\gamma}]^i, [\mathfrak{d}]^i)$ for all reachable states $(\vec{\gamma}, \mathfrak{d}) \in S$ of $\mathcal{M}$, and all $i \in I$. We now show that $\mathcal{F}$ is a functional simulation.

Case (F1). We show that $\overline{\mathcal{F}}(\varsigma) = \varsigma'$. Note that, due to the normal form, the initial distribution ς of $\mathcal{M}$ only maps to states of the form $S_\Box \times \mathsf{D}(\mathfrak{M})$, and the initial distribution ς' of $\mathcal{M}'$ only maps to states of the form $\prod_{i\in I} S^i_\Box \times \mathsf{D}(\mathfrak{M}^i)$. For such states $(\vec{s}, \mathfrak{d}) \in S_\Box \times \mathsf{D}(\mathfrak{M})$, we have $[\mathcal{F}(\vec{s}, \mathfrak{d})]_i = (s^i, [\mathfrak{d}]^i)$, and so $\overline{\mathcal{F}}(\varsigma)((s^1, [\mathfrak{d}]^1), (s^2, [\mathfrak{d}]^2), \ldots) = \varsigma(\vec{s}, \mathfrak{d}) = \varsigma'((s^1, [\mathfrak{d}]^1), (s^2, [\mathfrak{d}]^2), \ldots)$.

Case (F2). According to the definition of the induced PA, $\mathcal{M} = \mathcal{G}^{\|_{i\in I}\pi^i}$ has transitions $(\vec{\gamma}, \mathfrak{d}) \xrightarrow{a} \mu^{\pi}_{\mathfrak{d}_+}$, for some belief $\mathfrak{d}_+$, which are induced from transitions $\vec{s} \xrightarrow{a} \vec{\mu}$ of the game composition $\mathcal{G}$. For each component $\mathcal{G}^i$, we apply the strategy π^i separately, and obtain that, for each transition $s^i \xrightarrow{a}{}^i \mu^i$ in $\mathcal{G}^i$, the transition $([\vec{\gamma}]^i, [\mathfrak{d}]^i) \xrightarrow{a} (\mu^i)^{\pi^i}_{[\mathfrak{d}_+]^i}$ is in the induced PA $(\mathcal{G}^i)^{\pi^i}$. Then, composing the induced PAs $(\mathcal{G}^i)^{\pi^i}$ yields a transition $\mathcal{F}(\vec{\gamma}, \mathfrak{d}) \xrightarrow{a}{}' \nu$ in $\mathcal{M}'$, where ν is as follows: we consider states $\vec{\gamma}_{+\!\!+}$ of $\mathcal{M}$ and beliefs $\mathfrak{d}_{+\!\!+}$, such that $[\vec{\gamma}]^i = [\vec{\gamma}_{+\!\!+}]^i$ and $[\mathfrak{d}]^i = [\mathfrak{d}_+]^i = [\mathfrak{d}_{+\!\!+}]^i$ for all $i \notin \Gamma(a, \vec{\gamma}_{+\!\!+})$, that is, the components not involved in the transition do not evolve, and for such $\vec{\gamma}_{+\!\!+}$ and $\mathfrak{d}_{+\!\!+}$ we have

$$\begin{aligned}
\nu(\mathcal{F}(\vec{\gamma}_{+\!\!+}, \mathfrak{d}_{+\!\!+})) &= \prod_{i\in\Gamma(a,\vec{\gamma}_{+\!\!+})} (\mu^i)^{\pi^i}_{[\mathfrak{d}_+]^i}([\vec{\gamma}_{+\!\!+}]^i, [\mathfrak{d}_{+\!\!+}]^i) && \text{(Definition 4.3)} \\
&= \mu^{\pi}_{\mathfrak{d}_+}(\vec{\gamma}_{+\!\!+}, \mathfrak{d}_{+\!\!+}) && \text{(Definition 4.5)} \\
&= \overline{\mathcal{F}}(\mu^{\pi}_{\mathfrak{d}_+})(\mathcal{F}(\vec{\gamma}_{+\!\!+}, \mathfrak{d}_{+\!\!+})). && \text{(definition of } \overline{\mathcal{F}}\text{)}
\end{aligned}$$

We thus have that $\mathcal{F}(\vec{\gamma}, \mathfrak{d}) \xrightarrow{a}{}' \overline{\mathcal{F}}(\mu^{\pi}_{\mathfrak{d}})$ is in $\mathcal{M}'$, concluding the proof of (F2). □

In general, there is no simulation in the other direction, since in the PA composition the states that were originally Player 1 states can no longer be distinguished, see, for instance, Figure 4.5, as well as the following example.

Example$^{\mathsf{re}}$ 4.8. *We consider again the games in Figure 4.2 to illustrate Lemma 4.5. In Figure 4.6 we show the PA $\mathcal{M}_{re} = (\mathcal{G}^1_{re} \parallel \mathcal{G}^2_{re})^{\pi^1\|\pi^2}$, where $\mathcal{G}^1_{re}$ and $\mathcal{G}^2_{re}$ are the two component games on the left in Figure 4.2, and π^1 and π^2 are Player 1 strategies uniformly randomising between choices. Hence, $\mathcal{M}_{re}$ is obtained by first composing the component games, and then applying the composed strategy $\pi^1 \parallel \pi^2$, a process which we call "compose-and-schedule". In Figure 4.7 we show the PA $\mathcal{M}'_{re} = (\mathcal{G}^1_{re})^{\pi^1} \parallel (\mathcal{G}^2_{re})^{\pi^2}$, which is obtained by first applying the Player 1 strategies to the games, and then composing the resulting PAs, which we call "schedule-and-compose".*

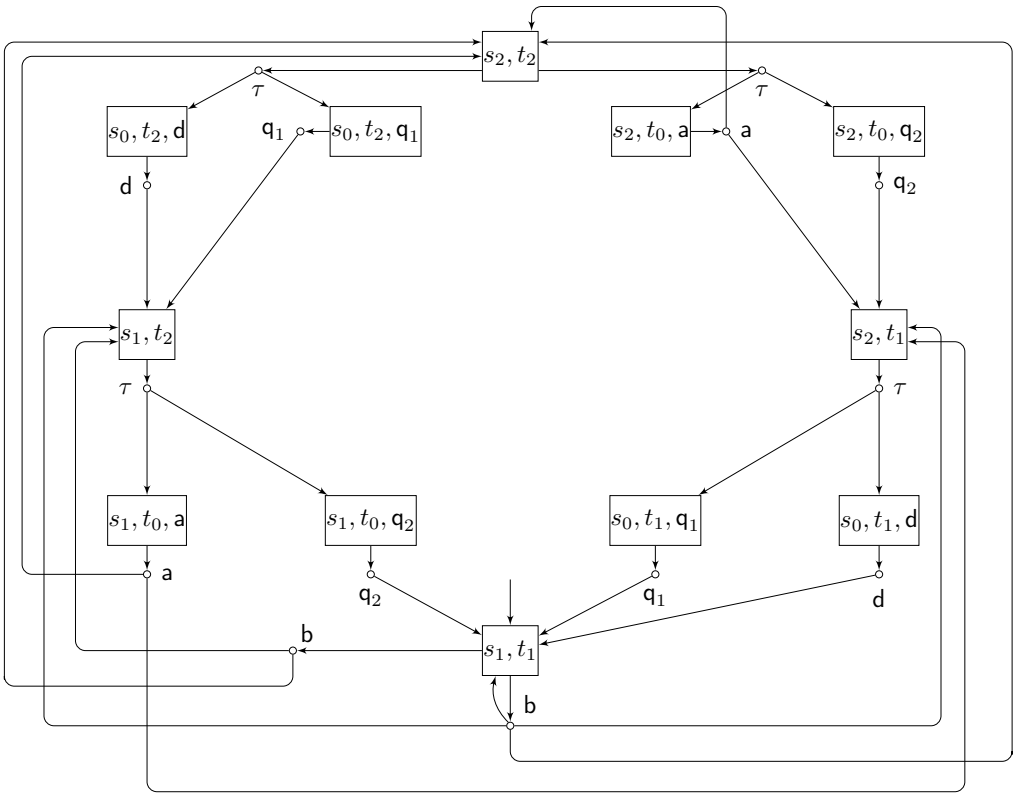


Figure 4.6: "Compose-and-Schedule": the PA $\mathcal{M}_{\mathsf{re}}$ induced from the composed game $\mathcal{G}_{\mathsf{re}} = \mathcal{G}^1_{\mathsf{re}} \parallel \mathcal{G}^2_{\mathsf{re}}$ from Figure 4.2 by applying a Player 1 strategy uniformly randomising between choices. State labels are abbreviated: (s, t, a) stands for $((s, t), (\mathsf{a}, \mu))$, where μ is the distribution corresponding to the outgoing transition.

According to Lemma 4.5, we can construct a functional simulation. Thus we have

$$\mathcal{F}(s_2, t_2) = s_2, t_2$$

$$\begin{aligned}
\mathcal{F}(s_0, t_2, \mathsf{d}) &= (s_0, \mathsf{d}), t_2 & \qquad \mathcal{F}(s_2, t_0, \mathsf{a}) &= s_2, (t_0, \mathsf{a}) \\
\mathcal{F}(s_0, t_2, \mathsf{q}_1) &= (s_0, \mathsf{q}_1), t_2 & \qquad \mathcal{F}(s_2, t_0, \mathsf{q}_2) &= s_2, (t_0, \mathsf{q}_2) \\
\mathcal{F}(s_1, t_2) &= s_1, t_2 & \qquad \mathcal{F}(s_2, t_1) &= s_2, t_1 \\
\mathcal{F}(s_1, t_0, \mathsf{a}) &= s_1, (t_0, \mathsf{a}) & \qquad \mathcal{F}(s_0, t_1, \mathsf{q}_1) &= (s_0, \mathsf{q}_1), t_1 \\
\mathcal{F}(s_1, t_0, \mathsf{q}_2) &= s_1, (t_0, \mathsf{q}_2) & \qquad \mathcal{F}(s_0, t_1, \mathsf{d}) &= (s_0, \mathsf{d}), t_1
\end{aligned}$$

$$\mathcal{F}(s_1, t_1) = s_1, t_1,$$

where we use the same notation as in the figures to abbreviate states: for example, s_1, t_2 is the state (s_1, t_2), and s_0, t_2, d, since only leading to (s_1, t_2) is the state $((s_0, t_2), (\mathsf{d}, (s_1, t_2)))$; and $(s_0, \mathsf{d}), t_2$ stands for $((s_0, (\mathsf{d}, s_1)), t_2)$.

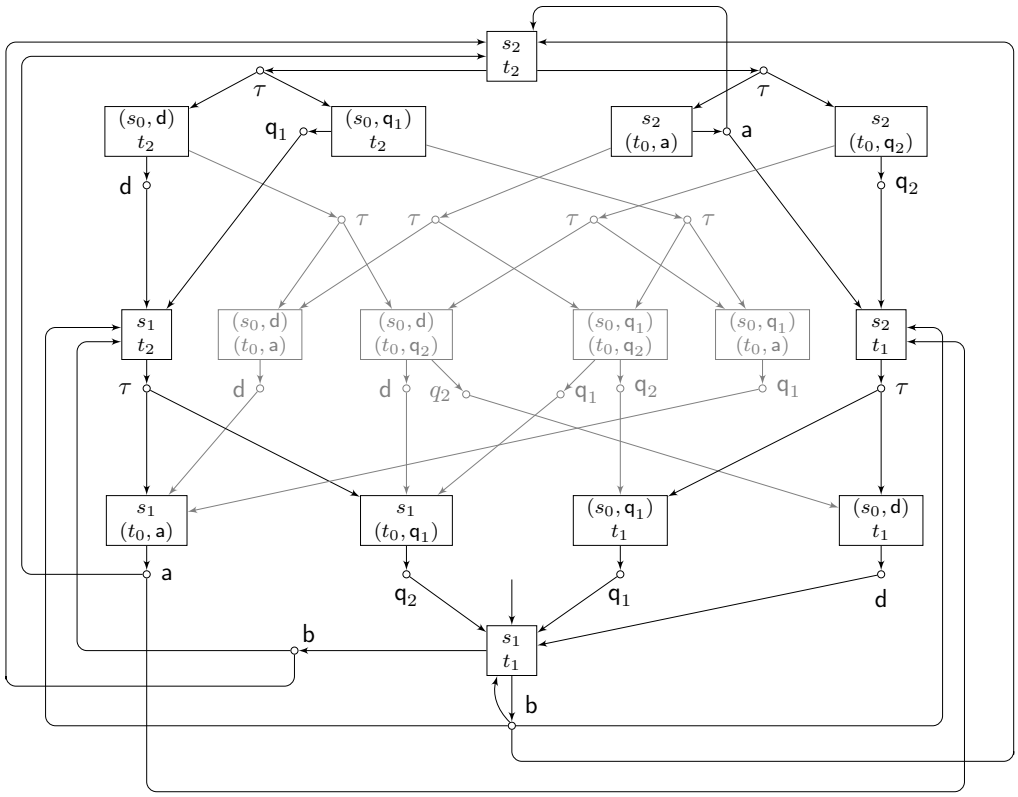


Figure 4.7: "Schedule-and-Compose": the PA $\mathcal{M}'_{\mathsf{re}}$ obtained from applying uniformly randomising Player 1 strategies to the games $\mathcal{G}^1_{\mathsf{re}}$ and $\mathcal{G}^2_{\mathsf{re}}$ in Figure 4.2, and then composing the induced PAs. State labels are abbreviated: (s, a) stands for $(s, (a, \delta)$, where δ is the distribution corresponding to the outgoing transition in the respective PA $(\mathcal{G}^1)^{\pi^1}$ or $(\mathcal{G}^2)^{\pi^2}$.

The functional simulation guarantees that all transitions in $\mathcal{M}_{re}$ are matched by transitions in $\mathcal{M}'_{re}$. However, note that there are transitions and states in $\mathcal{M}'_{re}$ that are not matched in $\mathcal{M}_{re}$ (shown in grey in Figure 4.7), and so no functional simulation exists in the other direction. $\mathcal{M}_{re}$ is obtained from the composed game, where the τ transitions are only available in Player 1 states due to the game composition; $\mathcal{M}'_{re}$ is obtained by composing the induced PAs, and so the τ transitions are not disabled by the composition, since the states are no longer specific to Player 1. Every path in $\mathcal{M}_{re}$ exists in $\mathcal{M}'_{re}$, and in this example it is clear that every Player 2 strategy of $\mathcal{M}_{re}$ can be mapped to a Player 2 strategy of $\mathcal{M}'_{re}$, preserving the trace distribution, and so any specification defined on traces satisfied in $\mathcal{M}'_{re}$ is satisfied in $\mathcal{M}_{re}$.

4.2.3 Synthesis Rules

Our main result for assume-guarantee synthesis is that any verification rule for PAs gives rise to a synthesis rule for games with the same side conditions. The idea is to induce PAs from the strategies synthesised for games, apply PA verification rules to show that Player 2 cannot spoil Player 1 in the composition, and, using Lemma 4.5, we can lift the result back into the game domain. A *rule* is a higher-order logic formula

$$\forall P \in \mathfrak{P}, Q \in \mathfrak{Q}(P) \,.\, P \Rightarrow Q,$$

where $\mathfrak{P}$ is a set of allowed *premises*, and $\mathfrak{Q}(P)$ is a set of allowed *conclusions*, which depend on how the premises P were chosen. We write such a rule as

$$(\textsc{Name}) \quad \frac{P}{Q} \quad \text{s.t. } P \in \mathfrak{P}, Q \in \mathfrak{Q}(P).$$

Typically, $\mathfrak{P}$ and $\mathfrak{Q}(P)$ are implicit, and defined informally via matching the syntax; however, we make explicit some *side conditions* that restrict $\mathfrak{P}$ and $\mathfrak{Q}(P)$, for instance, by restricting the allowable choices of action alphabets. An *instance* of a rule is written $\frac{P}{Q}$, where P and Q have been chosen in accordance with all restrictions.

Lifting Theorem. The following theorem, the main result of our assume-guarantee framework, establishes a lifting from PA verification rules to synthesis rules for games. Recall that the notation $\mathcal{M} \models \phi$ means that all strategies σ satisfy $\mathcal{M}^\sigma \models \phi$.

Theorem 4.6. *Given specifications φ^i_j and φ defined on traces, such that the PA verification rule*

$$\frac{\mathcal{M}^i \models \varphi^i_j \qquad j \in J \qquad i \in I}{\|_{i\in I}\, \mathcal{M}^i \models \varphi}$$

holds for PAs $(\mathcal{M}^i)_{i\in I}$. Then the rule

$$\frac{(\mathcal{G}^i)^{\pi^i} \models \bigwedge_{j\in J} \varphi^i_j \qquad i \in I}{(\|_{i\in I}\, \mathcal{G}^i)^{\|_{i\in I}\pi^i} \models \varphi}$$

holds for all Player 1 strategies $(\pi^i)_{i\in I}$ of compatible normal-form games $(\mathcal{G}^i)_{i\in I}$, with the same action alphabets as the corresponding PAs.

Proof. For all $i \in I$, let $\mathcal{G}^i$ be games, and let π^i be respective Player 1 strategies such that $(\mathcal{G}^i)^{\pi^i} \models \bigwedge_{j\in J} \varphi^i_j$. By applying the PA rule with the PAs $\mathcal{M}^i \stackrel{\text{def}}{=} (\mathcal{G}^i)^{\pi^i}$, where $\mathcal{M}^i \models \bigwedge_{j\in J} \varphi^i_j$ for all $i \in I$ from how the strategies π^i were picked, we have that

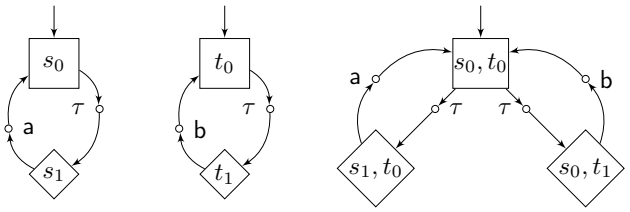


Figure 4.8: Games $\mathcal{G}^1$ (left) and $\mathcal{G}^2$ (centre) to illustrate fairness in their composition $\mathcal{G}^1 \parallel \mathcal{G}^2$ (right).

$\|_{i\in I}\, \mathcal{M}^i \models \varphi$. From Lemma 4.5, there is a functional simulation from $(\|_{i\in I}\, \mathcal{G}^i)^{\|_{i\in I}\pi^i}$ to $\|_{i\in I}\, (\mathcal{G}^i)^{\pi^i}$. Since $\|_{i\in I}\, (\mathcal{G}^i)^{\pi^i} \models \varphi$, applying Lemma 4.4 yields $(\|_{i\in I}\, \mathcal{G}^i)^{\|_{i\in I}\pi^i} \models \varphi$. □

The specifications φ^i_j and φ in Theorem 4.6 can be any MQs, as long as they are defined on traces over the restricted action alphabets required by the relevant PA verification rule. We write the side conditions on the action alphabets next to the rules. In particular, for the premises in Theorem 4.6 we can use the assume-guarantee rules for PAs developed in [85], which are stated for MQs consisting of total expected rewards or ω-regular objectives. Thus, the specification φ for the composed game can, for example, be a CQ, or a summation of rewards, among others.

Fairness. We recall the concept of unconditional process fairness based on [5], which we use in the PA verification rules. Given a composed PA $\mathcal{M} = \|_{i\in I}\, \mathcal{M}^i$, a strategy σ is (unconditionally) *fair* if in $\mathcal{M}^\sigma$ the actions $\mathcal{A}^i$ of each component $\mathcal{M}^i$ appear infinitely often on almost all paths,that is, each component makes progress infinitely often with probability 1. We write $\mathcal{M} \models^{\mathsf{u}} \varphi$ if, for all unconditionally fair strategies σ, $\mathcal{M}^\sigma \models \varphi$. Note that fairness can be expressed as a specification defined on traces, and thus incorporated into the rules of Theorem 4.6. Unconditional fairness corresponds precisely to the fairness conditions used in the PA rules of [85].

Our game composition does not guarantee freedom from *deadlocks*, that is, states without outgoing moves. For a single component, unconditional fairness is equivalent to only requiring deadlock-freedom, and so our monolithic synthesis method does not explicitly have to take the fairness assumption for component games into account. In a deadlocked game, if Player 2 cannot avoid reaching deadlocks, there are no fair Player 2 strategies, rendering the synthesis problem trivial, as any specification is then vacuously satisfied under fairness. We also observe that, when composing stopping games, fair strategies only exist in the composition if all components synchronise on actions in terminal states, and can enter such terminal states together with probability one, for instance by synchronising on entering terminal states.

Example 4.9 (Fairness)**.** *Consider the two normal-form games $\mathcal{G}^1$ and $\mathcal{G}^2$ on the left in Figure 4.8, and their composition $\mathcal{G} = \mathcal{G}^1 \parallel \mathcal{G}^2$ on the right. Fair* Player 2 *strategies are those that play actions in each of $\mathcal{A}^1 = \{a\}$ and $\mathcal{A}^2 = \{b\}$ infinitely often with probability one. For example, the* Player 2 *strategy $\sigma(\frac{1}{2})$ picking $(\tau, (s_1, t_0))$ and $(\tau, (s_0, t_1))$ with uniform probability at (s_0, t_0) is fair. More generally, for $0 < \delta < 1$, any* Player 2 *strategy $\sigma(\delta)$ assigning probability δ to $(\tau, (s_1, t_0))$ and probability $1-\delta$ to $(\tau, (s_0, t_1))$ is fair. However, the strategies that always pick $(\tau, (s_1, t_0))$ or $(\tau, (s_0, t_1))$ are not fair. Note that any* Player 2 *strategy in $\mathcal{G}^1$ and $\mathcal{G}^2$ is fair in any case, since they are (monolithic) component games without deadlocks.*

We now present a set of assume-guarantee strategy synthesis rules for games. The rules are stated for compatible normal-form games $\mathcal{G}^1$ and $\mathcal{G}^2$, and we are interested in synthesis of strategies under the assumption that the scheduler is fair. The conclusion of one rule can serve as the premise of other rules (thanks to the exchange in the order of composition enabled by Lemma 4.5), and so we can combine several rules to deduce properties about more than two components. Our rules are based on those in [85], but we emphasise that Theorem 4.6 is applicable to any PA verification rule.

The (Conj) **Rule.** A fundamental rule in our framework is the (Conj) rule that allows us to put specifications in conjunction. Each component $\mathcal{G}^i$, with action alphabet $\mathcal{A}^i$, has its specification φ^i defined on $\mathcal{A}^i$-traces, and the rule guarantees that the conjunction $\bigwedge_{i\in I} \varphi^i$ is satisfied in the composition $\|_{i\in I}\ \mathcal{G}^i$. We thus have for all π^i, $i \in I$, the rule

$$\text{(Conj)}\quad \frac{(\mathcal{G}^i)^{\pi^i} \models^{\mathsf{u}} \varphi^i \qquad i \in I}{(\|_{i\in I}\ \mathcal{G}^i)^{\|_{i\in I}\pi^i} \models^{\mathsf{u}} \bigwedge_{i\in I} \varphi^i}$$

To derive the (Conj) rule, we prove the corresponding PA verification rule, and use Theorem 4.6 to derive the synthesis rule.

Proposition 4.7. *Let $(\mathcal{M}^i)_{i\in I}$ be PAs with respective action alphabets $(\mathcal{A}^i)_{i\in I}$, and let $(\varphi^i)_{i\in I}$ be specifications, where, for all $i \in I$, φ^i is defined on $\mathcal{A}^i$-traces. The following rule holds:*

$$\text{(Conj)}\quad \frac{\mathcal{M}^i \models^{u} \varphi^i \qquad i \in I}{\|_{i\in I}\ \mathcal{M}^i \models^{u} \bigwedge_{i\in I} \varphi^i.}$$

Proof. Let $\mathcal{M} = \|_{i\in I}\ \mathcal{M}^i$. We first recall concepts of projections from [85]. Given a state $\vec{s} = (s^1, s^2, \ldots)$ of $\mathcal{M}$, the projection of $\vec{s}$ onto $\mathcal{M}^i$ is $\vec{s}{\uparrow}_{\mathcal{M}^i} \stackrel{\text{def}}{=} s^i$, and for a distribution $\vec{\mu}$ over states of $\mathcal{M}$ we define its projection by $\vec{\mu}{\uparrow}_{\mathcal{M}^i}(s^i) \stackrel{\text{def}}{=} \sum_{\vec{s}{\uparrow}_{\mathcal{M}^i}=s^i} \vec{\mu}(s)$.

Given a path λ of $\mathcal{M}$, the projection of λ onto $\mathcal{M}^i$, denoted by $\lambda{\restriction}_{\mathcal{M}^i}$, is the path obtained from λ by projecting each state and distribution, and removing all moves with actions not in the alphabet of $\mathcal{M}^i$, together with the subsequent states. Given a strategy σ of $\mathcal{M}$, its projection $\sigma{\restriction}_{\mathcal{M}^i}$ onto $\mathcal{M}^i$ is such that, for any finite path λ^i of $\mathcal{M}^i$ and transition $\mathsf{last}(\lambda^i) \xrightarrow{a} \mu^i$,

$$\sigma{\restriction}_{\mathcal{M}^i}(\lambda^i)(a,\mu^i) \stackrel{\text{def}}{=} \sum_{\lambda{\restriction}_{\mathcal{M}^i}=\lambda^i} \sum_{\vec{\mu}{\restriction}_{\mathcal{M}^i}=\mu^i} \mathbb{P}^{\sigma}_{\mathcal{M}}(\lambda)\cdot\sigma(\lambda)(a,\mu)/\mathbb{P}^{\sigma{\restriction}_{\mathcal{M}^i}}_{\mathcal{M}^i}(\lambda^i).$$

Fix a fair strategy σ for $\mathcal{M}$. From Lemma 2 in [85], the projections $\sigma{\restriction}_{\mathcal{M}^i}$ are well-defined fair strategies, for all $i \in I$. Hence, $\mathcal{M}^i \models^{\mathsf{u}} \varphi^i$ implies $(\mathcal{M}^i)^{\sigma{\restriction}_{\mathcal{M}^i}} \models \varphi^i$. Furthermore, from Lemma 7.2.6 in [118], $\tilde{\mathbb{P}}^{\sigma}_{\mathcal{M}}(w) = \tilde{\mathbb{P}}^{\sigma{\restriction}_{\mathcal{M}^i}}_{\mathcal{M}^i}(w)$ for any trace w over actions $\mathcal{A}^i$. Since φ^i is defined on $\mathcal{A}^i$-traces, we have that $(\mathcal{M}^i)^{\sigma^i{\restriction}_{\mathcal{M}^i}} \models \varphi^i \Leftrightarrow \varphi(\mathbb{P}^{\sigma^i{\restriction}_{\mathcal{M}^i}}_{\mathcal{M}^i}) \Leftrightarrow \varphi(\mathbb{P}^{\sigma}_{\mathcal{M}}) \Leftrightarrow \mathcal{M}^{\sigma} \models \varphi^i$ Finally, since σ was an arbitrary fair strategy of $\mathcal{M}$, we have $\mathcal{M} \models^{\mathsf{u}} \bigwedge_{i\in I} \varphi^i$. □

Example 4.10 (Conjunction Rule)**.** *We illustrate the* (Conj) *rule and its side conditions using the games in Figure 4.8. Define reward structures r_1 and c_1 with $r_1(\mathsf{a}) = c_1(\mathsf{a}) = 1$ (and zero otherwise), r_2 and c_2 with $r_2(\mathsf{b}) = c_2(\mathsf{b}) = 1$ (and zero otherwise), and c with $c(\mathsf{a}) = c(\mathsf{b}) = 1$. Clearly, $\mathcal{G}^1 \models \mathsf{Eratio}(r_1/c)(1)$ and $\mathcal{G}^2 \models \mathsf{Eratio}(r_2/c)(1)$, and, equivalently, $\mathcal{G}^1 \models \mathsf{Eratio}(r_1/c_1)(1)$ and $\mathcal{G}^2 \models \mathsf{Eratio}(r_2/c_2)(1)$. Now consider the conjunctions $\phi_1 = \mathsf{Eratio}(r_1/c)(1) \wedge \mathsf{Eratio}(r_2/c)(1)$ and $\phi_2 = \mathsf{Eratio}(r_1/c_1)(1) \wedge \mathsf{Eratio}(r_2/c_2)(1)$ in the composed game $\mathcal{G}$, and a (fair) Player 2 strategy $\sigma(\delta)$ with $0 < \delta < 1$, as defined in Example 4.9. We have, for arbitrary Player 1 strategies, that $\mathbb{E}^{\pi,\sigma(\delta)}_{\mathcal{G}}[\mathsf{ratio}(r_1/c)] = \delta$ and $\mathbb{E}^{\pi,\sigma(\delta)}_{\mathcal{G}}[\mathsf{ratio}(r_2/c)] = 1-\delta$, and so the conjunction ϕ_1 is not satisfied in the composed game $\mathcal{G}$. However, c does not satisfy the restrictions on the action alphabets. In contrast, c_1 and c_2 conform with this restriction: we have $\mathbb{E}^{\pi,\sigma(\delta)}_{\mathcal{G}}[\mathsf{ratio}(r_1/c_1)] = 1$ and $\mathbb{E}^{\pi,\sigma(\delta)}_{\mathcal{G}}[\mathsf{ratio}(r_2/c_2)] = 1$, and so ϕ_2 is satisfied in $\mathcal{G}$. Note that under a Player 2 strategy that is not fair, for example $\sigma(0)$ or $\sigma(1)$, neither conjunction is satisfied in $\mathcal{G}$.*

The (Prop) **Rule.** Another useful rule in our framework is the non-compositional (Prop) rule to deduce specifications via propositional logic. We define the relation $\vdash$ between MQs such that $\varphi \vdash \psi$ if and only if ψ follows from φ in propositional logic with each objective being interpreted as a literal. We thus have, for all π, the rule

$$(\text{Prop})\quad \frac{\mathcal{G}^{\pi} \models^{\mathsf{u}} \varphi}{\mathcal{G}^{\pi} \models^{\mathsf{u}} \psi} \text{ s.t. } \varphi \vdash \psi.$$

Example 4.11 (Propositional Rule)**.** *We show how to use the* (Prop) *rule together with the* (Conj) *rule by deducing a specification via sequential rule application. We start by instantiating the* (Conj) *rule with the games* $\mathcal{G}^1$ *and* $\mathcal{G}^2$ *and specifications* φ_A *and* φ_G *defined on* $\mathcal{A}_A$*- and* $\mathcal{A}_G$*-traces, respectively, such that* $\mathcal{A}_A \subseteq \mathcal{A}^1$ *and* $\mathcal{A}_A \cup \mathcal{A}_G \subseteq \mathcal{A}^2$*. We then instantiate the* (Prop) *rule, taking as premise the conclusion of the* (Conj) *rule; the side condition of the* (Prop) *rule is instantiated noting that* $\varphi_A \wedge (\varphi_A \to \varphi_G) \vdash \varphi_G$ *(modus ponens). Thus*

$$\begin{array}{l}(\textsc{Conj})\\(\textsc{Prop})\end{array}\ \frac{(\mathcal{G}^1)^{\pi^1} \models^{u} \varphi_A \qquad (\mathcal{G}^2)^{\pi^2} \models^{u} \varphi_A \to \varphi_G}{(\mathcal{G}^1 \parallel \mathcal{G}^2)^{\pi^1 \| \pi^2} \models^{u} \varphi_A \wedge (\varphi_A \to \varphi_G)}}{(\mathcal{G}^1 \parallel \mathcal{G}^2)^{\pi^1 \| \pi^2} \models^{u} \varphi_G.}$$

The (Asym) **and** (Circ) **Rules.** The (Asym) rule is used for asymmetric contracts, where one component $\mathcal{G}^2$ satisfies the specification $\varphi_A \to \varphi_G$ (an implication), and the other component $\mathcal{G}^1$ satisfies the specification φ_A. Often φ_A is called the *assumption* and φ_G is called the *guarantee*. The rule guarantees that the composition satisfies φ_G, even though it might be the case that Player 1 in $\mathcal{G}^2$ cannot force this. Rather, the (Asym) rule establishes a contract between the two components, where Player 1 in $\mathcal{G}^2$ is only required to satisfy φ_G if Player 1 in $\mathcal{G}^1$ satisfies φ_A. Thus, for all π^1, π^2, and φ_A and φ_G defined on $\mathcal{A}_A$- and $\mathcal{A}_G$-traces, respectively, we have the rule

$$(\textsc{Asym})\ \frac{(\mathcal{G}^1)^{\pi^1} \models^{\mathsf{u}} \varphi_A \qquad (\mathcal{G}^2)^{\pi^2} \models^{\mathsf{u}} \varphi_A \to \varphi_G}{(\mathcal{G}^1 \parallel \mathcal{G}^2)^{\pi^1 \| \pi^2} \models^{\mathsf{u}} \varphi_G}\ \text{s.t. } \mathcal{A}_A \subseteq \mathcal{A}^1, \mathcal{A}_G \cup \mathcal{A}_A \subseteq \mathcal{A}^2.$$

We note that the (Asym) rule can be deduced from the (Conj) and (Prop) rules: the sequential instantiation of rules in Example 4.11 can be generalised to yield a (uninstantiated) rule, where the side conditions of the resulting rule are all the side conditions taken together (unless they are necessarily satisfied).

We extend the idea of contracts between components further in the (Circ) rule, to resolve circular contracts (this terminology of circularity derives from [85]). $\mathcal{G}^2$ has the specification $\varphi_A \to \varphi_G$ as in the (Asym) rule, but Player 1 in $\mathcal{G}^1$ can only satisfy φ_A under its own assumption φ_B, defined on $\mathcal{A}_B$-traces. This additional specification φ_B must be guaranteed by Player 1 in $\mathcal{G}^2$, in order to resolve the circularity. Thus, for all π^1, π^2, we have the rule

$$(\textsc{Circ})\ \frac{\begin{array}{ll}(\mathcal{G}^1)^{\pi^1} & \models^{\mathsf{u}} \varphi_B \to \varphi_A\\ (\mathcal{G}^2)^{\pi^2} & \models^{\mathsf{u}} \varphi_B \wedge (\varphi_A \to \varphi_G)\end{array}}{(\mathcal{G}^1 \parallel \mathcal{G}^2)^{\pi^1 \| \pi^2} \models^{\mathsf{u}} \varphi_G}\ \text{s.t. } \mathcal{A}_A, \mathcal{A}_B \subseteq \mathcal{A}^1 \cap \mathcal{A}^2, \mathcal{A}_G \subseteq \mathcal{A}^2.$$

The (Circ) rule can be derived from the (Conj) and (Prop) rules in a way similarly to how the (Asym) rule is derived.

4.3 Compositional Pareto Sets

We can compute an (approximation) of the Pareto set of the global specification φ from the Pareto sets of the local specifications φ, which also allows us to pick the targets of the φ^i so that φ is achievable. Consider a rule

$$\text{(R)}\quad \frac{(\mathcal{G}^i)^{\pi^i} \models^{\mathsf{u}} \varphi^i \qquad i \in I}{(\|_{i\in I}\, \mathcal{G}^i)^{\|_{i\in I}\pi^i} \models^{\mathsf{u}} \varphi,}$$

where the specifications φ^i and φ consist of objectives $(O_j)_{j\in J}$ in the following way. For each $i \in I$, φ^i is a MQ over the subset $\{O_{j^i_1}, O_{j^i_2}, \ldots\}$, where the indices $j^i_1, j^i_2, \ldots$ determine the dimensions of the associated local Pareto set. Further, φ is a MQ over the subset $\{O_{j_1}, O_{j_2}, \ldots\}$, where the indices $j_1, j_2, \ldots$ determine the dimensions of the global Pareto set. Note that each objective O_j may be present in several specifications φ^i, and in φ: for instance, in the (Asym) rule we have local specifications O_1 and $O_1 \to O_2$, and O_2 in the global specification. Each objective occurring in a specification is associated with a dimension in the Pareto set, so that the dimensions of a Pareto set $\mathsf{Pareto}(\mathcal{G}^i \,|\, \varphi^i)$ correspond to the objectives $O_{j^i_1}$, $O_{j^i_2}$, ... For a vector $\vec{v} \in \mathbb{R}^{|J|}$, we let $\vec{v}\restriction_{\varphi^i} \stackrel{\text{def}}{=} (v_{j^i_1}, v_{j^i_2}, \ldots)$ be the subvector of $\vec{v}$ consisting of all coordinates in φ^i.

We now show how to compose Pareto sets computed for the local games. Recall that the boundary of a Pareto set may not be achievable, and so, for each Pareto set $\mathsf{Pareto}(\mathcal{G}^i \,|\, \varphi^i)$, we consider its interior, which we denote by P^i. We define the *lifting* $\uparrow P^i$ of a set P^i by $\uparrow P^i \stackrel{\text{def}}{=} \{\vec{v} \in \mathbb{R}^{|J|} \,|\, \vec{v}\restriction_{\varphi^i} \in P^i\}$, and the dimensions J of the lifting correspond to all objectives $O_1, O_2, \ldots$ in the rule under consideration. The set of target vectors that are consistent with the achievable objectives for all specifications φ^i in the respective games is $\bigcap_{i\in I} \uparrow P^i$, where intersection of the liftings $\uparrow P^i$ ensures that the targets are consistent with all local Pareto sets. We let the *compositional Pareto set* $\mathsf{CPareto}((\mathcal{G}^i)_{i\in I} \,|\, \varphi, (\varphi^i)_{i\in I}) \stackrel{\text{def}}{=} \mathsf{cl}(\{\vec{v}\restriction_{\varphi} \,|\, \vec{v} \in \bigcap_{i\in I} \uparrow P^i\})$, that is, the closure of the projection to the dimensions of φ. The compositional Pareto set is an under-approximation of the Pareto set computed directly on the composed game $\mathcal{G}$ for φ.

Proposition 4.8. *Let $(\mathcal{G}^i)_{i\in I}$ be compatible games, and let $(\varphi^i)_{i\in I}$ and φ be defined on traces. If the rule (R) holds, then $\mathsf{CPareto}((\mathcal{G}^i)_{i\in I} \,|\, \varphi, (\varphi^i)_{i\in I}) \subseteq \mathsf{Pareto}(\|_{i\in I}\, \mathcal{G}^i \,|\, \varphi)$.*

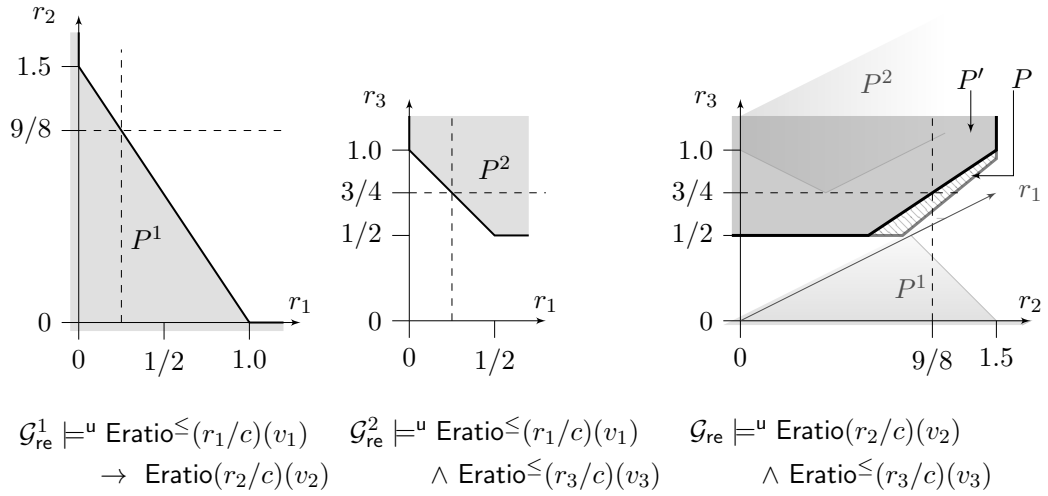


Figure 4.9: Pareto set under-approximations for games in Figure 4.2, with specifications beneath the respective sets. Signs of $\mathsf{Eratio}^{\leq}$ objectives are flipped. On the right is the compositional Pareto set $P' = \mathsf{CPareto}(\mathcal{G}^1_{\mathsf{re}}, \mathcal{G}^2_{\mathsf{re}} \,|\, \varphi_{\mathsf{re}}, (\varphi^1_{\mathsf{re}}, \varphi^2_{\mathsf{re}}))$, an approximation P to the Pareto set $\mathsf{Pareto}(\mathcal{G}^1_{\mathsf{re}} \parallel \mathcal{G}^2_{\mathsf{re}} \,|\, \varphi)$, as well as the oblique projections of P^1 and P^2 for reference.

Proof. Take $\vec{v}' \in \{\vec{v}\restriction_{\varphi} \,|\, \vec{v} \in \bigcap_{i\in I} \uparrow P^i\}$. Thus, there exists $\vec{v}$ such that $\vec{v}\restriction_{\varphi} = \vec{v}'$, and, for all $i \in I$, $\vec{v}\restriction_{\varphi^i} \in P^i$. For each i, since P^i is the interior of $\mathsf{Pareto}(\mathcal{G}^i \,|\, \varphi^i)$, there exists a finite DU Player 1 strategy π^i achieving $\varphi^i[\vec{v}\restriction_{\varphi^i}]$. We now instantiate the rule (R) with $\varphi^i[\vec{v}\restriction_{\varphi^i}]$ and $\varphi[\vec{v}']$. The strategy $\|_{i\in I}\, \pi^i$ is finite DU and achieves $\varphi[\vec{v}']$ against all fair Player 2 strategies. Hence $\vec{v}' \in \mathsf{Pareto}(\mathcal{G} \,|\, \varphi)$. As Pareto sets are closed, the result follows. □

Hence $\vec{v}'$ in the compositional Pareto set $\mathsf{CPareto}((\mathcal{G}^i)_{i\in I} \,|\, \varphi, (\varphi^i)_{i\in I})$ is achieved by instantiating the specifications φ^i with targets $\vec{v}\restriction_{\varphi^i} \in P^i$ for any $\vec{v}$ such that $\vec{v}\restriction_{\varphi} = \vec{v}'$.

Example$^{\mathsf{re}}$ 4.12 (Compositional Pareto Set)**.** *We return to our running example of Section 3.6 to illustrate compositional Pareto set computation. Consider again the games in Figure 4.2. We want to compute an under-approximation for the Pareto set for $\mathcal{G}_{re} \stackrel{\text{def}}{=} \mathcal{G}^1_{re} \parallel \mathcal{G}^2_{re}$ (right) by computing the under-approximations of the components $\mathcal{G}^1_{re}$ (left) and $\mathcal{G}^2_{re}$ (centre). The local Pareto sets $\mathsf{Pareto}(\mathcal{G}^1_{re} \,|\, \varphi^1_{re})$ and $\mathsf{Pareto}(\mathcal{G}^2_{re} \,|\, \varphi^2_{re})$ are shown in Figure 4.9 (left) and (centre) respectively (their interiors are denoted P^1 and P^2, respectively). We defer the discussion of computing these local Pareto sets to Section 5.5. In Figure 4.9 (right), we show the compositional Pareto set $P' = \mathsf{CPareto}((\mathcal{G}^1_{re}, \mathcal{G}^2_{re}) \,|\, \varphi_{re}, (\varphi^1_{re}, \varphi^2_{re}))$ as well as an approximation of the Pareto set $P = \mathsf{Pareto}(\mathcal{G}^1_{re} \parallel \mathcal{G}^2_{re} \,|\, \varphi_{re})$, computed for φ_{re} directly on the composition $\mathcal{G}_{re}$ (here for all Player 2 strategies), which is a superset of the compositional Pareto set.*

Composed strategies operate on a restricted view of the history in the composed game. The cause for the gap between the compositional Pareto set P' and the monolithically computed Pareto set P in Figure 4.2 (right) is due to this memory restriction for the composed strategy: in a $\mathsf{Player\ 1}$ state $\vec{s}$ of the composition, the strategy uses only the part of the history that would be seen in the component controlling $\vec{s}$. Note that the PA verification rules in [85] that we based our synthesis rules on are already incomplete.

4.4 Synthesis Procedure

We now develop a step-by-step method for assume-guarantee strategy synthesis. Recall that we require the designer to supply the component games $\mathcal{G}^i$, each with a local specification φ^i, as well as a global specification φ. Ideally, we would like to start with the component games and the global specification, and work backwards using the composition rules of Section 4.2.3 to find a local specification for each component; however, the exploration of this approach is outside the scope of this thesis. While we do not discuss eliminating the local specifications entirely from the information the designer has to supply, here we show that the targets of the local specifications can be found automatically. That is, we consider the local specifications parameterised by the target vectors, requiring only the reward structure and the precise Boolean combination to be supplied.

By computing the Pareto sets of the local specifications, and picking points that are consistent with the target of the global specification, we obtain targets of the local specifications. In the absence of additional criteria, each point on the Pareto frontier is equally desirable from a mathematical perspective. We thus cannot a-priori select a target to synthesise a strategy for, and instead employ an *a-posteriori* approach [88], where a *decision maker* is presented with a Pareto set, and picks the desired point to synthesise a strategy for. We summarise our method as follows:

(S1) **User Input:** a composed game $\mathcal{G} = \|_{i \in I}\, \mathcal{G}^i$, and specifications φ^i, φ, such that they instantiate a synthesis rule in Theorem 4.6.

(S2) **Pareto Set Computation:** compute (approximate) Pareto sets $\mathsf{Pareto}(\mathcal{G}^i \,|\, \varphi^i)$, and compute the compositional Pareto set $P = \mathsf{CPareto}((\mathcal{G}^i)_{i \in I} \,|\, \varphi, (\varphi^i)_{i \in I})$.

(S3) **Decision Maker Feedback:** pick a target vector $\vec{v}$ for the global specification φ from P; matching targets v^i for φ^i can be picked automatically from the Pareto set interiors P^i.

(S4) **Strategy Synthesis:** synthesise strategies π^i, for $\mathcal{G}^i \models \varphi^i[\vec{w}\restriction_{\varphi^i}]$.

(S5) **Output:** the strategy $\|_{i\in I}\, \pi^i$, winning for $\mathcal{G} \models \varphi[\vec{v}]$ by Theorem 4.6.

If the target $\vec{v}$ for the global specification φ is fixed beforehand, we can compute the local targets via step (S2). If all targets are fixed beforehand, the steps (S1), (S4) and (S5) are sufficient to synthesise strategies. The step (S3) allows the decision maker to pick the targets. Recall that the targets can only be picked from the compositional Pareto set $\mathsf{CPareto}((\mathcal{G}^i)_{i\in I} \,|\, \varphi, (\varphi^i)_{i\in I})$, but that the targets in the full Pareto set $\mathsf{Pareto}(\mathcal{G} \,|\, \varphi)$ may not be achievable by a composed strategy.

Example$^{\mathsf{re}}$ 4.13 (Synthesis Procedure)**.** *We discuss our assume-guarantee strategy synthesis procedure on our running example of Section 3.6. The inputs for step (S1) are the local games $\mathcal{G}^1_{\mathsf{re}}$ and $\mathcal{G}^2_{\mathsf{re}}$ with the local specifications φ^1_{re} and φ^2_{re}, and the global specification φ_{re} as defined in the running example. For step (S2), the Pareto sets for $\mathcal{G}^1_{\mathsf{re}}$ and $\mathcal{G}^2_{\mathsf{re}}$ are shown in Figure 4.9 (left) and (centre) respectively, and on the right, we show the compositional Pareto set for $\mathcal{G}_{\mathsf{re}}$. In step (S3) we consider synthesis for points in the compositional Pareto set. If, for example, we want to find a strategy satisfying φ_{re} with $(v_2, v_3) = (\frac{9}{8}, \frac{3}{4})$, we look up a value for v_1 which is consistent in both $\mathsf{Pareto}(\mathcal{G}^1_{\mathsf{re}} \,|\, \varphi^1_{\mathsf{re}})$ and $\mathsf{Pareto}(\mathcal{G}^2_{\mathsf{re}} \,|\, \varphi^2_{\mathsf{re}})$, as indicated by the dashed lines in Figure 4.9 (left) and (centre). We find $v_1 = \frac{1}{4}$ to be consistent for both components. In step (S4), we then synthesise a local strategy π^1 for $\varphi^1_{\mathsf{re}}[(\frac{1}{4}, \frac{9}{8})]$ in $\mathcal{G}^1_{\mathsf{re}}$, and a strategy π^2 for $\varphi^2_{\mathsf{re}}[(\frac{1}{4}, \frac{3}{4})]$ in $\mathcal{G}^2_{\mathsf{re}}$, which we discuss later in Chapter 6. Finally, in step (S5), we return the composed strategy $\pi = \pi^1 \parallel \pi^2$.*

4.5 Summary

In this chapter we presented a framework for assume-guarantee synthesis for strategies in turn-based two-player stochastic games. We developed a composition for such games, together with a compatibility condition, so that strategies of component games can be composed to a strategy of the composed game. Our game composition extends the composition of PAs [118] by distinguishing between the players of the games. Our framework utilises synthesis for specifications defined on traces, consisting of multiple objectives, which we discuss in the following chapters. Our main result of this chapter, Theorem 4.6, is that any verification rule for PAs gives rise to a synthesis rule for games, which allows us to deduce a winning strategy for the composed system. If we are only interested in strategies for the components, we could define game composition dependent on strategies via $(\mathcal{G}^1)^{\pi^1} \parallel (\mathcal{G}^2)^{\pi^2} \parallel \cdots$, that is, using PA composition. We

show, however, that we can also synthesise strategies π for a composed game $\mathcal{G}^1 \parallel \mathcal{G}^2 \parallel \cdots$, which, due to our compositional procedure, is of the form $\pi = \pi^1 \parallel \pi^2 \parallel \cdots$, but other ways of composing strategies are possible, see, for example, the strategy automata composition in [69].

We note that our assume-guarantee framework presents a general paradigm of obtaining strategy synthesis rules, which is not confined to the stochastic games that we use here. Theorem 4.6 can be restated for any type of nondeterministic system where we want to find strategies resolving the controllable nondeterminism: we need respective composition operators $\parallel$ for the systems and the strategies, and verification rules for the systems without the controllable nondeterminism, and we can similarly obtain a lifting theorem. For example, the framework can be used for finding strategies for coalitions in multi-player games in the style of [124], as well as for assume-guarantee strategy *synthesis* for PAs, based on verification rules of DTMCs. We further note that the only condition on specifications that we impose is that they are defined on traces, yielding a general class, while the rest of the thesis demonstrates strategy synthesis for specific examples of such specifications.

Our framework has several shortcomings. The compatibility condition requires that the full product game is constructed in order to be checked. For systems consisting of many small components, checking compatibility could become the dominating factor in computational complexity. However, for homogeneous components, an inductive argument might be able to establish compatibility without explicitly constructing the composition, showing in the inductive step compatibility of $(\|_{j \in J}\ \mathcal{G}^j)$ and $\mathcal{G}^i$ for $i \notin J \subset I$. Furthermore, our synthesis rules are not complete, meaning that a strategy may exist for the global specification, while it is not possible to find a strategy compositionally. Even if the underlying PA verification rules are complete, the strategies synthesised compositionally base their decisions on the local memory only. The advantage of this is, however, that the strategies can be implemented locally on the component games, and still achieve the global specification.

The (Circ) rule offers some flexibility to develop bidirectional interfaces. In general, we would like to consider components $\mathcal{G}^1$ and $\mathcal{G}^2$, where we require that each game provides a guarantee to the other game, formalised in corresponding specifications φ_1 and φ_2, respectively. For example, φ_1 is "$\mathcal{G}^1$ provides energy to $\mathcal{G}^2$", and φ_2 is "$\mathcal{G}^2$ pays $\mathcal{G}^1$ for the received energy". Each game can only fulfil its guarantee by assuming the other game fulfils its guarantee, and so we want local strategies for $\mathcal{G}^1 \models \varphi_1 \to \varphi_2$ and $\mathcal{G}^2 \models \varphi_2 \to \varphi_1$. In the current framework we can conclude $\mathcal{G}^1 \parallel \mathcal{G}^2 \models (\varphi_1 \vee \varphi_2) \to (\varphi_1 \wedge \varphi_2)$, but, ideally, we would like to find global strategies

for $\mathcal{G}^1 \parallel \mathcal{G}^2 \models \varphi_1 \wedge \varphi_2$. Development of appropriate side conditions in this setting is subject of ongoing research.

By computing compositional Pareto sets, a designer can pick the targets of given specifications. However, it still remains an open problem how to automatically derive appropriate local specifications in our assume-guarantee syntehsis rules, given the global specification. We note also that if we want a specification $\mathsf{Eratio}(r_G/c)(v_G)$ to be satisfied for a composed system $\mathcal{G}^1 \parallel \mathcal{G}^2$, we would want to employ the (ASYM) rule and find a reward structure r_A such that $\mathcal{G}^1 \models \mathsf{Eratio}(r_A/c)(v_A)$ and $\mathcal{G}^2 \models \mathsf{Eratio}(r_A/c)(v_A) \rightarrow \mathsf{Eratio}(r_G/c)(v_G)$ for some target v_A. In this case the objective $\mathsf{Eratio}(r_A/c)(v_A)$ does not need to carry semantic significance, as it is only establishing a contract between $\mathcal{G}^1$ and $\mathcal{G}^2$. For verification of PAs, some research has been done to learn the local specifications [44, 62, 106]. Another promising approach in this direction in the verification domain is finding premises for the rules using abstraction refinement [80]. While counterexample-guided abstraction refinement for stochastic games has been considered, for example, in [30], this approach is not directly applicable to our setting, because multi-objective games are in general not determined, and so counterexamples (that is, spoiler strategies of Player 2) cannot be straightforwardly derived. We observe, however, in Theorem 6.2 in Chapter 6, that for Pmp CQs games are determined and have MD spoiler strategies.

Chapter 5

Multi-Objective Queries

Contents

In the previous chapter we developed an assume-guarantee framework for strategy synthesis, where a key requirement is that we can find winning strategies for the individual components. Moreover, in order to make use of rules such as the asymmetric rule (Asym), or the circular rule (Circ), we need to be able to synthesise multi-objective queries (MQs), containing both conjunctions and disjunctions of objectives, and so we discuss in this chapter the synthesis of strategies for such queries. We focus on long-run objectives, and are specifically concerned with mean-payoffs and ratios of rewards, where we consider both expectations and almost sure satisfaction semantics. For example, in a game $\mathcal{G}$ under the Player 1 strategy π and Player 2 strategy σ, the expectation objective $\mathbb{E}_{\mathcal{G}}^{\pi,\sigma}[\mathsf{mp}(r)] \geq v$ means that the expected mean-payoff of r is above the threshold v; and the almost sure satisfaction objective $\mathbb{P}_{\mathcal{G}}^{\pi,\sigma}(\mathsf{mp}(r) \geq v) = 1$ means that the mean-payoff of r is above the threshold v with probability one, that is, almost surely.

In order to synthesise strategies, we convert the MQs consisting of any type of long-run objectives that we study to conjunctions of Pmp objectives. We develop three main transformation steps. First, we show that ratio objectives for both expectation and satisfaction semantics (Pratio and ratioE) can be converted to the corresponding

mean-payoff objectives (Pmp and Emp, respectively). Then, we introduce a general class of games called *controllable multichain* (CM), for which we show that strategies for conjunctions of expected mean-payoffs (Emp CQs) can be synthesised by considering Pmp CQs instead, with the same rewards. This transformation is complete for ε-optimal strategies for Emp CQs. Note also, from Lemma 3.1 that we can synthesise Eratio objectives soundly, but not completely, using Pratio objectives. Finally, we show how to reduce Boolean combinations of expectation objectives to conjunctions of expectation objectives. To summarise, we can represent each transformation T by an arrow, where an arrow $\Phi \to \Phi'$ means that the same strategy satisfying $\varphi \in \Phi$ satisfies $T(\varphi) \in \Phi'$, and an arrow $\Phi \rightsquigarrow \Phi'$ means that if a strategy satisfies $\varphi \in \Phi$, then there is a strategy satisfying $T(\varphi) \in \Phi'$. We hence obtain the following picture:

$$\mathsf{Pmp}\ \mathrm{CQ} \rightleftarrows \mathsf{Emp}\ \mathrm{CQ} \rightleftarrows \mathsf{Emp}\ \mathrm{MQ} \rightleftarrows \mathsf{ratioE}\ \mathrm{MQ},$$

$$\mathsf{Pmp}\ \mathrm{CQ} \rightleftarrows \mathsf{Pratio}\ \mathrm{CQ} \to \mathsf{Eratio}\ \mathrm{CQ} \rightleftarrows \mathsf{Eratio}\ \mathrm{MQ}.$$

We remark that, for almost sure satisfaction objectives, the reduction from Boolean combinations to conjunctions is not possible in general. For example, for a Pmp objective B, defining an implication $B \to C$ as the disjunction $\neg B \vee C$ means that $\neg B$ is no longer an almost sure satisfaction objective of the form $\mathbb{P}(\mathsf{mp}(r) \geq v) = 1$, but a positive satisfaction objective of the form $\mathbb{P}(\mathsf{mp}(r) < v) > 0$ (see also Example 3.7). Even disjunctions of Pmp objectives without negation, that is, MQs of the form $A \vee B$ are challenging, because a strategy needs to guarantee that either almost all paths satisfy A or almost all paths satisfy B, so we cannot straightforwardly apply arguments that isolate subsets of paths, by, for example, using BSCCs.

We begin the chapter in Section 5.1 by discussing necessary and sufficient classes of Player 1 strategies for the specifications we are considering. We are mostly concerned with synthesising finite-memory strategies, for which stochastic update gives rise to a more succinct representation. Then, in Section 5.2, we discuss ratio objectives in both expectation and satisfaction semantics, and show reductions to mean-payoff. In Section 5.3 we exhibit a class of games for which expectation objectives reduce to almost sure satisfaction objectives under ε-optimality. Section 5.4 is dedicated to showing how to convert Boolean combinations of expectation objectives to conjunctions. Finally, in Section 5.5 we discuss algorithms to approximate Pareto sets.

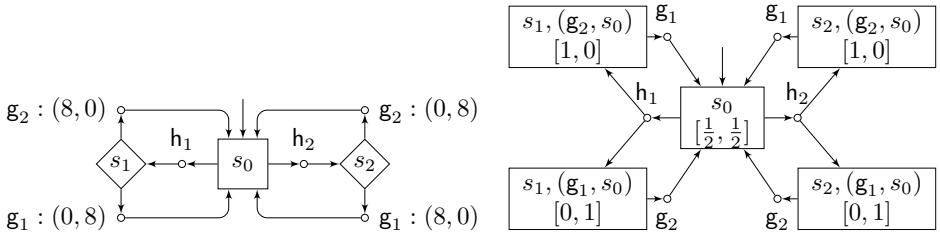


Figure 5.1: Illustrating the hiding of stochastic memory. Stochastic game $\mathcal{G}$ (left) and induced PA $\mathcal{G}^\pi$ (right). The labels $[p_1, p_2]$ show the belief $\mathfrak{d} \in \mathsf{D}(\mathfrak{M})$, where p_i is the probability of $\mathfrak{m}_i$. The reward structure $\vec{r}$ is annotated next to the action names.

5.1 Discussion of Player 1 Strategies

We discuss the classes of strategies that are necessary and sufficient for Player 1 for our objectives, focusing on the role of memory and randomisation. Further, we show that SU strategies can be exponentially more succinct than DU strategies.

5.1.1 Memory and Randomisation

Applying strategies for the respective players to a game resolves all nondeterminism. In [50] three types of strategies are distinguished: *behavioural strategies* select at each step a distribution over moves based on the history; *mixed strategies* randomise initially between several DU strategies that deterministically select moves; and *general strategies* can both randomise initially and on-the-fly. For perfect observation, all three strategy formulations are equally powerful, if infinite memory is allowed [50].

Hiding Memory. The memory of a strategy is internal, in particular, Player 2 does not directly observe the memory of Player 1. However, the strategy σ of Player 2 can depend on the strategy π of Player 1 (due to the order of quantifiers in a synthesis query), and so Player 2 can infer the belief $\mathfrak{d}^\pi_\lambda$ from the history λ, which both players observe. The following example demonstrates the problems if Player 1 reveals its actual memory to Player 2, not just the belief $\mathfrak{d}^\pi_\lambda$.

Example 5.1 (Hiding Memory)**.** *Consider the game $\mathcal{G}$ in Figure 5.1 (left). The PA $\mathcal{G}^\pi$ on the right is induced by the SU strategy π defined by $\mathfrak{M} = \{\mathfrak{m}_1, \mathfrak{m}_2\}$, $\pi_d(s_0)(\mathfrak{m}_i) = \pi_u(\mathfrak{m}_j, s_0)(\mathfrak{m}_i) = \frac{1}{2}$, $\pi_u(\mathfrak{m}_i, s) = \mathfrak{m}_i$, and $\pi_c(s_j, \mathfrak{m}_i) = (g_i, s_0)$, for all $i, j \in \{1, 2\}$ and $s \in S \backslash \{s_0\}$. We can see that Player 1 wins for $\mathsf{Pmp}(\vec{r})(1,1)$ using π, since in the induced PA $\mathcal{G}^\pi$ the choices of Player 2 do not affect the mean-payoff. However, if Player 2 knows the memory of Player 1 at s_0 (not just the belief), it can*

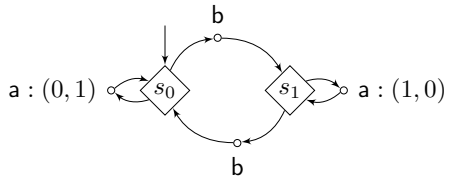


Figure 5.2: Player 1 needs infinite memory for $\mathsf{Pmp}(\vec{r})(\frac{1}{4}, \frac{1}{4})$. Example taken from [18].

pick (h_i, s_i) for $\mathfrak{m}_i$, for all $i \in \{1, 2\}$. Hence, after h_i, Player 1 plays g_i (as promised by the memory $\mathfrak{m}_i$), resulting in a mean-payoff of $(0, 2)$, and so Player 1 loses.

Infinite Memory. In general, infinite memory might be required to achieve a CQ of Pmp objectives. The same requirement is already present for strategy synthesis in MDPs, see [18], and thus carries over to games. We recall the intuition here: in order to achieve $\mathsf{Pmp}(\vec{r})(\frac{1}{4}, \frac{1}{4})$, in the game in Figure 5.2, Player 1 has to play the b-transitions to go between s_0 and s_1, but can only win optimally if the b-transitions do not contribute to the mean-payoff in the limit. Thus, an optimal strategy increases the number of steps between b-transitions and plays the sequence of actions $\mathsf{a}^{2^0}\mathsf{b}\mathsf{a}^{2^0}\mathsf{b}\mathsf{a}^{2^1}\mathsf{b}\mathsf{a}^{2^1}\mathsf{b}\cdots\mathsf{a}^{2^n}\mathsf{b}\mathsf{a}^{2^n}\mathsf{b}\cdots$, which requires infinite memory to keep track of n.

For Emp CQs, the minimum requirements on the Player 1 strategy memory size carries over from reachability objectives (see [41]), since in stopping games we can express the objective to reach a set $T \subseteq \mathsf{Term}$ with probability at least v by $\mathsf{Emp}(r_T)(v)$, where r_T is the reward structure such that $r_T(t) = 1$ if $t \in T$ and $r_T(t) = 0$ otherwise. However, in this thesis we consider Emp objectives only for the subclass of controllable multichain games (see Section 5.3.2), which is incomparable to stopping games.

Randomisation. Randomising between moves might be necessary for expectations, even for ε-optimality in non-stochastic games (see Lemma 14 of [36]). Using randomisation, Player 1 can play ε-optimally in the game in Figure 5.2 without needing memory. Consider the memoryless Player 1 strategy playing b with probability $\alpha > 0$. The resulting DTMC is irreducible, and so we can evaluate the stationary distribution: the probability to be at the self-loops is $\frac{1-\alpha}{4}$ each. Thus, the achieved mean-payoff, both in expectation and almost sure semantics, is $(\frac{1}{4} - \varepsilon, \frac{1}{4} - \varepsilon)$ with $\varepsilon = \frac{\alpha}{4}$.

However, memory is in general necessary for ε-optimality, even if randomisation is allowed. Consider the game in Figure 5.3 (left), with the two-dimensional reward structure $\vec{r}$ given in the figure. For the CQ $\mathsf{Pmp}(\vec{r})$, memoryless strategies of Player 1 can achieve any targets in the Pareto set in Figure 5.3 (right), by randomising between a and b, obtaining rewards $(1, 0)$ and $(0, \frac{1}{2})$ respectively. The Pareto set for $\mathsf{Pmp}(\vec{r})$ under general Player 1 strategies is shown in Figure 5.3 (centre). To achieve

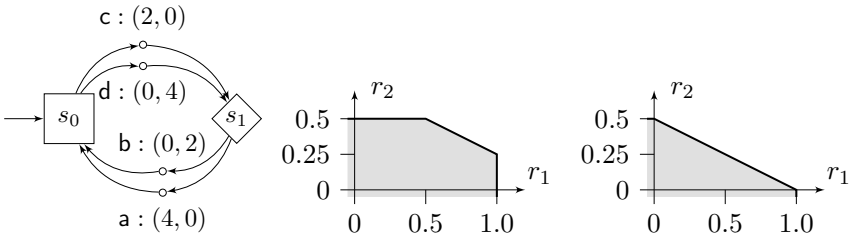


Figure 5.3: Game where the mean-payoff is stationary, that is, it is the same at every state. The Pareto set for $\mathsf{Pmp}(r_1)(v_1) \wedge \mathsf{Pmp}(r_2)(v_2)$ is shown in the middle, and is the same at every state. The Pareto set on the right results if Player 1 plays memoryless.

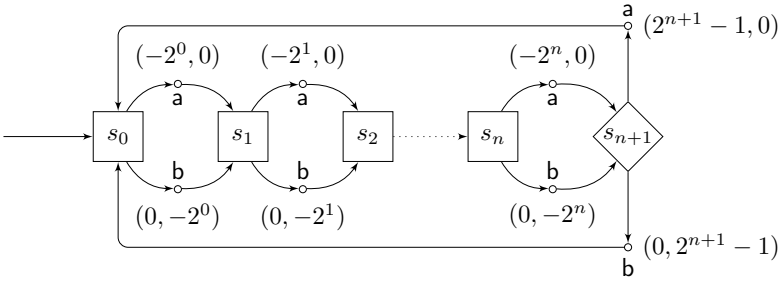


Figure 5.4: Game to illustrate that finite-memory SU strategies can be exponentially more succinct than finite-memory DU strategies for $\mathsf{Pmp}(\vec{r})$.

$\mathsf{Pmp}(\vec{r})(\frac{1}{2}, \frac{1}{2})$, Player 1 plays a following d, and b following c. To achieve $\mathsf{Pmp}(\vec{r})(1, \frac{1}{4})$, Player 1 randomises uniformly between a and b following c, and plays a following d. It is easy to see that, even for ε-optimality of these specifications, Player 1 needs memory. Note that the same example applies to Emp. A similar result for non-stochastic games is established in Lemma 13 of [36].

5.1.2 Succinctness of SU Strategies

In our strategy synthesis algorithm of Section 6.4 and its implementation, presented in Chapter 7, we use SU strategies, and we justify their use by demonstrating that they can be exponentially more succinct than DU strategies.

Proposition 5.1. *Finite SU strategies can be exponentially more succinct than finite DU strategies for expected and almost sure average rewards.*

Proof. We provide the intuition behind the result, and give a full proof in Appendix A.2. The proof method is based on similar results in [36, 124]. Consider the game $\mathcal{G}$ in Figure 5.4 with objective $\mathsf{Pmp}(\vec{r})$. From s_0, when Player 2 chooses a sequence w of actions with $|w| \leq n + 1$, the total rewards are shifted by the vector

$-(\alpha_w, 2^{|w|} - 1 - \alpha_w)$, where $\alpha_w \stackrel{\text{def}}{=} \sum_{j=1}^{|w|} \delta_{w_j=\mathsf{a}} 2^{j-1}$ is the number corresponding to the binary word w represented with the least significant bit first, with a coding for 1 and b for 0. A winning strategy of Player 1 has to compensate this shift by randomising between a and b at s_{n+1}, according to the distribution ν_w defined by $\nu_w(\mathsf{a}) = \frac{\alpha_w}{2^{n+1}-1}$ and $\nu_w(\mathsf{b}) = 1 - \nu_w(\mathsf{a})$, where w is the sequence of actions seen in this iteration of the loop. Then the total expected reward is balanced at $(0, 0)$ at every iteration, and so also the mean-payoff is above $(0, 0)$ almost surely. A winning Player 1 strategy can be constructed by keeping w in memory, requiring 2^{n+1} memory elements. We show that no DU strategy can win with less memory as follows. There would then be a memory element $\mathfrak{m}$ with at least two distinct sequences $w^1_\mathfrak{m}$ and $w^2_\mathfrak{m}$ corresponding to $\mathfrak{m}$, for which the accumulated reward differs. Thus, Player 1 can only compensate for either sequence $w^1_\mathfrak{m}$ or $w^2_\mathfrak{m}$ at s_{n+1}, and so Player 2 can pick the strategy that plays the opposite sequence that Player 1 is not compensating for, ensuring that Player 1 loses. However, we can construct a SU strategy for Player 1 that keeps a distribution over a and b in memory, updating it as Player 2 selects actions, so that at s_{n+1} the distribution between a and b in memory corresponds precisely to ν_w, which is sufficient to win the game. This strategy has only $2(n + 1)$ memory elements. □

5.1.3 Finite Strategies

When synthesising strategies that we can implement as controllers for autonomous systems (see Chapter 8), we are interested in finite strategies for Player 1. Operating with finite instead of infinite-memory strategies for both players means that the induced DTMCs $\mathcal{G}^{(\pi,\sigma)}$ are finite, allowing us to apply classical results for finite DTMCs. The following lemma characterises the long-run behaviour in finite DTMCs.

Lemma 5.2. *Given a finite DTMC $\mathcal{D}$ with BSCCs $\mathfrak{B}$, and a reward structure $\vec{r}$, for $\lambda \in \Omega_\mathcal{D}$ the limit $\lim_{N\to\infty} \frac{1}{N+1}\mathsf{rew}^N(\vec{r})(\lambda)$ almost surely exists and takes values $\vec{x}$ in the finite set $\{\mathsf{mp}(\vec{r})(\mathcal{B}) \mid \mathcal{B} \in \mathfrak{B}\}$ with probability $\sum_{\mathcal{B} \text{ s.t. } \mathsf{mp}(\vec{r})(\mathcal{B})=\vec{x}} \mathbb{P}_\mathcal{D}(\mathsf{F}\,\mathcal{B})$.*

Proof. Note first that, for every path $\lambda \in \Omega_\mathcal{D}$, $\frac{1}{N+1}\mathsf{rew}^N(\vec{r})(\lambda)$ converges if and only if, for every suffix λ' of λ, $\frac{1}{N+1}\mathsf{rew}^N(\vec{r})(\lambda')$ converges to the same limit. For every recurrent state t of $\mathcal{D}$, we denote by W_t the set of paths λ such that t is the first recurrent state along λ. Paths $\lambda \in W_t$ have suffixes λ' distributed according to $\mathbb{P}_{\mathcal{D},t}$, and by Lemma 3.5, $\frac{1}{N+1}\mathsf{rew}^N(\vec{r})(\lambda')$ almost surely converges to $\sum_{t' \in \mathcal{B}} \mu_\mathcal{B}(t')\vec{r}(t')$. Thus, with probability $\mathbb{P}_\mathcal{D}(\mathsf{F}\,\mathcal{B}) = \sum_{t\in\mathcal{B}} \mathbb{P}_\mathcal{D}(W_t)$, the sequence $\frac{1}{N+1}\mathsf{rew}^N(\vec{r})(\lambda)$ converges to $\mathsf{mp}(\vec{r})(\mathcal{B})$. To conclude, it suffices to recall that $\sum_{\mathcal{B}\in\mathfrak{B}} \mathbb{P}_\mathcal{D}(\mathsf{F}\,\mathcal{B}) = 1$. □

Remark 5.3. *Consequently, $\mathcal{D} \models \mathsf{Pmp}(\vec{r})$ if and only if $\mathsf{mp}(\vec{r})(\mathcal{B}) \geq \vec{0}$ for every BSCC $\mathcal{B}$ that is reached with positive probability.*

Further, in some proofs we require finite PAs resulting from applying strategies consecutively as in Definition 3.5. However, inducing a PA using the belief $\mathfrak{d}^{\pi}_{\lambda}$ corresponds to unrolling the game $\mathcal{G}$ into an infinite tree of all its paths, and applying the strategy π. We have already observed that, if the belief space $\{\mathfrak{d}^{\pi}_{\lambda} \mid \lambda \in \Omega^{\mathsf{fin}}_{\mathcal{G}}\}$ from an SU strategy π is infinite, then the induced PA is infinite as well. Therefore, when our proofs require finite PAs, we need to use finite DU strategies for Player 1.

5.2 Ratio Objectives

In Section 4.2.1 we discussed that the assume-guarantee framework is applicable to specifications defined on traces. One example of specifications defined on traces are ratios of rewards, since they allow us to synchronise both the numerator and denominator, as opposed to mean-payoffs, see Example 4.6. We demonstrate in this section that our discussion of Emp and Pmp objectives applies equally to ratio objectives, since we can reduce ratioE to Emp and Pratio to Pmp, and vice versa.

Proposition 5.4. *Given a DTMC $\mathcal{D}$ with reward structures r and c defined on actions $\mathcal{A}_r$ and $\mathcal{A}_c$, respectively, the objectives $\mathsf{Pratio}(r/c)$ and $\mathsf{Eratio}(r/c)$ are defined on $\mathcal{A}_r \cup \mathcal{A}_c$-traces.*

Proof. Fix a DTMC $\mathcal{D} = \langle S, \varsigma, \mathcal{A}, \chi, \Delta \rangle$. Let $\mathfrak{A} = \{\tau\} \cup \mathcal{A}$. Given a path $\lambda \in \Omega_{\mathcal{D}}$, the trace $\lambda_{rc} \stackrel{\text{def}}{=} \mathsf{proj}_{\mathfrak{A} \setminus (\mathcal{A}_r \cup \mathcal{A}_c)}(\lambda) = a_0 a_1 \ldots$ is such that, for all actions a_i, $r(a_i) \neq 0$ or $c(a_i) > 0$. Given an index N and a path $\lambda \in \Omega_{\mathcal{D}}$, we denote by $\lambda^{(N)}$ the prefix of λ of length N. We have for all odd N that $\mathsf{rew}^N(r)(\lambda) = \mathsf{rew}^{N'}(r)(\lambda_{rc})$, and $\mathsf{rew}^N(c)(\lambda) = \mathsf{rew}^{N'}(c)(\lambda_{rc})$, where $N' = |(\lambda^{(N)})_{rc}|$. Hence

$$\mathsf{rew}^N(r)(\lambda)/\mathsf{rew}^N(c)(\lambda) = \mathsf{rew}^{N'}(r)(\lambda_{rc})/\mathsf{rew}^{N'}(c)(\lambda_{rc}).$$

If λ contains an infinite number of actions in $\mathcal{A}_r \cup \mathcal{A}_c$, then, as $N \to \infty$, also $N' \to \infty$. Hence, we have that $\mathsf{ratio}(r/c)(\lambda)$ is equal to

$$\underline{\lim}_{N\to\infty} \frac{\mathsf{rew}^N(r)(\lambda)}{1 + \mathsf{rew}^N(c)(\lambda)} = \underline{\lim}_{N\to\infty} \frac{\mathsf{rew}^N(r)(\lambda_{rc})}{1 + \mathsf{rew}^N(c)(\lambda_{rc})} \stackrel{\text{def}}{=} \mathsf{ratio}(r/c)(\lambda_{rc}).$$

If λ contains finitely many actions in $\mathcal{A}_r \cup \mathcal{A}_c$, then $\mathsf{ratio}(r/c)(\lambda) = \mathsf{ratio}(r/c)(\lambda_{rc}) = 0$. Hence, for any DTMC $\mathcal{D}'$ that is $\mathcal{A}_r \cup \mathcal{A}_c$-trace equivalent to $\mathcal{D}$, $\mathbb{P}_{\mathcal{D}}(\mathsf{ratio}(r/c) \geq v) = \mathbb{P}_{\mathcal{D}'}(\mathsf{ratio}(r/c) \geq v)$. Thus Pratio and Eratio are defined on $\mathcal{A}_r \cup \mathcal{A}_c$-traces.

□

We now show, in Proposition 5.5, that Pmp can express Pratio, and that Emp can express ratioE. The same reduction does not apply to expectations of ratios.

Proposition 5.5. *Given a finite DTMC $\mathcal{D}$, the following hold:*

(i) $\mathcal{D}$ satisfies $\mathsf{Pratio}(r/c)(v)$ if and only if it satisfies $\mathsf{Pmp}(r - c \cdot v)(0)$; and

(ii) $\mathcal{D}$ satisfies $\mathsf{ratioE}(r/c)(v)$ if and only if it satisfies $\mathsf{Emp}(r - c \cdot v)(0)$.

Proof. Fix a finite DTMC $\mathcal{D}$. By Lemma 5.2, the limit inferior can be replaced by the true limit in $\mathsf{mp}(r)$ and $\mathsf{mp}(c)$. For Pratio, we first show that, for almost every path $\lambda \in \Omega_{\mathcal{D}}$, $\mathsf{ratio}(r/c)(\lambda) = \frac{\mathsf{mp}(r)(\lambda)}{\mathsf{mp}(c)(\lambda)}$. Using the conditions on c imposed by the definition of ratio rewards, we have that, with probability one, $\mathsf{mp}(c) > 0$. Hence,

$$\frac{\mathsf{mp}(r)(\lambda)}{\mathsf{mp}(c)(\lambda)} = \frac{\lim_{N\to\infty} \frac{1}{N+1}\mathsf{rew}^N(r)(\lambda)}{\lim_{N\to\infty} \frac{1}{N+1}\mathsf{rew}^N(c)(\lambda)} = \lim_{N\to\infty} \frac{\frac{1}{N+1}\mathsf{rew}^N(r)(\lambda)}{\frac{1}{N+1}\mathsf{rew}^N(c)(\lambda)}.$$

There is no indeterminacy for this quotient of limits, as the denominator is positive and both numerator and denominator are bounded. Simplifying the $\frac{1}{N+1}$ term yields the equality $\frac{\mathsf{mp}(r)(\lambda)}{\mathsf{mp}(c)(\lambda)} = \lim_{N\to\infty} \frac{\mathsf{rew}^N(r)(\lambda)}{\mathsf{rew}^N(c)(\lambda)}$. This is almost surely equal to $\mathsf{ratio}(r/c)(\lambda) = \lim_{N\to\infty} \frac{\mathsf{rew}^N(r)(\lambda)}{1+\mathsf{rew}^N(c)(\lambda)}$ since $\mathsf{rew}^N(c)(\lambda) \to +\infty$ almost surely. Then, for any v, it follows that $\mathsf{mp}(r)(\lambda) - v \cdot \mathsf{mp}(c)(\lambda) \geq 0$ holds almost surely exactly when $\mathsf{ratio}(r/c)(\lambda) = \frac{\mathsf{mp}(r)(\lambda)}{\mathsf{mp}(c)(\lambda)} \geq v$ holds almost surely.

For ratioE, we have, by Theorem 3.2, that $\mathbb{E}_{\mathcal{D}}[\mathsf{mp}(r)]/\mathbb{E}_{\mathcal{D}}[\mathsf{mp}(c)] \geq v$ if and only if $\mathbb{E}_{\mathcal{D}}[\mathsf{mp}(r)] \geq \mathbb{E}_{\mathcal{D}}[\mathsf{mp}(c)] \cdot v$, if and only if $\mathbb{E}_{\mathcal{D}}[\mathsf{mp}(r - c \cdot v)] \geq 0$. □

Example$^{\mathsf{re}}$ 5.2. *Consider the specification φ^1_{re}, φ^2_{re} and φ_{re} for our running example in Section 3.6, which are defined using Eratio objectives. If instead they were defined using ratioE objectives (which, however, are not defined on traces), using Proposition 5.5 (ii), we could equivalently consider $\mathsf{Emp}^{\leq}(r_1 - c \cdot v_1) \to \mathsf{Emp}(r_2 - c \cdot v_2)$ for φ^1_{re}, $\mathsf{Emp}^{\leq}(r_1 - c \cdot v_1) \wedge \mathsf{Emp}^{\leq}(r_3 - c \cdot v_3)$ for φ^2_{re}, and $\mathsf{Emp}(r_2 - c \cdot v_2) \wedge \mathsf{Emp}^{\leq}(r_3 - c \cdot v_3)$ for φ_{re}.*

5.3 Expectation Objectives

We discuss in this section objectives under the expectation semantics, in particular Emp objectives. We can then also synthesise ratioE objectives, using the transformation in Proposition 5.5 (ii). In order to synthesise strategies for $\mathsf{Emp}(\vec{r})$, we synthesise strategies for $\mathsf{Pmp}(\vec{r})$. This conversion is sound, since any strategy satisfying a property almost surely also satisfies the property in expectation, see Lemma 3.1. For completeness, we need to show that, whenever $\mathsf{Emp}(\vec{r})$ is achievable by π, we can find a strategy $\underline{\pi}$ achieving $\mathsf{Pmp}(\vec{r})$. The key idea is the following. We observe that, in order to achieve $\mathsf{Emp}(\vec{r})$, π induces one or more MECs in $\mathcal{G}^\pi$, and there is some distribution γ between these MECs. We thus construct the strategy $\underline{\pi}$ so that it induces a single MEC in $\mathcal{G}^{\underline{\pi}}$, within which the same expectation is attained as π achieves by randomising between the MECs in $\mathcal{G}^\pi$. The following lemma allows us to conclude that, within MECs, expectation semantics is equivalent to almost sure semantics.

Lemma 5.6. *If a PA contains only one MEC, then it achieves $\mathsf{Emp}(\vec{r})$ against finite strategies if and only if it achieves $\mathsf{Pmp}(\vec{r})$ against finite strategies.*

Proof. If $\mathsf{Pmp}(\vec{r})$, then $\mathsf{Emp}(\vec{r})$, by Lemma 3.1. We show the other direction by contraposition. If $\mathsf{Pmp}(\vec{r})$ does not hold in a PA $\mathcal{M}$ with a single MEC, then there exists a finite strategy σ, such that $\mathbb{P}^\sigma_\mathcal{M}(\mathsf{mp}(r_i) < 0) > 0$ for some i. By Lemma 5.2, there exists a BSCC $\mathcal{B}$ in the induced DTMC $\mathcal{M}^\sigma$ such that $\mathsf{mp}(r_i)(\mathcal{B}) < 0$. By Lemma 3.8, the set of states of the PA corresponding to the BSCC, formally given by $\mathcal{B}_\mathcal{M} \stackrel{\text{def}}{=} \{s \mid \exists \mathfrak{m} \,.\, (s, \mathfrak{m}) \in \mathcal{B}\}$, is reachable with probability one by an MD strategy from all states in $\mathcal{M}$. Hence, the strategy σ' that first reaches $\mathcal{B}_\mathcal{M}$ and then plays as σ to form the BSCC $\mathcal{B}$ is finite and induces a DTMC with a single BSCC $\mathcal{B}'$ in which the mean-payoff is $\mathsf{mp}(r_i)(\mathcal{B}') = \mathsf{mp}(r_i)(\mathcal{B}) < 0$. By Lemma 5.2, we have $\mathbb{P}^{\sigma'}_\mathcal{M}(\mathsf{mp}(r_i) = \mathsf{mp}(r_i)(\mathcal{B})) = \mathbb{P}^{\sigma'}_\mathcal{M}(\mathsf{F}\,\mathcal{B}) = 1$. Thus $\mathbb{P}^{\sigma'}_\mathcal{M}(\mathsf{mp}(r_i) < 0) = 1$, and hence $\mathbb{E}^\sigma_\mathcal{M}[\mathsf{mp}(r_i)] < 0$. We conclude that $\mathsf{Emp}(\vec{r})$ does not hold when $\mathsf{Pmp}(\vec{r})$ does not. □

The MEC $\mathcal{E}'$ is induced by $\underline{\pi}$ cycling between the MECs of $\mathcal{G}^\pi$, staying in each MEC $\mathcal{E}$ of $\mathcal{G}^\pi$ a number of steps proportional to $\gamma(\mathcal{E})$. We define below the class of *controllable multichain* (CM) games, where we can construct such strategies cycling between MECs. The idea underlying the definition of CM games is to make all the MECs of an induced PA almost surely reachable from each other.

5.3.1 MEC-Distribution for Emp CQs

We now characterise the distribution γ between MECs of the PA $\mathcal{M}$ induced by a Player 1 strategy that achieves a CQ $\mathsf{Emp}(\vec{r})$. Moreover, in each MEC $\mathcal{E} = (S_\mathcal{E}, \Delta_\mathcal{E})$ of $\mathcal{M}$, Player 1 achieves some mean-payoff $\vec{\mathsf{z}}^\mathcal{E}$, defined by

$$\mathsf{z}_i^\mathcal{E} \stackrel{\text{def}}{=} \min_{t \in S_\mathcal{E}} \inf_\sigma \mathbb{E}^\sigma_{\mathcal{E},t}[\mathsf{mp}(r_i)] = \min_{t \in S_\mathcal{E}} \inf_\sigma \mathbb{E}^\sigma_{\mathcal{E},t}[\underline{\lim}_{N\to\infty} \tfrac{1}{N} \mathsf{rew}^{N-1}(r_i)] \tag{5.1}$$

for all i. We show below that, for every ε, there exists an N such that $\mathsf{rew}^{N-1}(\vec{r})/N$ stays above the threshold $\vec{\mathsf{z}}^\mathcal{E} - \varepsilon$, independently of the Player 2 strategy σ and of the starting state t. Lemma 5.9 below ensures that we can safely interchange the quantification over σ, t, and N. We first show the following lemma, used in Lemmas 5.9 and 5.15 to bound the rewards acquired by the strategies constructed there.

Lemma 5.7. *Let $\mathcal{D}$ be a DTMC, let $b \geq 0$, let $(c_K)_{K\geq 0}$ be a sequence of positive reals, and let $(X_K)_{K\geq 0}$, $(Y_K)_{K\geq 0}$, $(Z_K)_{K\geq 0}$ be sequences of real-valued random variables on $\Omega_\mathcal{D}$ such that $Z_K \geq 0$, $|X_K| \leq b \cdot c_K$, and $|Y_K| \leq b \cdot Z_K$. Then*

$$\left| \mathbb{E}_\mathcal{D}\left[\frac{X_K + Y_K}{c_K + Z_K}\right] - \mathbb{E}_\mathcal{D}\left[\frac{X_K}{c_K}\right] \right| \leq \frac{2b}{c_K} \mathbb{E}_\mathcal{D}[Z_K].$$

Proof. From the assumptions of the lemma, we obtain

$$\begin{aligned}
\left| \mathbb{E}_\mathcal{D}\left[\frac{X_K + Y_K}{c_K + Z_k}\right] - \mathbb{E}_\mathcal{D}\left[\frac{X_K}{c_K}\right] \right| &= \left| \mathbb{E}_\mathcal{D}\left[\frac{Y_K}{c_K + Z_K}\right] - \mathbb{E}_\mathcal{D}\left[\frac{X_K \cdot Z_K}{c_K(c_K + Z_K)}\right] \right| \\
&\leq \mathbb{E}_\mathcal{D}\left[\frac{|Y_K|}{c_K + Z_K}\right] + \mathbb{E}_\mathcal{D}\left[\frac{|X_K| \cdot Z_K}{c_K(c_K + Z_K)}\right] \\
&\leq \mathbb{E}_\mathcal{D}\left[\frac{b \cdot Z_K}{c_K}\right] + \mathbb{E}_\mathcal{D}\left[\frac{b \cdot c_K \cdot Z_K}{c_K^2}\right] \\
&\leq \frac{2b}{c_K} \mathbb{E}_\mathcal{D}[Z_K].
\end{aligned}$$

□

Lemma 5.8. *Let $\mathcal{G}$ be a game with states S and with minimum non-zero probability $p_{\min}$. For any $s, t \in S$, such that t is reachable from s almost surely, the expected number of steps to reach t from s with an MD strategy is at most $|S| \cdot p_{\min}^{-|S|}$.*

Proof. After $|S|$ steps, t is reached from s with probability at least $p^* \stackrel{\text{def}}{=} p_{\min}^{|S|}$, Thus, the expected number of steps to reach t from s is upper-bounded by $N_\triangleright \stackrel{\text{def}}{=} |S|p^* + 2|S|p^*(1-p^*) + 3|S|p^*(1-p^*)^2 + \cdots = |S|/p^*$. □

Lemma 5.9. *Given a finite PA $\mathcal{M}$ with rewards $\vec{r}$, for every MEC $\mathcal{E} = \langle S_\mathcal{E}, \Delta_\mathcal{E}\rangle$ of $\mathcal{M}$ it holds that $\underline{\lim}_{N\to\infty} \min_{t\in S_\mathcal{E}} \inf_\sigma \mathbb{E}^\sigma_{\mathcal{E},t}[\frac{1}{N}\mathsf{rew}^{N-1}(\vec{r})] \geq \vec{z}^\mathcal{E}$.*

Proof. Fix a MEC $\mathcal{E} = (S_\mathcal{E}, \Delta_\mathcal{E})$ of a finite PA $\mathcal{M} = \langle S, (S_\Box, S_\bigcirc), \varsigma, \mathcal{A}, \chi, \Delta\rangle$. Denote by $p_{\min}$ the minimum non-zero probability in $\mathcal{M}$, and let $\rho^* \stackrel{\text{def}}{=} \max_{s\in S,i} |r_i(s)|$. Assume for the sake of contradiction that there exists $\delta > 0$ and i such that

$$\underline{\lim}_{N\to\infty} \min_{t\in S_\mathcal{E}} \inf_\sigma \mathbb{E}^\sigma_{\mathcal{E},t}\left[\frac{\mathsf{rew}^{N-1}(r_i)}{N}\right] < \mathsf{z}^\mathcal{E}_i - \delta.$$

In particular, we can fix $N \geq \lfloor 2\rho^* \cdot |S_\mathcal{E}| \cdot p_{\min}^{-|S_\mathcal{E}|}\delta^{-1}\rfloor$, $t \in S_\mathcal{E}$, and σ, such that

$$\mathbb{E}^\sigma_{\mathcal{E},t}\left[\frac{\mathsf{rew}^{N-1}(r_i)}{N}\right] < \mathsf{z}^\mathcal{E}_i - \delta.$$

We show that there exists a strategy σ', such that $\mathbb{E}^{\sigma'}_{\mathcal{E},t}[\mathsf{mp}(\vec{r}_i)] < \mathsf{z}^\mathcal{E}_i$, that is, σ' contradicts the definition of $\mathsf{z}^\mathcal{E}_i$. From Lemma 5.8, we have that $|S_\mathcal{E}| \cdot p_{\min}^{-|S_\mathcal{E}|}$ is an upper bound for the expected number of steps to reach t from s for MD strategies. We construct the strategy σ' as follows. Starting from t, σ' plays in a first phase the N first steps of σ, then plays in a second phase an MD strategy to reach t, and then repeats these two phases ad infinitum. For a path $\lambda \in \Omega_\mathcal{M}$, let $N^{(K)}(\lambda)$ be the index of the beginning of the Kth such loop, and $+\infty$ if λ contains no loops. We have

$$\begin{aligned}
\mathbb{E}^{\sigma'}_{\mathcal{E},t}[\mathsf{mp}(r_i)] &\stackrel{\text{def}}{=} \mathbb{E}^{\sigma'}_{\mathcal{E},t}\left[\underline{\lim}_{k\to\infty}\frac{1}{k+1}\mathsf{rew}^k(r_i)\right] \\
&\leq \mathbb{E}^{\sigma'}_{\mathcal{E},t}\left[\underline{\lim}_{K\to\infty}\frac{1}{N^{(K)}+1}\mathsf{rew}^{N^{(K)}}(r_i)\right] \quad \text{(subsequence)} \\
&\leq \underline{\lim}_{K\to\infty}\mathbb{E}^{\sigma'}_{\mathcal{E},t}\left[\frac{1}{N^{(K)}+1}\mathsf{rew}^{N^{(K)}}(r_i)\right],
\end{aligned}$$

where the last inequality holds by Fatou's lemma (Theorem 4.21 in [79]). For a path $\lambda \in \Omega_\mathcal{M}$, let $c_K(\lambda) - 1 \stackrel{\text{def}}{=} NK$ and $Z_K(\lambda)$ be the number of steps in the first and second phase, respectively, during the K first loops. Let $X_K(\lambda)$ and $Y_K(\lambda)$ be the respective total reward of r_i. Then, $\mathbb{E}^{\sigma'}_{\mathcal{E},t}\left[\frac{1}{N^{(K)}+1}\mathsf{rew}^{N^{(K)}}(r_i)\right] \stackrel{\text{def}}{=} \mathbb{E}^{\sigma'}_{\mathcal{E},t}\left[\frac{X_K+Y_K}{c_K+Z_K}\right]$, and so from Lemma 5.7 we obtain

$$\mathbb{E}^{\sigma'}_{\mathcal{E},t}\left[\frac{1}{N^{(K)}+1}\mathsf{rew}^{N^{(K)}}(r_i)\right] \leq \mathbb{E}^{\sigma'}_{\mathcal{E},t}\left[\frac{X_K}{c_K}\right] + \frac{2\rho^*}{c_K}\mathbb{E}^{\sigma'}_{\mathcal{E},t}[Z_K]. \tag{5.2}$$

Consider the right-hand side of (5.2). By definition of σ' in the first phase, the first term equals $\frac{K}{1+KN}\mathbb{E}^{\sigma}_{\mathcal{E},t}\left[\mathsf{rew}^{N-1}(r_i)\right]$. The second term is upper-bounded by δ, since $(2\rho^*/c_K)\cdot\mathbb{E}^{\sigma'}_{\mathcal{E},t}[Z_K] \leq (2\rho^*/KN)K\cdot|S_{\mathcal{E}}|\cdot p_{\min}^{-|S_{\mathcal{E}}|} = 2\rho^*\cdot|S_{\mathcal{E}}|\cdot p_{\min}^{-|S_{\mathcal{E}}|}/N \leq \delta$. Thus,

$$\mathbb{E}^{\sigma'}_{\mathcal{E},t}[\mathsf{mp}(r_i)] \leq \underline{\lim}_{K\to\infty}\frac{K}{1+KN}\mathbb{E}^{\sigma}_{\mathcal{E},t}[\mathsf{rew}^{N-1}(r_i)] + \delta = \frac{1}{N}\mathbb{E}^{\sigma}_{\mathcal{E},t}[\mathsf{rew}^{N-1}(r_i)] + \delta < \mathsf{z}_i^{\mathcal{E}}.$$

This contradicts the definition of $\mathsf{z}_i^{\mathcal{E}}$ and the proof is complete. □

Lemma 5.10. *Given a MEC $\mathcal{E}$, and an index i, the value $\inf_\sigma \mathbb{E}^{\sigma}_{\mathcal{E},t}[\mathsf{mp}(r_i)]$ does not depend on t, and is equal to $\mathsf{z}_i^{\mathcal{E}}$.*

Proof. Fix a MEC $\mathcal{E}$ and an index i. Given a state t in $\mathcal{E}$, denote by $\mathcal{E}_t$ a PA with a single MEC $\mathcal{E}$ and initial state t. Consider two states t, t' in $\mathcal{E}$, and a strategy σ in $\mathcal{E}_t$. Now consider the strategy σ in $\mathcal{E}_{t'}$ that first plays MD to reach t almost surely (which is possible within a MEC), and then switches to σ as soon as t is reached for the first time. Then $\mathbb{E}^{\sigma'}_{\mathcal{E},t'}[\mathsf{mp}(r_i)] = \mathbb{E}^{\sigma}_{\mathcal{E},t}[\mathsf{mp}(r_i)]$. Hence, for every t, t', $\inf_{\sigma'}\mathbb{E}^{\sigma'}_{\mathcal{E},t'}[\mathsf{mp}(r_i)] \leq \inf_\sigma \mathbb{E}^{\sigma}_{\mathcal{E},t}[\mathsf{mp}(r_i)]$. Reversing the role of t, t' leads to an equality. □

We are now ready to show Proposition 5.12, the main result in this section. We use the following result to reason about end components.

Lemma 5.11 (Theorem 3.2 of [53])**.** *Given a finite PA $\mathcal{M}$, for any finite strategy σ, with probability one, the states s in $\mathcal{M}$ corresponding to states $(s, \mathfrak{m})$ occurring infinitely often on a path of $\mathcal{M}^\sigma$ form an end component.*

Proposition 5.12. *Let $\mathcal{M}$ be a finite PA for which $\mathsf{Emp}(\vec{r})$ is satisfied and let $\mathfrak{E}$ be the set of MECs in $\mathcal{M}$. Then there exists $\gamma \in D(\mathfrak{E})$ such that $\sum_{\mathcal{E}\in\mathfrak{E}}\gamma(\mathcal{E})\cdot\vec{\mathsf{z}}^{\mathcal{E}} \geq \vec{0}$.*

Proof. Fix a PA $\mathcal{M}$ with MECs $\mathfrak{E}$. Let σ be an arbitrary Player 2 strategy. Given a MEC $\mathcal{E} = (S_{\mathcal{E}}, \Delta_{\mathcal{E}}) \in \mathfrak{E}$, we denote by $\mathcal{E}^{(k)}$ the set of paths that stay forever in $\mathcal{E}$ after the k first steps, and define $\mathcal{E}^{(\infty)} = \bigcup_k \mathcal{E}^{(k)}$. We define the distributions $\gamma^k(\mathcal{E}) \stackrel{\text{def}}{=} \mathbb{P}^{\sigma}_{\mathcal{M}}(\mathcal{E}^{(k)})$ and $\gamma(\mathcal{E}) \stackrel{\text{def}}{=} \mathbb{P}^{\sigma}_{\mathcal{M}}(\mathcal{E}^{(\infty)})$. Note that $(\mathcal{E}^{(k)})_{k\geq 0}$ is a non-decreasing sequence with respect to $\subseteq$, and hence $\gamma^k(\mathcal{E})$ is a non-decreasing sequence that converges toward $\gamma(\mathcal{E})$. By Lemma 5.11, with probability 1, the (control and stochastic) states seen infinitely often along a path form an end component, and hence are included in a MEC. Since MECs are disjoint, a further consequence is that $\sum_{\mathcal{E}\in\mathfrak{E}}\gamma(\mathcal{E}) = 1$.

Now fix $\delta > 0$. Consider for every state s that is in some MEC $\mathcal{E} \in \mathfrak{E}$, and every $\delta > 0$, a δ-optimal strategy $\sigma_{s,\delta}$, that is, such that $\mathbb{E}^{\sigma_{s,\delta}}_{\mathcal{M},s}[\mathsf{mp}(\vec{r})] \leq \vec{\mathsf{z}}^{\mathcal{E}} + \delta$ (which exists

due to Lemma 5.10). Consider further the strategy $\sigma_{k,\delta}$ that plays as σ for the k first steps, and then switches to the δ-optimal strategy $\sigma_{s,\delta}$ if it is at a state s in some MEC, or plays arbitrarily if not in a MEC. Let $\mathsf{F}^{=k}\, T \stackrel{\text{def}}{=} \{\lambda \in \Omega_{\mathcal{D}}^{\mathsf{fin}} \mid \mathsf{last}(\lambda) \in T \wedge |\lambda| = k\}$ be the set of paths reaching T in exactly $k \geq 0$ steps. Let $\rho^* \stackrel{\text{def}}{=} \max_{s \in S,i} |r_i(s)|$. Hence,

$$\vec{0} \leq \mathbb{E}_{\mathcal{M}}^{\sigma_{k,\delta}}[\mathsf{mp}(\vec{r})] \leq \sum_{\mathcal{E} \in \mathfrak{E}} \sum_{s \in S_{\mathcal{E}}} \mathbb{P}_{\mathcal{M}}^{\sigma}(\mathsf{F}^{=k}\, \{s\}) \cdot \mathbb{E}_{\mathcal{M},s}^{\sigma_{s,\delta}}[\mathsf{mp}(\vec{r})] + (1 - \sum_{\mathcal{E} \in \mathfrak{E}} \mathbb{P}_{\mathcal{M}}^{\sigma}(\mathsf{F}^{=k}\, S_{\mathcal{E}})) \cdot \rho^*,$$

where the second term upper bounds the reward contributed by the paths that are not in a MEC after k steps. Let $p_k(\mathcal{E}) \stackrel{\text{def}}{=} \mathbb{P}_{\mathcal{M}}^{\sigma}(\mathsf{F}^{=k}\, S_{\mathcal{E}}) = \sum_{s \in S_{\mathcal{E}}} \mathbb{P}_{\mathcal{M}}^{\sigma}(\mathsf{F}^{=k}\, \{s\})$. Thus,

$$\vec{0} \leq \sum_{\mathcal{E} \in \mathfrak{E}} p_k(\mathcal{E}) \cdot (\vec{\mathsf{z}}^{\mathcal{E}} + \delta) + (1 - \sum_{\mathcal{E} \in \mathfrak{E}} p_k(\mathcal{E})) \cdot \rho^*. \tag{5.3}$$

We now show that $p_k(\mathcal{E}) \to \gamma(\mathcal{E})$ for every $\mathcal{E} \in \mathfrak{E}$. Indeed it holds that

$$\gamma^k(\mathcal{E}) \leq p_k(\mathcal{E}) \leq 1 - \sum_{\mathcal{E}' \neq \mathcal{E}} p_k(\mathcal{E}') \leq 1 - \sum_{\mathcal{E}' \neq \mathcal{E}} \gamma^k(\mathcal{E}'),$$

and the outermost terms converge to the same limit $\gamma(\mathcal{E}) = 1 - \sum_{\mathcal{E}' \neq \mathcal{E}} \gamma(\mathcal{E}')$, and hence so does the inner term $p_k(\mathcal{E})$. Finally, we let $k \to +\infty$ and $\delta \to 0$ in (5.3) to obtain the desired result $\vec{0} \leq \sum_{\mathcal{E} \in \mathfrak{E}} \gamma(\mathcal{E}) \cdot \vec{\mathsf{z}}^{\mathcal{E}}$. □

5.3.2 Controllable Multichain Games

Intuitively, a game is controllable multichain if, from every state in the game, every possible MEC can be reached with probability one. To this end, we define irreducible components of a game, which are the smallest units that need to be reachable by Player 1 from any state in the game. A game $\mathcal{G}$ is *irreducible* if, for all finite DU Player 1 strategies π, the induced PA $\mathcal{G}^\pi$ with states $S_{\mathcal{G}^\pi}$ and transitions $\Delta_{\mathcal{G}^\pi}$ forms a single MEC $(S_{\mathcal{G}^\pi}, \Delta_{\mathcal{G}^\pi})$. A *subgame* $\mathcal{H}$ of a game $\mathcal{G} = \langle S, (S_\Diamond, S_\Box, S_\bigcirc), s_{\mathsf{init}}, \mathcal{A}, \chi, \Delta \rangle$ is a game $\langle S', (S'_\Diamond, S'_\Box, S'_\bigcirc), s'_{\mathsf{init}}, \mathcal{A}, \chi', \Delta' \rangle$, such that $S' \subseteq S$; $S'_\Diamond \subseteq S_\Diamond$; $S'_\Box \subseteq S_\Box$; $S'_\bigcirc \subseteq S_\bigcirc$; $s_{\mathsf{init}} \in S'_\Diamond \cup S'_\Box$; $\chi' \subseteq \chi$; $\Delta' \subseteq \Delta$; and where $s \in S'$ if and only if s is reachable from s'_{init} via Δ'. A subgame $\mathcal{H}$ is *Player 2-closed* if, for all $s, t \in S'_\Box$, $\Delta(s,t) = 1$ implies $\Delta'(s,t) = 1$, and so Player 2 cannot escape from $\mathcal{H}$. An irreducible Player 2-closed subgame of $\mathcal{G}$ is called an *irreducible component* (IC) of $\mathcal{G}$.

Definition 5.1. *A game $\mathcal{G}$ is a* controllable multichain *(CM) game if each IC $\mathcal{H}$ of $\mathcal{G}$ is reachable almost surely from any state $s \in S$ of $\mathcal{G}$.*

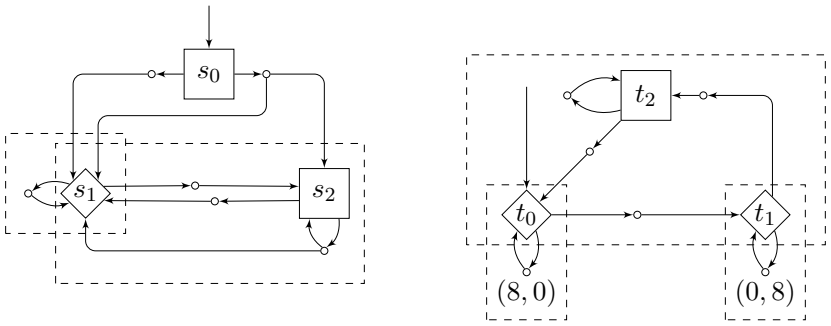


Figure 5.5: A CM game (left) and, a non-CM game (right). Stochastic states have uniform distributions. Irreducible components are annotated using dashed rectangles.

Example 5.3. *Consider the games in Figure 5.5. The game $\mathcal{G}^1$ on the left contains two ICs, consisting of the states $S_1 = \{s_1\}$ and $S_2 = \{s_1, s_2\}$ respectively. Note that the ICs can overlap. S_2 is clearly reached almost surely from s_0, as well as from s_1 and s_2, as these states already are in S_2. Further, S_1 is almost surely reached from s_1. From s_2, the only way to avoid going to S_1 is by taking the self-loop, but the probability to stay in s_2 approaches zero. Thus, S_1 is reached from s_2 and s_0 as well. The game $\mathcal{G}^1$ is therefore CM.*

However, in the game $\mathcal{G}^2$ on the right, the IC containing only t_0 cannot be reached almost surely by Player 1 from t_1, that is, for all π there is some σ such that $\mathbb{P}^{\pi,\sigma}_{\mathcal{G},t_1}(F\{t_0\}) < 1$. Indeed, there can be a strategy achieving a Emp CQ, but not the corresponding Pmp CQ. Consider the reward structure $\vec{r}(t_0) = (8,0)$ and $\vec{r}(t_1) = (0,8)$. Player 1 can achieve $\mathsf{Emp}(\vec{r})(2,2)$ from t_0, but cannot achieve $\mathsf{Pmp}(\vec{r})(2,2)$, not even ε-optimally. Player 1 can achieve $\mathsf{Pmp}(\vec{r})(4,0)$ by looping in t_0, and $\mathsf{Pmp}(\vec{r})(0,4)$ by looping in t_1, but, in order to cycle between t_0 and t_1, Player 1 has to pass through t_2, where Player 2 can decide to stay and spoil $\mathsf{Pmp}(\vec{r})(2,2)$.

The key motivation for defining CM games is that Player 1 can control the MECs to be reached almost surely. In the following lemma, we show that being able to reach ICs in a game $\mathcal{G}$ is necessary and sufficient to reach MECs in the induced PA $\mathcal{G}^\pi$ for any finite DU strategy π, and hence equate the syntactic CM condition of Definition 5.1 with a semantic condition. Given a game $\mathcal{G}$ with states $S_\mathcal{G}$ and a finite DU strategy π, the control states of the induced PA $\mathcal{G}^\pi$ are of the form $(s, \mathfrak{m})$, where $\mathfrak{m}$ is a memory element of π, since π is DU. Given a MEC $\mathcal{E} = (S_\mathcal{E}, \Delta_\mathcal{E})$ of $\mathcal{G}^\pi$, define the set $S_{\mathcal{G},\mathcal{E}} \stackrel{\text{def}}{=} \{s \in S_\mathcal{G} \mid \exists \mathfrak{m} \in \mathfrak{M} \,.\, (s,\mathfrak{m}) \in S_\mathcal{E}\}$ of $\mathcal{G}$-states $S_\mathcal{G}$ occurring in $\mathcal{E}$.

Lemma 5.13. *A game $\mathcal{G}$ is CM if and only if, for every finite DU Player 1 strategy π, for every MEC $\mathcal{E}$ of $\mathcal{G}^\pi$, $S_{\mathcal{G},\mathcal{E}}$ is almost surely reachable from every state of s.*

Proof. "Only if" direction. Fix a CM game $\mathcal{G} = \langle S, (S_\Diamond, S_\Box, S_\bigcirc), \varsigma, \mathcal{A}, \chi, \Delta\rangle$, fix a finite DU Player 1 strategy π, and let $\mathcal{E} = (S_\mathcal{E}, \Delta_\mathcal{E})$ be a MEC of $\mathcal{G}^\pi$. It suffices to show that there exists an IC $\mathcal{H}$ of $\mathcal{G}$ such that $S_\mathcal{H} \subseteq S_{\mathcal{G},\mathcal{E}}$, since by the CM property $S_\mathcal{H}$ is reachable almost surely. We first build a Player 2-closed subgame $\mathcal{H}'$ as follows. Define $\mathcal{H}' \stackrel{\text{def}}{=} \langle S_{\mathcal{G},\mathcal{E}}, (S_{\mathcal{G},\mathcal{E}} \cap S_\Diamond, S_{\mathcal{G},\mathcal{E}} \cap S_\Box, S'_\bigcirc), \varsigma', \mathcal{A}', \chi', \Delta'\rangle$, where $\varsigma' \in \mathsf{D}(S_{\mathcal{G},\mathcal{E}})$ is arbitrary, the move (a,μ) is in $S'_\bigcirc$ whenever there is $\mathfrak{m}$ such that $(a, \mu^\pi_\mathfrak{m})$ is in $\mathcal{E}$ (which also defines χ'), and Δ' is defined by $\Delta'(s,(a,\mu)) = \Delta_\mathcal{E}((s,\mathfrak{m}),(a,\mu^\pi_\mathfrak{m}))$ whenever $s \in S_\Box$ and $\mathfrak{m}$ as before, $\Delta'(s,(a,\mu)) = \Delta_\mathcal{E}((s,(a,\mu),\mathfrak{m}),(a,\mu^\pi_\mathfrak{m}))$ whenever $s \in S_\Diamond$ and $\mathfrak{m}$ as before. Hence, $s \xrightarrow{a} \mu$ in $\mathcal{G}$ whenever $s \xrightarrow{a}{}' \mu$ in $\mathcal{H}'$, and so we have $\Delta' \subseteq \Delta$. Further, since there is a finite path within $\mathcal{E}$ between each $s', t' \in S_\mathcal{E}$, there is also a finite path within $\mathcal{H}'$ between each $s, t \in S_{\mathcal{G},\mathcal{E}}$; hence $\mathcal{H}'$ is a game. Finally, since for a MEC in $\mathcal{G}^\pi$ we require that $\mathsf{supp}(\mu^\pi_\mathfrak{m}) \subseteq S_\mathcal{E}$ whenever $\Delta_\mathcal{E}(s',(a,\mu^\pi_\mathfrak{m})) > 0$, all successors of Player 2 states $S_{\mathcal{G},\mathcal{E}} \cap S_\Box$ must be in $S_{\mathcal{G},\mathcal{E}}$. Hence $\mathcal{H}'$ is Player 2-closed. We now remove all but one choice per Player 1 state in $\mathcal{H}'$, and obtain a subgame $\mathcal{H}''$ of $\mathcal{G}$, which corresponds to an PA, since Player 1 has no longer any choice. Since we remove only Player 1 choices, $\mathcal{H}''$ is still Player 2 closed. A corollary of Lemma 2.2 of [27] is that every nonempty maximal subset of states that is closed under the transition relation of the PA is a MEC. We can thus take any such set of states in $\mathcal{H}''$, which corresponds to a MEC, and thus an irreducible Player 2-closed subgame $\mathcal{H}$ of $\mathcal{G}$.

"If" direction. Fix a game $\mathcal{G}$ and assume that, for every finite DU Player 1 strategy π, for every MEC $\mathcal{E}$ of $\mathcal{G}^\pi$, $S_{\mathcal{G},\mathcal{E}}$ is almost surely reachable from every state of $\mathcal{G}$. Take any IC $\mathcal{H}$ in $\mathcal{G}$. For any Player 1 strategy π', $\mathcal{H}^{\pi'}$ forms a single MEC $\mathcal{E}$. Take the strategy π in $\mathcal{G}$ that plays arbitrarily outside of $\mathcal{H}$, and plays π' upon reaching $\mathcal{H}$. Then $\mathcal{E}$ is also a MEC in $\mathcal{G}^\pi$. By assumption, $S_{\mathcal{G},\mathcal{E}}$ is almost surely reachable from every state of $\mathcal{G}$, and so is $S_\mathcal{H}$, since $S_{\mathcal{G},\mathcal{E}} = S_\mathcal{H}$. Hence $\mathcal{G}$ is CM. □

Checking the CM Property. In general, checking whether a game is CM is in co-NP, since we can guess unreachable ICs in polynomial time.

Theorem 5.14. *The problem of whether a game is CM is in co-NP.*

Proof. A game is not a CM game if it has an IC $\mathcal{H}$ and a state $s \in S_\mathcal{G}$, such that $\mathcal{H}$ is not reachable almost surely from s. We can, in polynomial time: guess such

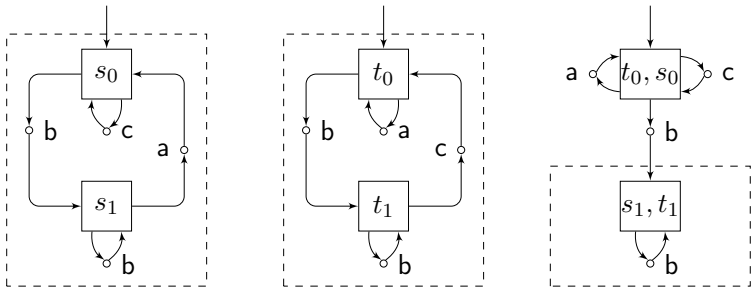


Figure 5.6: Composing CM games yields a non-CM game. ICs are indicated by the rectangular boxes.

a subgame $\mathcal{H}$ and state s; check whether $\mathcal{H}$ is an IC; and check whether $\mathcal{H}$ is not reachable almost surely from s (by Lemma 3.8). Hence, the problem lies in co-NP. □

Composing CM Games. When composing CM games using our composition in Definition 4.3, the result may not be a CM game, which we illustrate with an example.

Example 5.4 (Composing CM Games)**.** *Consider the games in Figure 5.6 (they only consist of Player 2 states and associated moves). Composing the two games on the left yields the game on the right. All three actions* a*,* b *and* c *are synchronised. The games $\mathcal{G}^1$ and $\mathcal{G}^2$ are CM, since the ICs consisting of $\{s_0, s_1\}$ and $\{t_0, t_1\}$ are equivalent to the full state spaces of the respective games $\mathcal{G}^1$ and $\mathcal{G}^2$, and hence reachable from every state. However, in the game on the right, which is the composition $\mathcal{G}^1 \parallel \mathcal{G}^2$, there is one IC containing only the state (s_1, t_1), but it is not reachable from (s_0, t_0), that is, there is no Player 1 strategy such that, for all Player 2 strategies, (s_1, t_1) is reached from (s_0, t_0), since (s_0, t_0) is a Player 2 state. We note that there are back-edges from s_1 to s_0, and from t_1 to t_0, but since they do not synchronise, there is no back-edge from (s_1, t_1) to (s_0, t_0).*

5.3.3 Emp-Pmp Equivalence

While a Player 1 strategy π achieving an Emp objective may randomise between several MECs, a strategy $\underline{\pi}$ for Pmp must be winning in every reached MEC. Given a strategy π achieving $\mathsf{Emp}(\vec{r})$ in a CM game $\mathcal{G}$, we can construct a strategy $\underline{\pi}$ that ε-achieves $\mathsf{Pmp}(\vec{r})$, by inducing a single MEC in $\mathcal{G}^{\underline{\pi}}$, and simulating the distribution over the MECs in $\mathcal{G}^{\pi}$.

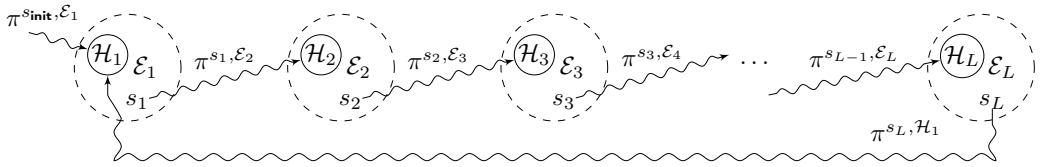


Figure 5.7: The step strategy $\underline{\pi}$ to simulate the probability distribution between MECs $\mathcal{E}_1, \mathcal{E}_2, \ldots, \mathcal{E}_L$ of $\mathcal{G}^\pi$, where we take $\pi^l = \pi$ in Definition 5.2 for simplicity. The MECs are only induced after fixing π, and so each MEC $\mathcal{E}_l$ has some associated IC $\mathcal{H}_l$ in $\mathcal{G}$.

Strategy Construction. We construct $\underline{\pi}$ by looping between MECs, where each MEC $\mathcal{E}_l$ is of a PA $\mathcal{G}^{\pi^l}$ and has an associated finite *step count* N_l. Allowing distinct strategies π^l only slightly complicates our proof of Theorem 5.16, but allows us in Theorem 5.19 to show that Pareto sets for Emp CQs in CM games are convex.

Since $\mathcal{G}$ is CM, from each $s \in S_\mathcal{G}$, each MEC $\mathcal{E}$ can be reached almost surely by an MD strategy $\pi^{s,\mathcal{E}} : S \to S_\bigcirc$ (see Lemma 5.13). We first explain the intuition of our construction of $\underline{\pi}$. We start $\underline{\pi}$ by playing $\pi^{s_{\mathsf{init}},\mathcal{E}_1}$, the MD strategy to reach $\mathcal{E}_1$. As soon as $\mathcal{E}_1$ is reached, $\underline{\pi}$ switches to π^1, which is played for N_1 steps, that is, $\underline{\pi}$ stays inside $\mathcal{E}_1$ for N_1 steps. Then, from whatever state s in $\mathcal{E}_1$ the game is in, $\underline{\pi}$ plays $\pi^{s,\mathcal{E}_2}$, and then in a similar fashion switches to π^2 for N_2 steps within $\mathcal{E}_2$. This continues until $\mathcal{E}_L$ is reached, at which point $\underline{\pi}$ goes back to $\mathcal{E}_1$ again. The strategy $\underline{\pi}$ keeps track in memory of whether it is going from a state s to a MECs $\mathcal{E}$, denoted $s \rhd \mathcal{E}$, or whether it is at a MEC $\mathcal{E}$ and has played j steps, denoted $j@\mathcal{E}$. We emphasise that the strategies are finite DU. See Figure 5.7 for an illustration of $\underline{\pi}$.

Definition 5.2. *Let $\pi^l = \langle \mathfrak{M}^l, \pi^l_c, \pi^l_u, \pi^l_d \rangle$ be finite DU Player 1 strategies, for $1 \leq l \leq L$, with respective MECs $\mathcal{E}_l$ and step counts N_l. The* step strategy *$\underline{\pi}$ is defined as $\langle \underline{\mathfrak{M}}, \underline{\pi}_c, \underline{\pi}_u, \underline{\pi}_d \rangle$, where*

$$\underline{\mathfrak{M}} \stackrel{\text{def}}{=} (\mathfrak{M} \times \{j@\mathcal{E}_l \,|\, l \leq L, j \leq N_l\}) \cup \textstyle\bigcup_{l=1}^{L} \{s \rhd \mathcal{E}_l\},$$

and where, for all $s, t, u \in S_{\mathcal{G}}$, $l \leq L$, $j \leq N_l$, *and* $\mathfrak{m} \in \mathfrak{M}$,

$$\underline{\pi}_{\mathsf{d}}(s) \stackrel{\text{def}}{=} \begin{cases} (s \rhd \mathcal{E}_1) & \text{if } s \notin S_{\mathcal{G},\mathcal{E}_1} \\ (\pi^1_{\mathsf{d}}(s), 0@\mathcal{E}_1) & \text{if } s \in S_{\mathcal{G},\mathcal{E}_1} \end{cases}$$

$$\underline{\pi}_{\mathsf{u}}((s \rhd \mathcal{E}_l), t) \stackrel{\text{def}}{=} \begin{cases} (s \rhd \mathcal{E}_l) & \text{if } t \notin S_{\mathcal{G},\mathcal{E}_l} \\ (\pi^l_{\mathsf{d}}(t), 0@\mathcal{E}_l) & \text{if } t \in S_{\mathcal{G},\mathcal{E}_l} \end{cases}$$

$$\underline{\pi}_{\mathsf{u}}((\mathfrak{m}, j@\mathcal{E}_l), s) \stackrel{\text{def}}{=} \begin{cases} (s \rhd \mathcal{E}_{l'}) & \text{if } j = N_l \text{ and } l' = 1 + (l \mod L) \\ (\pi^l_{\mathsf{u}}(\mathfrak{m}, s), j+1@\mathcal{E}_l) & \text{if } j < N_l \end{cases}$$

$$\underline{\pi}_{\mathsf{c}}(s, (t \rhd \mathcal{E}_l))(u) \stackrel{\text{def}}{=} \pi^{t,\mathcal{E}_l}(s)(u)$$

$$\underline{\pi}_{\mathsf{c}}(s, (\mathfrak{m}, j@\mathcal{E}_l))(t) \stackrel{\text{def}}{=} \pi^l(s, \mathfrak{m})(t).$$

Lemma 5.15 justifies that, for appropriate choices of the step counts N_l, the strategy $\underline{\pi}$ approximates a strategy π, and induces a single MEC in $\mathcal{G}^{\underline{\pi}}$.

Lemma 5.15. *Let* $\mathcal{G}$ *be a CM game, let* π^l *be finite DU strategies, for* $1 \leq l \leq L$, *with associated MECs* $\mathcal{E}_l$ *of* $\mathcal{G}^{\pi^l}$, *and let* $\mathfrak{E}$ *be the set of MECs* $\{\mathcal{E}_l \mid 1 \leq l \leq L\}$. *For all* $\gamma \in \mathsf{D}(\mathfrak{E})$ *and* $\varepsilon > 0$, *there exists a finite DU strategy* $\underline{\pi}$ *such that* $\mathcal{G}^{\underline{\pi}}$ *contains only one MEC, and for all finite* Player 2 *strategies* σ,

$$\mathbb{E}^{\underline{\pi},\sigma}_{\mathcal{G}}[\mathsf{mp}(\vec{r})] \geq \sum_{\mathcal{E} \in \mathfrak{E}} \gamma(\mathcal{E}) \vec{\mathsf{z}}^{\mathcal{E}} - \varepsilon.$$

Proof. Fix a CM game $\mathcal{G}$, finite DU Player 1 strategies π^l for $1 \leq l \leq L$, and the set of L MECs $\mathfrak{E}$, indexed by l, and fix $\gamma \in \mathsf{D}(\mathfrak{E})$. We now pick step counts N_l so that the step strategy $\underline{\pi}$ satisfies the lemma, that is, it ε-approximates γ. From Lemma 5.13, every MEC is almost surely reachable in $\mathcal{G}$ from any state s. Thus, we have an upper bound $N_{\rhd} = |S| \cdot p^*$ on the mean time spent between two MECs. For every l, we define A_l such that, for every $N_l \geq A_l$, $\min_{t \in S_{\mathcal{E}_l}} \inf_\sigma \mathbb{E}^\sigma_{\mathcal{E}_l,t} \left[\mathsf{rew}^{N_l - 1}(\vec{r})\right] \geq N_l(\vec{\mathsf{z}}^{\mathcal{E}_l} - \varepsilon/3)$, which exists by virtue of Lemma 5.9. We now define the step counts for $\underline{\pi}$ by $N_l \stackrel{\text{def}}{=} \lfloor h\gamma(\mathcal{E}_l) \rfloor$, and let $N \stackrel{\text{def}}{=} \sum_{l=1}^{L} N_l$, where we choose h such that

(h1) for every l, $N_l \geq A_l$;

(h2) $1/h \leq \varepsilon / (3 \sum_{l=1}^{L} \|\vec{\mathsf{z}}^{\mathcal{E}_l}\|_\infty)$;

(h3) $(L\gamma(\mathcal{E}_l) + 1)/(h - L) \leq \varepsilon / (3 \sum_{l=1}^{L} \|\vec{\mathsf{z}}^{\mathcal{E}_l}\|_\infty)$; and

(h4) $\frac{1}{N} 2\rho^* L N_{\rhd} \leq \varepsilon/3$,

where $\rho^* \stackrel{\text{def}}{=} \max_{s\in S,i} |r_i(s)|$. For an infinite path $\lambda \in \Omega_{\mathcal{G}}$, we let $N^{(K)}(\lambda)$ be the index of the beginning of the Kth loop, or $+\infty$ if λ has fewer than K loops. For every finite DU strategy σ, for almost every path λ, $N^{(K)}(\lambda)$ is finite for all K, and thus $\lim_{k\to\infty} \frac{1}{k+1}\mathsf{rew}^k(\vec{r})(\lambda) = \lim_{K\to\infty} \frac{1}{N^{(K)}+1}\mathsf{rew}^{N^{(K)}}(\vec{r})(\lambda)$. Hence,

$$\begin{aligned}\mathbb{E}^{\pi,\sigma}_{\mathcal{G}}\left[\mathsf{mp}(\vec{r})\right] &= \mathbb{E}^{\pi,\sigma}_{\mathcal{G}}\left[\lim_{K\to\infty}\frac{1}{N^{(K)}+1}\mathsf{rew}^{N^{(K)}}(\vec{r})\right] \quad \text{(almost sure equality)}\\ &= \lim_{K\to\infty}\mathbb{E}^{\pi,\sigma}_{\mathcal{G}}\left[\frac{1}{N^{(K)}+1}\mathsf{rew}^{N^{(K)}}(\vec{r})\right]. \qquad \text{(Lemma 3.2)}\end{aligned}$$

For a path $\lambda \in \Omega_{\mathcal{D}}$, we denote by $c_K - 1 \stackrel{\text{def}}{=} NK$ and $Z_K(\lambda)$ the number of steps in the MEC phase and the inter-MEC phase, respectively, during the K first loops. We denote by $X_K(\lambda)$ and $Y_K(\lambda)$ the respective total rewards of $\vec{r}$. We are interested in

$$\mathbb{E}^{\pi,\sigma}_{\mathcal{G}}\left[\frac{1}{N^{(K)}+1}\mathsf{rew}^{N^{(K)}}(\vec{r})\right] = \mathbb{E}^{\pi,\sigma}_{\mathcal{G}}\left[\frac{X_K+Y_K}{c_K+Z_K}\right],$$

as $K\to\infty$, and from Lemma 5.7 we get that

$$\mathbb{E}^{\pi,\sigma}_{\mathcal{G}}\left[\frac{1}{N^{(K)}+1}\mathsf{rew}^{N^{(K)}}(\vec{r})\right] \geq \mathbb{E}^{\pi,\sigma}_{\mathcal{G}}\left[\frac{X_K}{c_K}\right] - \frac{2\rho^*}{c_K}\mathbb{E}^{\pi,\sigma}_{\mathcal{G}}[Z_K]. \tag{5.4}$$

We let $X_{l,k}(\lambda)$ be the reward accumulated in the lth MEC phase during the kth loop, and thus have $X_K = \sum_{k=0}^{K-1}\sum_{l=1}^{L} X_{l,k}$. By virtue of (h1), $N_l \geq A_l$, and hence it holds that $\mathbb{E}^{\pi,\sigma}_{\mathcal{G}}[X_{l,k}] \geq N_l(\vec{\mathsf{z}}^{\mathcal{E}_l} - \varepsilon/3)$. Therefore,

$$\mathbb{E}^{\pi,\sigma}_{\mathcal{G}}\left[\frac{X_K}{c_K}\right] \geq \frac{1}{1+KN}\sum_{k=0}^{K-1}\sum_{l=1}^{L} N_l(\vec{\mathsf{z}}^{\mathcal{E}_l} - \frac{\varepsilon}{3}) \geq \left(\frac{K}{1+KN}\sum_{l=1}^{L} N_l\vec{\mathsf{z}}^{\mathcal{E}_l}\right) - \frac{\varepsilon}{3},$$

and taking the limit, we get

$$\lim_{K\to\infty}\mathbb{E}^{\pi,\sigma}_{\mathcal{G}}\left[\frac{X_K}{c_K}\right] \geq \sum_{l=1}^{L}\frac{N_l}{N}\vec{\mathsf{z}}^{\mathcal{E}_l} - \frac{\varepsilon}{3} \geq \sum_{l=1}^{L}\gamma(\mathcal{E}_l)\vec{\mathsf{z}}^{\mathcal{E}_l} - \sum_{l=1}^{L}\left|\gamma(\mathcal{E}_l) - \frac{N_l}{N}\right|\cdot\|\vec{\mathsf{z}}^{\mathcal{E}_l}\|_\infty - \frac{\varepsilon}{3}.$$

We can upper- and lower-bound the term $\frac{N_l}{N}$ by noting that

$$\begin{aligned}\frac{N_l}{N} &\geq \frac{h\gamma(\mathcal{E}_l)-1}{\sum_{l'=1}^{L} h\gamma(\mathcal{E}_{l'})} && \geq \gamma(\mathcal{E}_l) - \frac{1}{h},\\ \frac{N_l}{N} &\leq \frac{h\gamma(\mathcal{E}_l)+1}{\sum_{l'=1}^{L}(h\gamma(\mathcal{E}_{l'})-1)} = \frac{h\gamma(\mathcal{E}_l)+1}{h-L} && = \gamma(\mathcal{E}_l) + \frac{1}{h-L}(L\gamma(\mathcal{E}_l)+1).\end{aligned}$$

By conditions (h2) and (h3) for h, we get $\left|\gamma(\mathcal{E}_l) - \frac{N_l}{N}\right| \leq \varepsilon/(3\sum_{l'=1}^{L} \|\vec{\mathsf{z}}^{\mathcal{E}_{l'}}\|)$, and so

$$\lim_{K\to\infty} \mathbb{E}_{\mathcal{G}}^{\underline{\pi},\sigma}\left[\frac{X_K}{c_K}\right] \geq \sum_{l=1}^{L} \gamma(\mathcal{E}_l)\vec{\mathsf{z}}^{\mathcal{E}_l} - \frac{2\varepsilon}{3}. \tag{5.5}$$

We now upper-bound the absolute value of the second term of (5.4) using

$$\frac{2\rho^*}{c_K}\mathbb{E}_{\mathcal{G}}^{\underline{\pi},\sigma}[Z_K] \leq \frac{2\rho^*}{KN}KLN_{\triangleright} = \frac{1}{N}2\rho^* LN_{\triangleright} \leq \frac{\varepsilon}{3}, \tag{5.6}$$

where the last inequality comes from condition (h4) on h and hence on N. Applying the bounds (5.5) and (5.6) to (5.4), we obtain $\mathbb{E}_{\mathcal{G}}^{\underline{\pi},\sigma}[\mathsf{mp}(\vec{r})] \geq \sum_{l=1}^{L} \gamma(\mathcal{E}_l)\vec{\mathsf{z}}^{\mathcal{E}_l} - \varepsilon$. □

We now show, in Theorem 5.16, the main result of this section, that, in CM games, for any $\varepsilon > 0$, we can find a strategy $\underline{\pi}$ that achieves $\mathsf{Pmp}(\vec{r} + \varepsilon)$, whenever $\mathsf{Emp}(\vec{r})$ is achievable by a finite DU strategy. The ε degradation is unavoidable for finite strategies, due to the need for infinite memory in general, as illustrated in Figure 5.2: here, the strategy $\underline{\pi}$ has to minimise the relative impact of the $N_{\triangleright}$ steps between MECs which only vanishes if the step counts N_l go to infinity.

Theorem 5.16. *If, in a CM game $\mathcal{G}$, a finite DU strategy achieves $\mathsf{Emp}(\vec{r})$, then, for all $\varepsilon > 0$, there is a finite DU strategy achieving $\mathsf{Pmp}(\vec{r} + \varepsilon)$.*

Proof. Let $\varepsilon > 0$ and let π be a finite DU strategy such that $\mathcal{G}^{\pi} \models \mathsf{Emp}(\vec{r})$. The induced PA $\mathcal{G}^{\pi}$ contains a set $\mathfrak{E}$ of L MECs. If $L = 1$, we let $\underline{\pi} = \pi$. If, on the other hand, $L > 1$, we construct a strategy $\underline{\pi}$ such that $\mathcal{G}^{\underline{\pi}} \models \mathsf{Emp}(\vec{r} + \vec{\varepsilon})$ as follows. From Proposition 5.12, we obtain a distribution γ such that $\sum_{\mathcal{E}\in\mathfrak{E}} \gamma(\mathcal{E})\vec{\mathsf{z}}^{\mathcal{E}} \geq \vec{0}$. We then apply Lemma 5.15 with $\pi^l = \pi$ to find a strategy $\underline{\pi}$, so that $\mathcal{G}^{\underline{\pi}}$ contains only one MEC, and, for all finite Player 2 strategies σ, it holds that $\mathbb{E}_{\mathcal{G}}^{\underline{\pi},\sigma}[\mathsf{mp}(\vec{r})] \geq \sum_{l=1}^{L} \gamma(\mathcal{E}_l)\vec{\mathsf{z}}^{\mathcal{E}_l} - \varepsilon \geq -\varepsilon$. We conclude that $\underline{\pi}$ achieves $\mathsf{Pmp}(\vec{r} + \varepsilon)$ using Lemma 5.6. □

Expected Ratio Objectives. We briefly discuss the difficulty of synthesis for expected ratio objectives, that is, $\mathsf{Eratio}(r/c)(v) \stackrel{\text{def}}{=} \mathbb{E}_{\mathcal{D}}[\mathsf{ratio}(r/c)] \geq v$. For MDPs, synthesis of one-dimensional Eratio objectives has been discussed, for example, in [135]: the algorithm operates by identifying the MECs in the MDPs, synthesising a strategy for each MEC, and then putting the strategies together. In games, imposing the CM property allows Player 1 to control the MECs accessed. However, the same argument as used in Theorem 5.16 is not applicable to equate Pratio CQs and Eratio CQs.

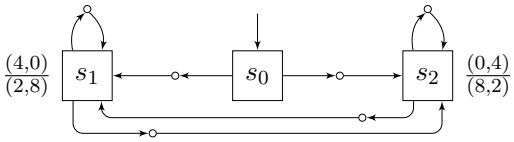


Figure 5.8: CM game illustrating that the proof technique of Theorem 5.16 does not carry over to expected ratio objectives. Reward structures $\vec{r}$ and $\vec{c}$ are shown as $\frac{\vec{r}}{\vec{c}}$ where nonzero.

Consider the CM game in Figure 5.8 with $\vec{r}(s_1) = (4,0)$, $\vec{c}(s_1) = (2,8)$ and $\vec{r}(s_2) = (0,4)$, $\vec{c}(s_2) = (8,2)$. Player 1 can achieve $\mathsf{Eratio}(\vec{r}/\vec{c})(1,1)$ by randomising in s_0 between going to s_1 and s_2 uniformly, and then staying forever in s_1 and s_2 respectively, which corresponds to the distribution γ in Lemma 5.15. However, Player 1 has no strategy to achieve this objective by inducing a single MEC. For example, playing s_1 for 2^n steps, then playing in s_2 for 2^n steps yields, in the limit as $n \to \infty$ a reward of $\frac{2+0}{1+4} = 0.4$ for $\mathsf{Eratio}(r_1/c_1)$, and $\frac{0+2}{1+4} = 0.4$ for $\mathsf{Eratio}(r_2/c_2)$, which is clearly below the required target. The same argument applies for finite ε-optimal strategies.

Note, however, that Eratio CQs can be soundly synthesised by converting them to Pratio CQs using Lemma 3.1, and Eratio MQs can be converted to Eratio CQs as we show in Section 5.4 in Theorem 5.17.

5.4 Boolean Combinations

In this section we show that Boolean combinations of objectives with bounded expected rewards reduce to conjunctions of expected reward objectives. Note that, in a PA $\mathcal{M}$, synthesising strategies satisfying an MQ $\bigvee_j \bigwedge_i \phi_{i,j}$ reduces to CQs straightforwardly: $\exists \sigma \,.\, \mathcal{M}^\sigma \models \bigvee_j \bigwedge_i \phi_{i,j}$ is equivalent to $\exists j \,.\, \exists \sigma \,.\, \mathcal{M}^\sigma \models \bigwedge_i \phi_{i,j}$, for which it is sufficient to consider the CQs $\bigwedge_i \phi_{i,j}$ in isolation for each j. In games, the approach for a Player 1 strategy to achieve $\bigvee_j \bigwedge_i \phi_{i,j}$ is to leave it open to Player 2 to choose the CQ $\bigwedge_i \phi_{i,j}$; Player 1 then has to be able respond (with the already fixed strategy) by satisfying it. Boolean combinations can be converted to conjunctive normal form (CNF), and we show in Theorem 5.17 that MQs in CNF can be converted to CQs.

Theorem 5.17. *Let $\mathcal{G}$ be a game, let $\vec{\varrho}_i : \Omega_\mathcal{G} \to \mathbb{R}^m$ be bounded measurable functions, let $\vec{u}_i \in \mathbb{R}^m$, for $1 \leq i \leq n$, and let π be a Player 1 strategy. There are non-zero* weight *vectors $\vec{x}_1, \ldots \vec{x}_n \in \mathbb{R}^m_{\geq 0}$, such that $\varphi = \bigwedge_{i=1}^n \mathbb{E}_\mathcal{G}^{\pi,\sigma}[\vec{x}_i \cdot \vec{\varrho}_i] \geq \vec{x}_i \cdot \vec{u}_i$ holds for all finite Player 2 strategies σ if and only if $\psi = \bigwedge_{i=1}^n \bigvee_{j=1}^m \mathbb{E}_\mathcal{G}^{\pi,\sigma}[\varrho_{i,j}] \geq u_{i,j}$ holds for all finite Player 2 strategies σ.*

Proof. The proof method is based on a similar result in [41]. Fix a strategy π.

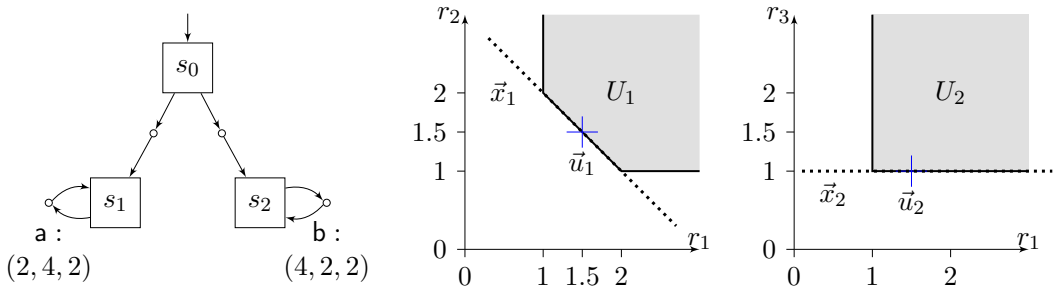


Figure 5.9: Illustrating the necessity of converting MQs to CNF before finding weight vectors. The hyperplanes separating the targets $\vec{u}_i$ (marked by crosses) from U_i are marked as dotted lines, annotated with their normals $\vec{x}_i$, for $i \in \{1, 2\}$.

"If" direction. Assume π achieves ψ. Fix i. Let $U_i \stackrel{\text{def}}{=} \mathsf{upc}(\{\vec{y} \in \mathbb{R}^m \mid \exists \sigma \,.\, \mathbb{E}^{\pi,\sigma}_{\mathcal{G}}[\vec{\varrho}_i] = \vec{y}\})$. This set is convex by A.6 of [18]. Since π achieves ψ, there is a j satisfying $y_j \geq u_{i,j}$ for every $\vec{y} \in U_i$. We have that $\vec{u}_i \notin \mathrm{int}(U_i)$: suppose otherwise; then there is $\epsilon > 0$ such that $\vec{u}_i - \epsilon \in U_i$, contradicting that for all $\vec{y} \in U_i$ there is a j satisfying $y_j \geq u_{i,j}$ (take $\vec{y} = \vec{u}_i - \epsilon$ to derive the contradiction $u_{i,j} - \epsilon \geq u_{i,j}$). By the separating hyperplane theorem (Theorem 11.3 of [114]), there is a non-zero vector $\vec{x}_i \in \mathbb{R}^m$, such that, for all $\vec{w} \in U_i$, $\vec{w} \cdot \vec{x}_i \geq \vec{u}_i \cdot \vec{x}_i$. We now show $\vec{x}_i \geq 0$. Assume for the sake of contradiction that $x_{i,j} < 0$ for some j. Take any $\vec{w} \in U$, let $d = \vec{w} \cdot \vec{x}_i - \vec{u}_i \cdot \vec{x}_i \geq 0$, and let $\vec{w}'$ be the vector obtained from $\vec{w}$ by replacing the jth coordinate with $w_j + \frac{d+1}{-x_{i,j}}$. Since $\frac{d+1}{-x_{i,j}}$ is positive and U_i is upwards closed in $\mathbb{R}^m$, we have $\vec{w}' \in U_i$. So

$$\vec{w}' \cdot \vec{x}_i = \sum_{h=1}^{m} w'_h \cdot x_{i,h} = \frac{d+1}{-x_{i,j}} + \sum_{h=1}^{m} w_h \cdot x_{i,h} = -(d+1) + \vec{w} \cdot \vec{x}_i = \vec{u}_i \cdot \vec{x}_i - 1,$$

implying $\vec{u}_i \cdot \vec{x}_i > \vec{w}' \cdot \vec{x}_i$, which contradicts $\vec{w}' \cdot \vec{x}_i \geq \vec{u}_i \cdot \vec{x}_i$, since $\vec{w}' \in U_i$. Now fix a finite strategy σ. Since $\mathbb{E}^{\pi,\sigma}_{\mathcal{G}}[\vec{\varrho}] \in U_i$, it follows that $\mathbb{E}^{\pi,\sigma}_{\mathcal{G}}[\vec{x}_i \cdot \vec{\varrho}_i] = \vec{x}_i \cdot \mathbb{E}^{\pi,\sigma}_{\mathcal{G}}[\vec{\varrho}_i] \geq \vec{x}_i \cdot \vec{u}_i$.

"Only If" direction. Assume there are non-zero vectors $\vec{x}_1, \ldots \vec{x}_n \in \mathbb{R}^m_{\geq 0}$ such that π achieves φ. Assume for the sake of contradiction that π does not achieve ψ. Fix a finite strategy σ and i such that $\neg(\mathbb{E}^{\pi,\sigma}_{\mathcal{G}}[\varrho_{i,j}] \geq u_{i,j})$ for all j, which exist by assumption. Since $\vec{x}_i$ is such that π achieves φ, we have $\vec{x}_i \cdot \mathbb{E}^{\pi,\sigma}_{\mathcal{G}}[\vec{\varrho}_i] = \mathbb{E}^{\pi,\sigma}_{\mathcal{G}}[\vec{x}_i \cdot \vec{\varrho}_i] \geq \vec{x}_i \cdot \vec{u}_i$. Because $0 \neq \vec{x}_i \in \mathbb{R}^m_{\geq 0}$, there must be a j such that $\mathbb{E}^{\pi,\sigma}_{\mathcal{G}}[\varrho_{i,j}] \geq u_{i,j}$, a contradiction. □

Example 5.5. *Consider the game $\mathcal{G}$ in Figure 5.9 (left). The three-dimensional reward structure $\vec{r}$ is defined as $\vec{r}(\mathsf{a}) = (2, 4, 2)$, $\vec{r}(\mathsf{b}) = (4, 2, 2)$, and zero otherwise. We want to synthesise a strategy achieving the MQ $\psi = \mathsf{Emp}(r_1)(1.5) \vee$*

$(\mathsf{Emp}(r_2)(1.5) \wedge \mathsf{Emp}(r_3)(1))$. *To apply Theorem 5.17, we first convert* φ *to CNF, obtaining* $\psi' = (\mathsf{Emp}(r_1)(1.5) \vee \mathsf{Emp}(r_2)(1.5)) \wedge (\mathsf{Emp}(r_1)(1.5) \vee \mathsf{Emp}(r_3)(1))$.

We thus let $\vec{\varrho}_1 = (\mathsf{mp}(r_1), \mathsf{mp}(r_2))$ *and* $\vec{u}_1 = (1.5, 1.5)$ *for the first conjunct, and* $\vec{\varrho}_2 = (\mathsf{mp}(r_1), \mathsf{mp}(r_3))$ *and* $\vec{u}_2 = (1.5, 1)$ *for the second conjunct. We now want to find non-zero vectors* $\vec{x}_1, \vec{x}_2 \in \mathbb{R}^2_{\geq 0}$ *such that the CQ* $\varphi = \mathsf{Emp}(\vec{x}_1 \cdot \vec{\varrho}_1)(\vec{x}_1 \cdot \vec{u}_1) \wedge \mathsf{Emp}(\vec{x}_2 \cdot \vec{\varrho}_2)(\vec{x}_2 \cdot \vec{u}_2)$ *is achievable. The intuition is that the choices of* $\vec{x}_1$ *and* $\vec{x}_2$ *should neutralise the power of* Player 2*, that is, no matter what* Player 2 *picks, the outcome should be the same. Let* $\vec{x}_1 = (\frac{1}{2}, \frac{1}{2})$ *and* $\vec{x}_2 = (0, 1)$. *We get new reward structures* $\rho_1 = \vec{x}_1 \cdot \vec{\varrho}_1$ *and* $\rho_2 = \vec{x}_2 \cdot \vec{\varrho}_2$, *where* $\rho_1(\mathsf{a}) = \rho_1(\mathsf{b}) = 3$ *and* $\rho_2(\mathsf{a}) = \rho_2(\mathsf{b}) = 2$. *The CQ* φ *becomes* $\mathsf{Emp}(\rho_1)(1.5) \wedge \mathsf{Emp}(\rho_2)(1)$, *which is clearly achievable.*

We show, in Lemma 5.20, that ε-optimal strategies can be obtained by discretising the search space of weight vectors, and in Section 7.2.3 we show a heuristic for efficiently finding suitable weights. However, we must not reduce the search space to a degree that sacrifices completeness of Theorem 5.17, as the following example demonstrates.

Example 5.6. *We continue with Example 5.5, and illustrate the necessity of considering MQs in CNF. Consider again the game in Figure 5.9 (left). The MQ* $\psi = \mathsf{Emp}(r_1)(1.5) \vee (\mathsf{Emp}(r_2)(1.5) \wedge \mathsf{Emp}(r_3)(1.5))$ *suggests to find weight vectors* $\vec{x}_1 = (z_1, z_2)$ *and* $\vec{x}_2 = (z_1, z_3)$ *that coincide in* z_1, *the dimension corresponding to* $\mathsf{Emp}(r_1)$. *However, no such* z_1, z_2 *and* z_3 *exist, as we demonstrate in the following. We convert* ψ *to the CQ* φ *using* $\vec{x}_1$ *and* $\vec{x}_2$ *under our restriction. Then, at* s_1, *the mean-payoff is* $(z_1 + 2z_2, z_1 + z_3)$, *and at* s_2 *the mean-payoff is* $(2z_1 + z_2, 2z_1 + z_3)$. *Recall that we divided the rewards by the number of steps (that is, two) in the loops. The target vector in* φ *is* $(1.5z_1 + 1.5z_2, 1.5z_1 + z_3)$. *Let* p *be the probability of* Player 2 *going left (to* s_1*) from* s_0. *We need to pick* z_1, z_2 *and* z_3 *so that they satisfy*

$$\forall p \in [0,1]\,. \quad p(z_1 + 2z_2) + (1-p)(2z_1 + z_2) \geq 1.5z_1 + 1.5z_2$$
$$\forall p \in [0,1]\,. \quad p(z_1 + z_3) + (1-p)(2z_1 + z_3) \geq 1.5z_1 + z_3.$$

We get

$$\forall p \in [0,1]\,. \quad p(z_2 - z_1) \geq 0.5(z_2 - z_1)$$
$$\forall p \in [0,1]\,. \quad (2-p)z_1 \geq pz_1.$$

These conditions are only satisfiable if $z_1 = z_2$ *from the first line, and* $z_1 = 0$ *from the second line. But then* $\vec{x}_1 = (z_1, z_2) = \vec{0}$, *which is not allowed.*

We illustrate the situation in Figure 5.9 (centre) and (right), showing the sets $U_i = \mathsf{upc}\{\vec{y} \in \mathbb{R}^2 \mid \exists \sigma \,.\, \mathbb{E}_{\mathcal{G}}^{\pi,\sigma}[\varrho_i] = \vec{y}\}$ *for* $i = \{1, 2\}$*, respectively, which are used in the proof of Theorem 5.17 to find the weights* $\vec{x}_1$ *and* $\vec{x}_2$ *by separating the targets* $\vec{u}_1$ *and* $\vec{u}_2$ *from the respective sets* U_1 *and* U_2*. In these figures it becomes clear that the only solutions are weights* $\vec{x}_1$ *and* $\vec{x}_2$ *proportional to* $(1, 1)$ *and* $(0, 1)$*, respectively.*

We now summarise our synthesis method for ratios of expectations. Recall that the componentwise product of two vectors is denoted by $\bullet$.

Theorem 5.18. *Given a CM game* $\mathcal{G}$ *with MQs* $\psi \stackrel{\text{def}}{=} \bigwedge_{i=1}^{n} \bigvee_{j=1}^{m} \mathsf{ratioE}(r_{i,j}/c_{i,j})(v_{i,j})$ *and* $\varphi(\vec{x}) = \bigwedge_{i=1}^{n} \mathsf{Pmp}(\vec{x}_i \cdot (\vec{r}_i - \vec{v}_i \bullet \vec{c}_i))$*.*

(i) If a finite DU strategy π *achieves* $\varphi(\vec{x})$ *for some non-zero vectors* $\vec{x}_1, \ldots, \vec{x}_n \in \mathbb{R}^m_{\geq 0}$*, then* π *achieves* ψ*.*

(ii) If ψ *is* ε*-achievable by a finite DU strategy, then there exist non-zero vectors* $\vec{x}_1, \ldots, \vec{x}_n \in \mathbb{R}^m_{\geq 0}$ *such that* $\varphi(\vec{x})$ *is* ε*-achievable.*

Proof. Let $\mathcal{G}$ be a CM game. For item (i), fix a finite DU strategy π and non-zero vectors $\vec{x}_1, \ldots \vec{x}_n \in \mathbb{R}^m_{\geq 0}$ such that π achieves $\varphi(\vec{x})$. By Lemma 3.1, π also satisfies $\bigwedge_{i=1}^{n} \mathsf{Emp}(\vec{x}_i \cdot (\vec{r}_i - \vec{v}_i \bullet \vec{c}_i))$, and by Theorem 5.17, it satisfies $\bigwedge_{i=1}^{n} \bigvee_{j=1}^{m} \mathsf{Emp}(r_{i,j} - v_{i,j} \cdot c_{i,j})$. Finally, by Proposition 5.5, π satisfies $\bigwedge_{i=1}^{n} \bigvee_{j=1}^{m} \mathsf{ratioE}(r_{i,j}/c_{i,j})(v_{i,j})$.

For item (ii), assume ψ is ε-achievable by a finite DU Player 1 strategy π. Fix $\varepsilon > 0$. By Proposition 5.5, π achieves $\bigwedge_{i=1}^{n} \bigvee_{j=1}^{m} \mathsf{Emp}(r_{i,j} - v_{i,j} \cdot c_{i,j} + \frac{\varepsilon}{2})$. From Theorem 5.17, there exist non-zero vectors $\vec{x}_1, \ldots, \vec{x}_n \in \mathbb{R}^m_{\geq 0}$ such that π achieves $\bigwedge_{i=1}^{n} \mathsf{Emp}(\vec{x}_i \cdot (\vec{r}_i - \vec{v}_i \bullet \vec{c}_i + \frac{\vec{\varepsilon}}{2}))$, and, by linearity of expectations, that moreover $x_{i,j} \leq \frac{1}{m}$ for all i, j. Hence, $\vec{x}_i \cdot \frac{\vec{\varepsilon}}{2} = \sum_{j=1}^{m} x_{i,j} \cdot \frac{\varepsilon}{2} \leq \frac{\varepsilon}{2}$. Thus, π also achieves $\bigwedge_{i=1}^{n} \mathsf{Emp}(\vec{x}_i \cdot (\vec{r}_i - \vec{v}_i \bullet \vec{c}_i) + \frac{\varepsilon}{2})$. Since $\mathcal{G}$ is CM, by Theorem 5.16, there is a finite DU strategy achieving $\bigwedge_{i=1}^{n} \mathsf{Pmp}(\vec{x}_i \cdot (\vec{r}_i - \vec{v}_i \bullet \vec{c}_i) + \varepsilon)$. □

Theorem 5.18 (i) justifies that a strategy synthesised for the Pmp CQ $\varphi(\vec{x})$ is also winning for the ratioE MQ ψ. In the other direction, in item (ii), the strategy π for ψ is modified when applying Theorem 5.16 in the proof; this step is necessary as π might be randomising between MECs that are both losing in order to win for ψ.

We also remark that for synthesis with an Eratio MQ in a CM game, we can soundly use the corresponding ratioE MQ: the Eratio MQ can be converted to ta Pratio MQ using Lemma 3.1, and to a PMP MQ using Proposition 5.5, after which we can use Theorem 5.18 to obtain the Eratio MQ. A consequence of this as Pareto sets for ratioE MQs are under-approximated using Eratio MQs.

Example[re] 5.7. *The specification φ^1_{re} in the running example of Section 3.6, considered as* ratioE *objectives, is equivalent (up to ε-achievability) to a disjunction of* ratioE *objectives,* $\mathsf{ratioE}(r_1/c)(v_1) \vee \mathsf{ratioE}(r_2/c)(v_2)$. *From Theorem 5.18, we can find non-zero weights $\vec{x} \in \mathbb{R}^2_{\geq 0}$ such that we can synthesise an ε-optimal strategy for φ^1_{re} by considering instead the specification* $\mathsf{Pmp}(r')$ *with the single-objective reward structure $r' = x \cdot (\vec{r} - \vec{v} \bullet \vec{c})$.*

Recall from Example 4.13 that we are interested in the target $(\frac{1}{4}, \frac{9}{8})$ for φ^1_{re}. Consider the weight vector $(1, \frac{2}{3})$. The reward structure r' then is given by

$$\begin{aligned} r'(\mathsf{a}) = (1, \tfrac{2}{3}) \cdot (\tfrac{3}{4}, -\tfrac{9}{8}) &= 0 \\ r'(\mathsf{b}) = (1, \tfrac{2}{3}) \cdot (-\tfrac{1}{4}, -\tfrac{1}{8}) &= -\tfrac{1}{3} \\ r'(\mathsf{d}) = (1, \tfrac{2}{3}) \cdot (0, 1) &= \tfrac{2}{3}, \end{aligned}$$

and zero everywhere else. The optimal strategy for Player 1 *here clearly is to always take* d. *To spoil, the best* Player 2 *can do is play* b, *but, due to the distribution,* b *can be taken at most $\sum_{k \geq 0} 2^{-k} = 2$ times before* a *is taken again, balancing exactly the mean-payoff to zero. Hence,* Player 1 *wins for* $\mathsf{Pmp}(r')$, *and also for φ^1_{re}.*

5.5 Pareto Set Computation

We discuss how to compute Pareto sets for CM games. Recall that the Pareto set $\mathsf{Pareto}(\mathcal{G} \,|\, \varphi)$ contains the ε-achievable trade-offs for a given quantitative specification φ in a game $\mathcal{G}$, that is, if $\vec{v} \in \mathsf{Pareto}(\mathcal{G} \,|\, \varphi)$, then $\varphi[\vec{v}]$ is ε-achievable. We have already shown that the Pareto sets of satisfaction objectives may not necessarily be convex in Example 3.7. For Emp CQs in CM games, the Pareto sets are convex.

Theorem 5.19. *Emp CQs in CM games have convex Pareto sets.*

Proof. Fix a CM game $\mathcal{G}$ and an Emp CQ φ. Let P be the Pareto set for φ, and let $\vec{v}^{(1)}, \vec{v}^{(2)} \in P$. We show that the interior of P is convex: fix $\alpha \in [0, 1]$ and $\varepsilon > 0$; we have $\vec{v}^{(1)} - \varepsilon$ and $\vec{v}^{(2)} - \varepsilon$ in the interior of P and need to show that $\vec{v} \stackrel{\text{def}}{=} \alpha\vec{v}^{(1)} + (1 - \alpha)\vec{v}^{(2)} - \varepsilon$ is in the interior of P.

For $i \in \{1, 2\}$, there exists a finite DU strategy, $\pi^{(i)}$ achieving $\varphi[\vec{v}^{(i)} - \frac{\varepsilon}{2}]$. Further, for $i \in \{1, 2\}$, let $\mathfrak{E}^{(i)}$ be the set of MECs in $\mathcal{G}^{\pi^{(i)}}$, and so from Proposition 5.12 we obtain $\gamma^{(i)} \in \mathsf{D}(\mathfrak{E}^{(i)})$ satisfying $\sum_{\mathcal{E} \in \mathfrak{E}^{(i)}} \gamma^{(i)}(\mathcal{E})\vec{z}^{\mathcal{E}} \geq \vec{v}^{(i)} - \frac{\varepsilon}{2}$. We now apply Lemma 5.15 with $\pi^l = \pi^{(1)}$ for all $1 \leq l \leq |\mathfrak{E}^{(1)}|$ and $\pi^l = \pi^{(2)}$ for all $|\mathfrak{E}^{(1)}| + 1 \leq l \leq |\mathfrak{E}^{(1)}| + |\mathfrak{E}^{(2)}|$, with the respective MECs collected in $\mathfrak{E} = \mathfrak{E}^{(1)} \cup \mathfrak{E}^{(2)}$, and with $\gamma = \alpha\gamma^{(1)} + (1 - \alpha)\gamma^{(2)}$.

We thus obtain a finite DU strategy π achieving

$$\begin{aligned}
\mathbb{E}^{\pi,\sigma}_{\mathcal{G}}[\mathsf{mp}(\vec{r})] &\geq \alpha \sum_{\mathcal{E}\in\mathfrak{E}^{(1)}} \gamma^{(1)}(\mathcal{E})\vec{\mathsf{z}}^{\mathcal{E}} + (1-\alpha)\sum_{\mathcal{E}\in\mathfrak{E}^{(2)}} \gamma^{(2)}(\mathcal{E})\vec{\mathsf{z}}^{\mathcal{E}} - \frac{\varepsilon}{2} && \text{(Lemma 5.15)}\\
&\geq -\alpha(\vec{v}^{(1)} - \frac{\varepsilon}{2}) - (1-\alpha)(\vec{v}^{(2)} - \frac{\varepsilon}{2}) - \frac{\varepsilon}{2} && \text{(Proposition 5.12)}\\
&= \alpha\vec{v}^{(1)} + (1-\alpha)\vec{v}^{(2)} - \varepsilon. && \text{(arithmetic)}
\end{aligned}$$

Thus $\vec{v}$ is in the interior of P. As P is the closure of a convex set, it is itself convex. □

Note that, by linearity of expectations, Theorem 5.19 together with Proposition ii also yields that the Pareto sets for ratioE CQs are convex.

5.5.1 Emp Objectives

We first show how to approximate Pareto sets for Emp objectives.

Emp CQs. For finite Player 1 strategies, we can approximate the Pareto set of the CQ $\mathsf{Emp}(\vec{r})$ by computing the Pareto set approximations $(Y^k)_{k\geq 0}$ for the step-bounded total expected reward $\mathbb{E}^{\pi,\sigma}_{\mathcal{G}}[\mathsf{rew}^k(\vec{r})]$, using the F operator from Lemma 3.12, and evaluating $Z^k = \frac{1}{k+1}Y^k$. By Lemma 5.2, the limit $\lim_{k\to\infty}\frac{1}{k+1}\mathsf{rew}^k(\vec{r})$ exists almost surely, and so, for each $\varepsilon > 0$, there is a k such that Z^k is $\frac{\varepsilon}{2}$-close to the Pareto set of $\mathsf{Emp}(\vec{r})$. Then, $Z^k - \varepsilon$ is an under-approximation of $\mathsf{Pareto}(\mathcal{G} \mid \mathsf{Emp}(\vec{r}))$. Alternatively, since by Theorem 5.16 Emp objectives are ε-achievable when interpreted as Pmp objectives, Pareto sets for Emp can be approximated by the same method as in Theorem 6.7 of the next chapter.

Disjunctions. To compute ε-approximations for the specification $\bigvee_{j=1}^m \mathsf{Emp}(r_j)(v_j)$, we utilise Lemma 5.20 below, which is a straightforward extension of Theorem 9 of [124], where the idea is to discretise the space of weight vectors used. We first give a condition on the sets of weight vectors to be used in approximating the Pareto sets. For given $\varepsilon > 0$ and $\rho^* \geq 0$, let $\mathsf{Weights}(\rho^*)$ be a set of non-zero vectors $\vec{x} \in \mathbb{R}^m_{\geq 0}$, such that the corresponding unit vectors $\vec{y} = \vec{x}/\|\vec{x}\|$ are of the form $y_j = \lfloor y_j \rfloor_\tau$ for all i, with $\tau = \frac{\varepsilon}{2\cdot m^2\cdot(\rho^*+1)}$. The quantity ρ^* corresponds to the maximum extent in any dimension of the Pareto sets. We utilise this definition of $\mathsf{Weights}(\rho^*)$ later in Section 7.2.4 to develop a heuristic to iterate through the weight vectors.

Lemma 5.20. *Let $\mathcal{G}$ be a game, let $\vec{\varrho}: \Omega_{\mathcal{G}} \to \mathbb{R}^m$ be a bounded measurable function, and let $\rho^* = \sup_{\lambda\in\Omega_{\mathcal{G}}} \max_j |\varrho_j(\lambda)|$. The set $\bigcup_{\vec{x}\in\mathsf{Weights}(\rho^*)}\{\vec{u} \mid \vec{x}\cdot\vec{u} \leq v_{\vec{x}}\}$, where $v_{\vec{x}} = \sup_\pi \inf_\sigma \mathbb{E}^{\pi,\sigma}_{\mathcal{G}}[\vec{x}\cdot\vec{\varrho}]$, is an ε-approximation of $\mathsf{Pareto}(\mathcal{G} \mid \bigvee_{j=1}^m \mathbb{E}[\varrho_j])$.*

Using Lemma 5.20, we can compute an ε-approximation of Pareto sets for disjunctions of Emp objectives using a bounded number of one-dimensional queries.

Boolean Combinations. Consider the Emp MQ $\bigwedge_{i=1}^{n}\bigvee_{j=1}^{m}\mathsf{Emp}(r_{i,j})(v_{i,j})$ in CNF. We define the set $\mathsf{Weights}_{\mathsf{MQ}}(\vec{\rho}^*) \stackrel{\text{def}}{=} \prod_{i=1}^{n}\mathsf{Weights}(\rho_i^*)$, reflecting the requirement that for each i individually we need non-zero weight vectors $\vec{x}_i \in \mathbb{R}^m_{\geq 0}$. We let $\vec{\rho}$ be such that $\rho_i^* = \max_{s\in S,j}|r_{i,j}(s)|$ for all i, and for each choice of weight vectors $\vec{x} = (\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_n) \in \mathsf{Weights}_{\mathsf{MQ}}(\vec{\rho}^*)$, we compute an ε-approximation $P_{\vec{x}}$ of the Pareto set $\mathsf{Pareto}(\mathcal{G} \mid \bigwedge_{i=1}^{n}\mathsf{Emp}(\vec{x}_i \cdot \vec{r}_i))$. Given a point $\vec{v} \in P_{\vec{x}}$, each dimension v_i corresponds to $v_{\vec{x}_i}$ in Lemma 5.20 for the ith disjunction in the MQ. We thus obtain that

$$\bigcup_{\vec{x}\in\mathsf{Weights}_{\mathsf{MQ}}(\vec{\rho}^*)} \mathsf{conv}\Big(\bigcup_{\vec{v}\in P_{\vec{x}}}\bigcap_{i=1}^{n}\{\vec{u}\in\mathbb{R}^{n\times m} \mid \vec{x}_i\cdot\vec{u}_i \leq v_i\}\Big) \tag{5.7}$$

is the ε-approximation of the Pareto set $\mathsf{Pareto}(\mathcal{G} \mid \bigwedge_{i=1}^{n}\bigvee_{j=1}^{m}\mathsf{Emp}(r_{i,j}))$. Note that it is sufficient to consider the extreme points $\vec{v}$ of the closure of $P_{\vec{x}}$.

5.5.2 ratioE Objectives

The computation of Pareto sets for ratioE objectives is based on Emp objectives, using the transformation in Proposition 5.5 (ii), whence $\mathsf{ratioE}(r/c)(v)$ is achievable if and only if $\mathsf{Emp}(r - c\cdot v)(0)$ is achievable.

Conjunctions. Knowing the Pareto set for $\mathsf{Emp}(\vec{r} - \vec{c}\bullet\vec{v})$ does not help to determine the values for $\vec{v}$. We compute ε-approximate Pareto sets for ratioE CQs by checking achievability of $\mathsf{Emp}(\vec{r} - \vec{c}\bullet\vec{v})$ for all vectors $\vec{v}$ such that $v_i = \lfloor v_i \rfloor_\varepsilon \stackrel{\text{def}}{=} \lfloor v_i/\varepsilon \rfloor \cdot \varepsilon$ and

$$\frac{\min_{s\in S} r_i(s)}{\min_{s\in S, c_i(s)>0} c_i(s)} \leq v_i \leq \frac{\max_{s\in S} r_i(s)}{\min_{s\in S, c_i(s)>0} c_i(s)}, \tag{5.8}$$

for all i, that is, we grid the space of possible achievable targets $\vec{v}$ by ε. Since the Pareto sets of ratioE are convex by Theorem 5.19, this gridding process can be slightly accelerated, as we show in Section 7.2.4.

Boolean Combinations. We can apply Theorem 5.17 to an Emp MQ and reduce it to a CQ. However, when computing the Pareto set for ratioE MQs, we have to be able to extract the targets, which is the same problem as for ratioE CQs. For each non-zero $\vec{x}_i \in \mathbb{R}^n_{\geq 0}$, there is an index $\ell(i)$ such that $x_{i,\ell(i)} > 0$. We define reward structures $\vec{\rho}_i(s) \stackrel{\text{def}}{=} \vec{x}_i \cdot \vec{r}_i(s) + \sum_{j\neq\ell(i)} x_{i,j}\cdot c_{i,j}(s)$ and $\gamma_i(s) \stackrel{\text{def}}{=} x_{i,\ell(i)} \cdot c_{i,\ell(i)}(s)$ for every state $s \in S$ of the game. Given $\ell(i)$ and some $\kappa_i \in \mathbb{R}$, define $\vec{w}_i$ by $w_{i,j}(\kappa_i) \stackrel{\text{def}}{=} 1$ for all $j \neq \ell(i)$,

and set $w_{i,\ell(i)}(\kappa_i) \stackrel{\text{def}}{=} \kappa_i$. Then, $\bigwedge_{i=1}^{n} \mathsf{Emp}(\vec{x}_i \cdot (\vec{r}_i - \vec{c}_i \bullet \vec{w}_i(\kappa_i)))$ is achievable if and only if $\mathsf{Emp}(\vec{\rho} - \vec{\gamma} \bullet \vec{\kappa})$ is achievable. Note that, by varying κ_i, the value of $\vec{x}_i \cdot \vec{w}_i(\kappa_i)$ is determined. We obtain the values for $\vec{\kappa}$ by computing the ratioE CQ Pareto set $Q_{\vec{x}} = \mathsf{Pareto}(\mathcal{G} \mid \mathsf{ratioE}(\vec{\rho}/\vec{\gamma}))$. We thus get that $\mathsf{Pareto}(\mathcal{G} \mid \bigwedge_i \bigvee_j \mathsf{ratioE}(r_{i,j}/c_{i,j}))$ is ε-approximated by

$$\bigcup_{\vec{x} \in \mathsf{Weights}_{\mathsf{MQ}}(\vec{\rho}^*)} \mathsf{conv}\Big(\bigcup_{\vec{\kappa} \in Q_{\vec{x}}} \bigcap_{i=1}^{n} \{\vec{u} \in \mathbb{R}^{n \times m} \mid \vec{x}_i \cdot \vec{u}_i \leq \vec{x}_i \cdot \vec{w}_i(\kappa_i)\}\Big), \tag{5.9}$$

where $\rho_i^* = \max_{s \in S, j} r_{i,j}(s) \cdot (\min_{s \in S, j, c_{i,j}(s) > 0} c_{i,j}(s))^{-1}$, for all i, according to (5.8).

Example[re] 5.8. *We resume the discussion from Example 4.12 to compute Pareto sets for the running example in Section 3.6, and show here how to compute the local Pareto sets for the individual components. Recall that the reward structures $\vec{r}$ and c are given by $\vec{r}(a) = (1, 0, 0)$, $\vec{r}(b) = (0, 1, 1)$, $\vec{r}(d) = (0, 1, 0)$, $c(a) = c(b) = 1$, and zero everywhere else, and note that the games $\mathcal{G}^1_{\mathsf{re}}$, $\mathcal{G}^2_{\mathsf{re}}$ and $\mathcal{G}_{\mathsf{re}}$ are CM, so we can use the methods of this section to approximate Pareto sets for expected rewards. Recall that we can underapproximate Pareto sets in CM games for Eratio objectives using ratioE objectives. The Pareto sets are shown for the specifications $\varphi^1_{\mathsf{re}} = \mathsf{ratioE}^{\leq}(r_1/c)(v_1) \rightarrow \mathsf{ratioE}(r_2/c)(v_2)$ for $\mathcal{G}^1_{\mathsf{re}}$ and $\varphi^2_{\mathsf{re}} = \mathsf{ratioE}^{\leq}(r_1/c)(v_1) \wedge \mathsf{ratioE}^{\leq}(r_3/c)(v_3)$ for $\mathcal{G}^2_{\mathsf{re}}$. Note, however, that we defined Pareto sets for maximising objectives only, so the specifications used for computation are $\mathsf{ratioE}(r_1/c)(v_1) \vee \mathsf{ratioE}(r_2/c)(v_2)$ for $\mathcal{G}^1_{\mathsf{re}}$ (the strict inequality is ignored, see the discussion in Section 3.4.2), and $\mathsf{ratioE}(-r_1/c)(-v_1) \wedge \mathsf{ratioE}(-r_3/c)(-v_3)$ for $\mathcal{G}^2_{\mathsf{re}}$. The resulting Pareto sets are shown in Figure 4.9, where we flip the negated dimensions to reflect the sign of the original specifications.*

5.6 Summary

In this chapter we discussed multi-objective queries involving Boolean combinations of objectives, which was motivated by the assume-guarantee strategy synthesis rules of Chapter 4, such as (Asym) and (Circ). Specifications consisting of Eratio objectives are particularly suited for our assume-guarantee strategy synthesis framework, because they are defined on traces and we can (soundly) synthesise strategies for arbitrary Boolean combinations, using the corresponding ratioE objectives. Likewise, Pratio objectives are defined on traces, for which we can synthesise strategies for conjunctions.

The core of this chapter presented a series of reductions between specifications. We began by providing a taxonomy of strategies required for the Pmp and Emp objectives under discussion, and moreover justified our use of SU strategies by showing that they can be exponentially more compact than DU strategies. Using the more compact SU strategies is particularly relevant for our tool implementation presented in Chapter 7. We then showed that ratioE and Pratio objectives can be reduced to Emp and Pmp objectives, respectively, by instantiating the reward structures appropriately, and so the discussion of strategies carries over to these objectives. To be able to synthesise strategies winning for expectation semantics, we further defined the class of controllable multichain (CM) games, in which Emp and Pmp objectives are equivalent under ε-achievability. The class of CM games contains the games we use in our case studies in Sections 8.3 and 8.4, demonstrating that CM games can model versatile control applications. We then showed that Boolean combinations of expectation objectives reduce to conjunctions, if appropriate weight vectors are found. For ε-achievability, these weight vectors can be found from a discretised set, as we further develop algorithmically in Sections 7.2.3–7.2.4. We used our reductions to compute Pareto sets for Boolean combinations of expectation objectives, which provides a decision maker with valuable feedback about the achievable queries.

We highlight some open questions emerging from the foregoing discussion. Firstly, it remains an open question how to synthesise strategies for general Boolean combinations of Pmp objectives in non-CM games. For a special class of Pmp MQs of the form $\bigwedge_j \mathsf{Pmp}(r_j^A)(v_j^A) \to \bigwedge_i \mathsf{Pmp}(r_j^G)(v_j^G)$, it is possible to slightly relax the CM assumption to only require reachability of all ICs from the initial state of the game, and thus obtain a reduction to Pmp CQs. Secondly, we do not address completeness for synthesis with expected ratio rewards. We already noted in Example 3.6 that the values for expected ratios and ratios of expectations do not coincide in general. For MDPs, synthesis for conjunctions of (one-dimensional) expected ratio objectives has been discussed in [135], using that a strategy in an MDP can control which MECs are entered, and thus a synthesis algorithm can reduce the problem to synthesis in MECs. In games, the same flexibility does not exist for Player 1 in general, since it is up to Player 2 to control the induced PA. Finally, it remains to be shown whether finite SU strategies and finite DU strategies are equally powerful, both for Pmp and Emp objectives. For conjunctions of expected total reward (Erew) objectives, [124] shows that for the particular case of conjunctions of target reachability objectives infinite DU strategies are necessary, but finite SU strategies are sufficient; however, the same argument does not straightforwardly carry over to Pmp objectives.

Chapter 6

Strategy Synthesis

Contents

In this chapter we develop the main strategy synthesis algorithm for conjunctions of objectives (CQs) requiring to keep a mean-payoff almost surely above a threshold (Pmp). Synthesis for Pmp CQs is sufficient to synthesise ε-optimal strategies for Boolean combinations of expectation objectives (Emp and ratioE), as well as conjunctions of almost sure ratio (Pratio) objectives, as we have shown in Chapter 5.

We develop a co-NP decision procedure for the achievability problem of Pmp CQs, which decides whether a (potentially infinite) Player 1 strategy achieves the specification. This decision procedure allows us to precede our synthesis algorithm by a check of achievability, yielding a complete algorithm. We then show that we can construct strategies that achieve Pmp CQs by considering *expected energy* (EE) objectives instead, which require maintaining the expected total reward above a bounded *shortfall* at every step, due to the observation that a strategy winning for expected energy almost surely keeps the mean-payoff above zero. The argument for the reduction to EE objectives relies on finiteness of the induced DTMCs, and so we show that it suffices to consider finite memory strategies for Player 2, if Player 1 also uses finite memory.

Our strategy construction is based on a geometric interpretation of the strategy memory, where each memory element is mapped to the vector of shortfalls achievable when starting the strategy with this memory. We therefore develop a Bellman

operator to characterise the sets of achievable shortfalls, and formalise a fixpoint computation in order to approximate the shortfall sets. We show that if a Pmp CQ is achievable, our fixpoint computation terminates after a finite number of steps, and so the resulting shortfall sets are polytopes with a finite number of extreme points. The strategies we construct keep the extreme points in memory, and so we obtain finite (SU) strategies. While we have shown that finite SU strategies can be exponentially more succinct than DU strategies, which is particularly relevant for our tool implementation in Chapter 7, they cannot, in general, be unrolled to finite DU strategies. Since our transformation between Pmp and EE objectives was shown for finite DU strategies, in order to obtain a complete algorithm we give a discretisation of SU strategies, showing that we can also obtain finite ε-optimal DU strategies for Pmp CQs. We remark that discretising SU strategies for Emp CQs is an open question.

The technical contributions of this chapter are mainly based on our previous work in [10]. We start the chapter by analysing the complexity of the Pmp CQ achievability problem in Section 6.1, which is based on the observation that it is sufficient for Player 1 to win against MD Player 2 strategies. In Section 6.2 we then develop our reduction of Pmp objectives to expected (truncated) energy, and we give a Bellman operator to characterise the truncated energy shortfalls. We develop our strategy construction in Section 6.3, both for SU strategies and the discretisation to finite DU strategies. Finally, in Section 6.4 we summarise our synthesis algorithm for Pmp CQs.

6.1 Decision Procedure

In this section we present our decidability result of the achievability problem for Pmp CQs, based on a general class of objectives defined via *shift-invariant submixing* functions. A function $\varrho : \Omega_{\mathcal{G}} \to \mathbb{R}$ is *shift-invariant* if $\forall \kappa \in \Omega_{\mathcal{G}}^{\mathsf{fin}} \,.\, \lambda \in \Omega_{\mathcal{G}} \,.\, \varrho(\kappa\lambda) = \varrho(\lambda)$. A function $\varrho : \Omega_{\mathcal{G}} \to \mathbb{R}$ is *submixing* if, for all $\kappa, \kappa', \lambda \in \Omega_{\mathcal{G}}$ such that λ is an interleaving of κ and κ', it holds that $\varrho(\lambda) \leq \max\{\varrho(\kappa), \varrho(\kappa')\}$.

We obtain a co-NP algorithm by studying the strategies Player 2 needs to win for Pmp objectives against Player 1, and using qualitative determinacy for Pmp objectives (see Lemma 3.9). From [72] we have that MD strategies suffice for Player 1 to win for one-dimensional shift-invariant submixing functions.

Lemma 6.1 (Theorem V.2 of [72])**.** *Let $\mathcal{G}$ be a game, and let $\varrho : \Omega_{\mathcal{G}} \to \mathbb{R}$ be a measurable, shift-invariant and submixing function. Player 1 has an MD strategy $\tilde{\pi}$ such that $\inf_\sigma \mathbb{E}_{\mathcal{G}}^{\tilde{\pi},\sigma}[\varrho] = \sup_\pi \inf_\sigma \mathbb{E}_{\mathcal{G}}^{\pi,\sigma}[\varrho]$.*

Given a measurable function ϱ, we write $\mathsf{P}(\varrho)$ for the objective $\mathbb{P}_{\mathcal{D}}(\varrho \geq 0) = 1$.

Theorem 6.2. *Let $\mathcal{G}$ be a game, and let $\vec{\varrho} : \Omega_{\mathcal{G}} \to \mathbb{R}^n$ be a vector of shift-invariant submixing functions. If Player 1 wins for $\mathsf{P}(-\vec{\varrho})$ against all MD Player 2 strategies, then Player 1 wins for $\mathsf{P}(-\vec{\varrho})$ against all Player 2 strategies.*

Proof. We use Lemma 6.1 by evaluating the expectation of the indicator function of the objective. Given a measurable subset Λ of $\Omega_{\mathcal{G}}$, we denote by 1_Λ its *indicator function*, defined as $1_\Lambda(\lambda) \stackrel{\text{def}}{=} 1$ if $\lambda \in \Lambda$ and 0 otherwise. Fix shift-invariant submixing functions $\varrho_1, \ldots, \varrho_n$, and let $A \stackrel{\text{def}}{=} \{\lambda \mid \exists i \,.\, -\varrho_i(\lambda) < 0\}$ be the set of paths falsifying the property of interest. We first show the following intermediate result.

Lemma 6.3. *The function 1_A is shift-invariant and submixing.*

Proof. Since ϱ_i is shift-invariant for all i, also 1_A is shift-invariant. We now show that 1_A is submixing. Let $\lambda, \kappa, \kappa' \in \Omega_{\mathcal{G}}$ such that λ is an interleaving of κ and κ'. Assume $1_A(\kappa) = 1_A(\kappa') = 0$, that is $-\varrho_i(\kappa) \geq 0$ and $-\varrho_i(\kappa') \geq 0$ for all i. Since ϱ_i is submixing, $\varrho_i(\lambda) \leq \max\{\varrho_i(\kappa), \varrho_i(\kappa')\}$, for all i. Then, for all i, $0 \leq \min\{-\varrho_i(\kappa), -\varrho_i(\kappa')\} \leq -\varrho_i(\lambda)$. Thus, $1_A(\lambda) = 0$. □

We show the theorem by contraposition. Assume $\forall \pi \,.\, \exists \sigma \,.\, \mathbb{P}_{\mathcal{G}}^{\pi,\sigma}(\exists i \,.\, -\varrho_i < 0) > 0$. Hence $\forall \pi \,.\, \exists \sigma \,.\, \mathbb{E}_{\mathcal{G}}^{\pi,\sigma}[1_A] > 0$, and so, from Lemma 3.9, we have that $\exists \sigma \,.\, \forall \pi \,.\, \mathbb{E}_{\mathcal{G}}^{\pi,\sigma}[1_A] > 0$. Since 1_A is shift-invariant and submixing, by Lemma 6.1 (via switching players), there is an MD Player 2 strategy $\tilde{\sigma}$ in $\mathcal{G}$ such that $\forall \pi \,.\, \mathbb{E}_{\mathcal{G}}^{\pi,\tilde{\sigma}}[1_A] > 0$. Then, for all Player 1 strategies π, we have $\mathbb{P}_{\mathcal{G}}^{\pi,\tilde{\sigma}}(\exists i \,.\, -\varrho_i < 0) > 0$. □

Our proof for Pmp CQs derives from a transfer theorem, that characterises the complexity of achievability problems for games if the complexity is known for PAs.

Theorem 6.4. *Let $\mathcal{G}$ be a game, and let Φ be a class of specifications, such that for every $\varphi \in \Phi$, $\mathcal{G}$ with φ is qualitatively determined and MD strategies suffice for Player 2. Suppose that, for a PA $\mathcal{M}$ with $\varphi \in \Phi$, the problem $\exists \sigma \,.\, \mathcal{M}^\sigma \models \varphi$ is in the time-complexity class A. The problem $\exists \pi \,.\, \forall \sigma \,.\, \mathcal{G}^{\pi,\sigma} \models \varphi$ is in co-NP if A $\subseteq$ co-NP, and in A if A $\supseteq$ co-NP.*

Proof. By qualitative determinacy, the decision problem of interest is equivalent to $\forall \sigma \,.\, \exists \pi \,.\, \mathcal{G}^{\pi,\sigma} \models \varphi$. The answer is negative exactly if $\exists \sigma \,.\, \forall \pi \,.\, \mathcal{G}^{\pi,\sigma} \models \neg\varphi$, which is equivalent to deciding whether some MD strategy σ satisfies $\forall \pi \,.\, \mathcal{G}^{\pi,\sigma} \models \neg\varphi$. Such an MD spoiling strategy σ can be guessed in polynomial time. To decide $\forall \pi \,.\, \mathcal{G}^{\pi,\sigma} \models \neg\varphi$,

it suffices to decide its negation $\exists \pi \,.\, \mathcal{G}^{\pi,\sigma} \models \varphi$, and this problem is in the class A. The overall complexity is hence the maximum complexity of co-NP and A. □

Using Lemma 3.9 and Theorem 6.2, we obtain the following corollary of Theorem 6.4.

Corollary 6.5. *Let $\mathcal{G}$ be game, let $\vec{\varrho} : \Omega_{\mathcal{G}} \to \mathbb{R}^n$ be a measurable shift-invariant submixing function, and suppose the problem whether there exists a strategy σ for a PA $\mathcal{M}$ such that $\mathcal{M}^\sigma$ satisfies $P(-\vec{\varrho})$ is in the time-complexity class A. The problem $\exists \pi \,.\, \forall \sigma \,.\, \mathcal{G}^{\pi,\sigma} \models P(-\vec{\varrho})$ is in co-NP if $\mathsf{A} \subseteq$ co-NP, and in A if $\mathsf{A} \supseteq$ co-NP.*

As a further corollary, the achievability problem for Pmp CQs in co-NP, since we can decide the corresponding achievability problem in PAs in polynomial time.

Corollary 6.6. *The Pmp CQ achievability problem is in co-NP.*

Proof. From Proposition VI.1 in [72], $-\mathsf{mp}(\vec{r})$ is shift-invariant and submixing, and from Lemma 3.11, the problem $\exists \sigma \,.\, \mathcal{M}^\sigma \models \mathsf{Pmp}(\vec{r})$ is polynomial-time for PAs $\mathcal{M}$. □

The decision procedure we propose does not yield a straightforward algorithm to synthesise Player 1 strategies, which is the main subject of the rest of this chapter. However, the decision procedure gives us an alternative way to ε-approximate Pareto sets for Pmp objectives, compared to Section 5.5.

Pareto Sets for Pmp CQs. We grid the set of targets in the n dimensional hyperrectangle $\{\vec{v} \in \mathbb{R}^n \mid \forall i \,.\, -\rho^* \leq v_i \leq \rho^*\}$ with $\rho^* \stackrel{\text{def}}{=} \max_{i,s \in S} |r_i(s)|$, selecting points at most a distance ε apart. At every such point $\vec{v}$ in the grid, we call the co-NP decision procedure of Corollary 6.6, and hence obtain an ε-approximation of the Pareto set by taking the downward closure of the set of achievable points. There are ρ^*/ε sections per dimension, and $2^{|S|}$ strategies to be checked with the polynomial-time oracle of B.3. in [18] (Lemma 3.11), and so we obtain the following result.

Theorem 6.7. *The ε-approximation of the Pareto set for an n-dimensional conjunction of Pmp objectives can be computed in time $\mathcal{O}((\rho^*/\varepsilon)^n \cdot 2^{|S|})$.*

6.2 Expected Energy Objectives

In order to construct a strategy, we need information on how the rewards change from one state to the next, so that the correct choices can be made. However, if we compute the mean-payoff at every state in the game, we find that the values achievable at each state might be the same, and so do not yield sufficient information on how the

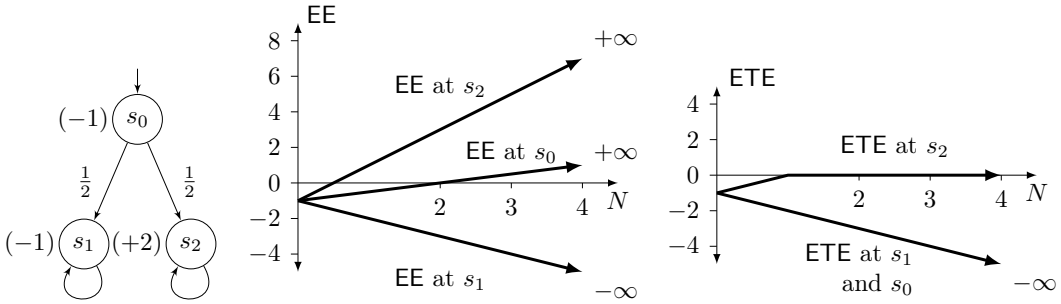


Figure 6.1: Illustration of expected energy (EE) and expected truncated energy (ETE).

strategy should pick successor states. For example, in the game in Figure 5.3 (left) of Chapter 5, the Pareto sets for Pmp objectives, shown in the centre of the figure, are the same at every state of the game, and so we cannot construct the memory updates for a strategy from them, which is required to win for some targets.

Instead, we convert Pmp objectives to an alternative objective, called *expected energy* (EE), for which we develop our strategy construction. We explain the intuition behind this conversion. Observe that, for finite DTMCs, if the mean-payoff $\mathsf{mp}(r)$ is non-negative, at almost every step N, the corresponding total reward $\mathsf{rew}(r)$ up to N steps is bounded from below. In particular, if the total reward diverges to negative infinity, the mean-payoff must be negative. We call the total reward up to a finite number of steps *energy*, since it represents the amount of resources that need to be spent up to that point. If the energy converges to a finite, negative, *shortfall* v_0, we know that the system, when started with a supply of $-v_0$, does not run out of resources. Consider the DTMC in Figure 6.1 (left). The mean-payoff that can be achieved almost surely in s_0 is -1, since there is some non-zero probability that the BSCC consisting of $\{s_1\}$ is accessed, where the mean-payoff is -1. Correspondingly, in Figure 6.1 (centre), we show the expected energy at each state. At s_1 the total reward diverges towards negative infinity, at s_2 the total reward diverges towards positive infinity, and at s_0 the expectation is positive after the second step. We observe that in order for the mean-payoff to be non-negative almost surely at the initial state, it must be non-negative almost surely in every BSCC of the DTMC, and so the expected total reward up to any step is lower-bounded at every state in the DTMC. We now formally define expected energy objectives.

Definition 6.1. *A DTMC $\mathcal{D}$ satisfies the* expected energy objective $\mathsf{EE}(r)$ *if there is a finite* shortfall v_0 *such that, for all states s of $\mathcal{D}$, and all $N \geq 0$, $\mathbb{E}_{\mathcal{D},s}[\mathsf{rew}^N(r)] \geq v_0$.*

Note that a game satisfies an EE objective if there is a Player 1 strategy, and a finite shortfall for each Player 2 strategy. Given a strategy satisfying an EE objective, we show below in Proposition 6.18 that it also satisfies the corresponding Pmp objective. The reduction to energy objectives is inspired by a similar result for non-stochastic games [26, 36], which we summarised in Section 3.5.3 (with inverted signs).

However, as shown in Figure 6.1, even if the expected energy is lower-bounded for one state, it may diverge towards positive infinity at other states. We therefore truncate the positive values of the expected energy at every step N, which is illustrated in Figure 6.1 (right). Then, we can find finite shortfalls for the expected truncated energy, and hence we can construct in Section 6.3 a strategy that satisfies the EE objective. Initially, we use the expected truncated energy for a single dimension to show that, for EE objectives, it suffices to consider finite memory strategies for Player 2.

6.2.1 Finite Memory Strategies for EE

When Player 1 fixes a finite DU strategy π with the aim to satisfy a conjunction of EE objectives, then the aim of Player 2 is to falsify (that is, spoil) at least one objective in the finite induced PA $\mathcal{M} = \mathcal{G}^\pi = \langle S, (S_\Box, S_\bigcirc), \varsigma, \mathcal{A}, \chi, \Delta \rangle$. We therefore fix in this section a single reward r, for which Player 2 aims to falsify $\mathsf{EE}(r)$. A Player 2 strategy σ spoils if for every shortfall v_0 there exists a state $(s, \mathfrak{m})$ of $\mathcal{M}^\sigma$, such that, for some k, $e^k_{s,\mathfrak{m}} \leq v_0$, where we define $e^k_{s,\mathfrak{m}} \stackrel{\text{def}}{=} \mathbb{E}_{\mathcal{M}^\sigma,(s,\mathfrak{m})}[\mathsf{rew}^k(r)]$. In the following, we use boldface notation for vectors over the state space, reserving the arrow notation for vectors over the reward dimensions, for example, a vector $\mathbf{a}$ is defined by its components $[\mathbf{a}]_s$ for every state $s \in S$ (or some subset of S). To witness whether Player 2 can spoil, without needing to induce the DTMC $\mathcal{M}^\sigma$, we also define the sequence of *expected truncated energy* $(\mathbf{u}^k)_{k\geq 0}$ parameterised by states of the PA $\mathcal{M}$: for all states s of $\mathcal{M}$, we let $u^0_s \stackrel{\text{def}}{=} 0$, and for every $k > 0$, we let

$$u^{k+1}_s \stackrel{\text{def}}{=} \begin{cases} \min(0, r(s) + \min_{t\in\Delta(s)} u^k_t) & \text{if } s \in S_\Box \\ \min(0, r(s) + \sum_{t\in\Delta(s)} \Delta(s,t) \cdot u^k_t) & \text{if } s \in S_\bigcirc. \end{cases} \tag{6.1}$$

One can see by induction on k that $(\mathbf{u}^k)_{\geq 0}$ is a non-increasing sequence, and we denote by $\mathbf{u}^*$ its limit in $(\mathbb{R}_{\leq 0} \cup \{-\infty\})^{|S|}$ as $k \to \infty$. Let $S_{\mathsf{fin}} \stackrel{\text{def}}{=} \{s \in S \mid u^*_s > -\infty\}$ be the set of states s of $\mathcal{M}$ such that u^*_s is finite, and let $S_\infty = S \setminus S_{\mathsf{fin}}$. We now show that $S_\infty \neq \emptyset$ witnesses that Player 2 can spoil the EE objective with a finite strategy.

Proposition 6.8. *Let $\mathcal{M}$ be a finite PA with a one-dimensional reward structure r. If $S_\infty \neq \emptyset$, then Player 2 has a finite strategy to spoil $\mathsf{EE}(r-\varepsilon)$, for every $\varepsilon > 0$.*

Proof. The proof is as follows. For k large enough and for states $s \in S_\infty$, there is no cut-off used to define u_s^k, similarly to the non-truncated energy. We can then express the equations for $\mathbf{u}^k$ in terms of matrices, from which we construct a finite memory Player 2 strategy that keeps the expected energy with respect to the reward r bounded from below. By operating with the reward $r - \varepsilon$, we subtract ε at each step, and so the expected energy goes to $-\infty$, falsifying $\mathsf{EE}(r-\varepsilon)$.

Let k_0 be the least integer such that, for all $k \geq k_0$, $u_s^k < 0$ for every $s \in S_\infty$. For $k \geq 0$ and $s \in S_\Box$, let $\sigma_k(s)$ be a successor of s, for which the minimum is attained, that is, $u_s^{k+1} = \min\{0, r(s) + u_{\sigma_k(s)}^k\}$. Let $U^{(k)}$ be the $S \times S$ matrix for $\mathcal{M}$ defined by

$$U_{s,t}^{(k)} = \begin{cases} 1 & \text{if } s \in S_\Box \text{ and } t = \sigma_k(s) \\ \Delta(s,t) & \text{if } s \in S_\bigcirc \\ 0 & \text{otherwise,} \end{cases}$$

for all $s, t \in S$. Let $U^{(j,i)}$ be the matrix product $U^{(j-1)} \cdot U^{(j-2)} \cdot \dots \cdot U^{(i)}$ for $j > i$, and let $U^{(i,i)} = I_S$ (the identity). We use the following block decomposition of $U^{(k)}$:

$$U^{(k)} = \begin{pmatrix} U_{S_\infty}^{(k)} & U_{S_\infty, S_{\mathsf{fin}}}^{(k)} \\ 0 & U_{S_{\mathsf{fin}}}^{(k)} \end{pmatrix}. \tag{6.2}$$

The zero block in the lower left corner of $U^{(k)}$ arises because all successors of states in S_{fin} are in S_{fin}. Moreover, $U_{S_\infty}^{(j,i)} = U_{S_\infty}^{(j-1)} \cdot U_{S_\infty}^{(j-2)} \cdot \dots \cdot U_{S_\infty}^{(i)}$.

Remark 6.9. *For every $k \geq k_0$, it holds that $\mathbf{u}_{S_\infty}^{k+1} = \mathbf{r}_{S_\infty} + [U^{(k)} \cdot \mathbf{u}^k]_{S_\infty}$.*

We now proceed to show Proposition 6.8 as a consequence of Lemmas 6.10–6.15. Our first step is to show in Lemma 6.12 that there is a subset E of states in S_∞ where Player 2 has no incentive to leave to spoil EE. Two technical results are needed for this step: a) in Lemma 6.10 we show that, within S_∞, the expected truncated energy behaves as a sequence whose update is bounded by an affine recursion; and b) in Lemma 6.11 we show that, for such sequences to diverge with a bounded rate $c_m \leq 1$, the rate must be saturated to 1 in the limit.

Lemma 6.10. *For every $l \geq 0$, there exists a constant $b_l \geq 0$, such that, for every $k \geq k_0$, it holds that $\|\mathbf{u}^{k+l}_{S_\infty}\|_\infty \leq \|U^{(k+l,k)}_{S_\infty}\|_\infty \cdot \|\mathbf{u}^{k}_{S_\infty}\|_\infty + b_l$.*

Proof. We first show, by induction on l, the more general statement

$$\mathbf{u}^{k+l}_{S_\infty} \geq U^{(k+l,k)}_{S_\infty} \cdot \mathbf{u}^{k}_{S_\infty} + \mathbf{a} - l\boldsymbol{\rho}^*, \tag{6.3}$$

where $\mathbf{a}$ and $\boldsymbol{\rho}^*$ are the constant vector with equal components $a \stackrel{\text{def}}{=} \min_{s \in S_{\mathsf{fin}}} u^*_s$ and $\rho^* \stackrel{\text{def}}{=} \max_{s \in S} |r(s)|$, respectively. The base case, for $l = 0$, is satisfied. Now assume (6.3) holds for some index l, and we show that then (6.3) is true for $l+1$. Recall that for $k \geq k_0$ and $s \in S_\infty$, there is no cut-off of positive values in u^k_s. We thus obtain

$$\begin{aligned}
\mathbf{u}^{k+l+1}_{S_\infty} &= \mathbf{r}_{S_\infty} + [U^{(k+l)} \cdot \mathbf{u}^{k+l}]_{S_\infty} && \text{(Remark 6.9)}\\
&= \mathbf{r}_{S_\infty} + U^{(k+l)}_{S_\infty} \cdot \mathbf{u}^{k+l}_{S_\infty} + U^{(k+l)}_{S_\infty, S_{\mathsf{fin}}} \cdot \mathbf{u}^{k+l}_{S_{\mathsf{fin}}} && \text{(by (6.2))}\\
&\geq -\boldsymbol{\rho}^* + U^{(k+l)}_{S_\infty} \cdot \mathbf{u}^{k+l}_{S_\infty} + U^{(k+l)}_{S_\infty, S_{\mathsf{fin}}} \cdot \mathbf{a} && \text{(definition of } \mathbf{a} \text{ and } \boldsymbol{\rho}^*)\\
&\geq -\boldsymbol{\rho}^* + U^{(k+l)}_{S_\infty} \cdot (U^{(k+l,k)}_{S_\infty} \cdot \mathbf{u}^{k}_{S_\infty} + \mathbf{a} - l\boldsymbol{\rho}^*) + U^{(k+l)}_{S_\infty, S_{\mathsf{fin}}} \cdot \mathbf{a} \\
& && \text{(induction hypothesis)}\\
&\geq U^{(k+l+1,k)}_{S_\infty} \cdot \mathbf{u}^{k}_{S_\infty} + (U^{(k+l)}_{S_\infty} + U^{(k+l)}_{S_\infty, S_{\mathsf{fin}}}) \cdot \mathbf{a} - (l+1)\boldsymbol{\rho}^* && \text{(rearranging)}\\
&\geq U^{(k+l+1,k)}_{S_\infty} \cdot \mathbf{u}^{k}_{S_\infty} + \mathbf{a} - (l+1)\boldsymbol{\rho}^*. && (U^{(k+l)} \text{ is stochastic})
\end{aligned}$$

It now suffices to define $b_l \stackrel{\text{def}}{=} a - l\rho^*$, and take the norm in (6.3), and so we get

$$\begin{aligned}
\|\mathbf{u}^{k+l}_{S_\infty}\|_\infty &= \max_{s \in S_\infty} |\mathbf{u}^{k+l}_s| = \max_{s \in S_\infty} (-\mathbf{u}^{k+l}_s)\\
&\leq \max_{s \in S_\infty} (U^{(k+l,k)}_{S_\infty} \cdot (-\mathbf{u}^{k}_{S_\infty}) + b_l) && \text{(by (6.3))}\\
&= \|U^{(k+l,k)}_{S_\infty} \cdot (-\mathbf{u}^{k}_{S_\infty})\|_\infty + b_l\\
&\leq \|U^{(k+l,k)}_{S_\infty}\|_\infty \cdot \|\mathbf{u}^{k}_{S_\infty}\|_\infty + b_l. && \text{(sub-multiplicativity)}
\end{aligned}$$

□

Lemma 6.11. *Let $b \geq 0$, and let $(x_m)_{m \geq 0}$ and $(c_m)_{m \geq 0}$ be non-negative real sequences. If $x_m \to \infty$ as $m \to \infty$, and, for every $m \geq 0$, $x_{m+1} \leq c_m x_m + b$ and $c_m \leq 1$, then it holds that $\sup_m c_m = 1$.*

Proof. Assume toward a contradiction that there exists $\theta < 1$ such that, for every m, $c_m \leq \theta$. As $x_m \to \infty$, there exists m_0 such that, for every $m \geq m_0$, it holds that $x_m > b/(1-\theta)$, and hence that $x_{m+1}/x_m \leq c_m + b/x_m < \theta + b/(b/(1-\theta)) = 1$. This

yields that from the index m_0, the sequence $(x_m)_{m \geq m_0}$ is decreasing, and thus cannot go to $+\infty$, a contradiction. □

In the proof of Lemma 6.12 we show that, if the set E contains a state in S_{fin}, then the probability of staying in E is strictly less than 1, allowing us to derive a contradiction to the divergence of the expected truncated energy in S_∞.

Lemma 6.12. *If $S_\infty \neq \emptyset$, then there exists a set $E \subseteq S_\infty$ and indices $j > i \geq k_0$, such that $U_E^{(i,j)}$ is stochastic.*

Proof. Given a subset of states $A \subseteq S$, and a $S \times S$ stochastic matrix P, we define $\mathtt{Reach}(A, P) \stackrel{\text{def}}{=} \{s' \mid \exists s \in A \,.\, P_{s,s'} > 0\}$. Note that P_E, the matrix P restricted to the states E, is stochastic if and only if $\mathtt{Reach}(E, P) \subseteq E$, and further that $\mathtt{Reach}(\mathtt{Reach}(A, P), P') = \mathtt{Reach}(A, P \cdot P')$. Let $l = 2^{|S|}$, $s \in S$ and $k \in \mathbb{N}$. Consider the sets $\mathtt{Reach}(\{s\}, U^{(k+l,k+l)})$, $\mathtt{Reach}(\{s\}, U^{(k+l,k+l-1)})$, …, $\mathtt{Reach}(\{s\}, U^{(k+l,k)})$. By the pigeonhole principle, there are at least two indices i, j with $k \leq i < j \leq k + l$ such that $\mathtt{Reach}(\{s\}, U^{(k+l,j)}) = \mathtt{Reach}(\{s\}, U^{(k+l,i)})$, and we denote this common set by $E_{s,k}$. We thus have that

$$\begin{aligned}
\mathtt{Reach}(E_{s,k}, U^{(j,i)}) &= \mathtt{Reach}(\mathtt{Reach}(\{s\}, U^{(k+l,j)}), U^{(j,i)}) \\
&= \mathtt{Reach}(\{s\}, U^{(k+l,j)} \cdot U^{(j,i)}) \\
&= \mathtt{Reach}(\{s\}, U^{(k+l,i)}) \\
&= E_{s,k}.
\end{aligned}$$

Hence $U_{E_{s,k}}^{(j,i)}$ is stochastic. It now suffices to prove that $E_{s,k} \subseteq S_\infty$ for some $s \in S_\infty$ and $k \geq k_0$. Assume for the sake of contradiction that $E_{s,k} \not\subseteq S_\infty$ for every $s \in S_\infty$ and $k \geq k_0$. This means that, for every $s \in S_\infty$ and $k \geq k_0$, there exists i such that $\mathtt{Reach}(U^{(i,k+l)}, \{s\}) \cap S_{\mathsf{fin}} \neq \emptyset$. For $j \geq i$, we write $U^{(j,i)}$ as the block decomposition $(U_{S,S_\infty}^{(j,i)}, U_{S,S_{\mathsf{fin}}}^{(j,i)})$, and write $U_{s,S_{\mathsf{fin}}}^{(j,i)}$ for the row of $U_{S,S_{\mathsf{fin}}}^{(j,i)}$ corresponding to state s. Now recall that $U_{S_{\mathsf{fin}}}^{(i,k)}$ is stochastic, since every successors of a state in S_{fin} is in S_{fin}. We deduce that $\|U_{s,S_{\mathsf{fin}}}^{(k+l,i)} \cdot U_{S_{\mathsf{fin}}}^{(i,k)}\|_\infty > 0$, hence that $\|U_{s,S_{\mathsf{fin}}}^{(k+l,k)}\|_\infty > 0$. The matrix $U^{(k+l,k)}$ is the product of l matrices, each of which has entries either zero or greater than $p_{\min}$, the minimal probability on edges of the PA $\mathcal{M}$. Therefore, coefficients of $U^{(k+l,k)}$ are either zero or greater than $p_{\min}^l$, and so $\|U_{s,S_{\mathsf{fin}}}^{(k+l,k)}\|_\infty \geq p_{\min}^l$. Since $U^{(k+l,k)}$ is stochastic, its row-sum are equal to one, that is, $\sum_{s' \in S_{\mathsf{fin}}} U_{s,s'}^{(k+l,k)} + \sum_{s' \in S_\infty} U_{s,s'}^{(k+l,k)} = 1$, for every $s \in S$ and $k \geq 0$. This implies that $\sum_{s' \in S_\infty} U_{s,s'}^{(k+l,k)} \leq 1 - p_{\min}^l$, for every $s \in S$ and $k \geq 0$. We let $c_m \stackrel{\text{def}}{=} \|U_{S_\infty}^{(k_0+lm+l,k_0+lm)}\|_\infty$, and have by the above discussion that

$\sup_m c_m \leq 1 - p_{\min}^l < 1$, to which we now derive a contradiction. Let $x_m \stackrel{\text{def}}{=} \|\mathbf{u}_{S_\infty}^{k_0+lm}\|_\infty$, for which we have, by Lemma 6.10, that $x_{m+1} \leq c_m \cdot x_m + b_l$ for some $b_l \geq 0$. We now use Lemma 6.11 to obtain $\sup_m c_m = 1$, a contradiction. □

We now define Player 2 strategies and the expected energies they induce in terms of matrices. We consider *ultimately periodic* sequences of matrices that after a finite prefix $\mathfrak{n}$ keep repeating the same $\mathfrak{p}$ elements in a loop. Formally, an ultimately periodic sequence $(P^{[m]})_{m\geq 0}$ with *prefix* $\mathfrak{n}$ and *period* $\mathfrak{p}$ is such that the mth element is equal to the element of index $m \mod (\mathfrak{n}, \mathfrak{p})$ (that is, $P^{[m]} = P^{[m \mod (\mathfrak{n},\mathfrak{p})]}$), where

$$m \mod (\mathfrak{n}, \mathfrak{p}) \stackrel{\text{def}}{=} \begin{cases} m & \text{if } m \leq \mathfrak{n} + \mathfrak{p} - 1 \\ \mathfrak{n} + (m - \mathfrak{n} \mod \mathfrak{p}) & \text{otherwise.} \end{cases}$$

A stochastic matrix P *conforms* to $\mathcal{M}$ if, for every $s \in S_\bigcirc$ and all $s' \in \Delta(s)$, it holds that $P_{s,s'} = \Delta(s, s')$. We define a finite strategy by an ultimately periodic sequence of matrices $(P^{[k]})_{k\geq 0}$ that conform to $\mathcal{M}$: the memory is a counter $m \leq \mathfrak{n} + \mathfrak{p}$ that is updated at every step from m to $m + 1 \mod (\mathfrak{n}, \mathfrak{p})$; and in state s and memory $\mathfrak{m}$ the choice function selects s' with probability $P_{s,s'}^{[\mathfrak{m}]}$. To express several steps of the strategy, we define the interval matrices $P^{[m,m+l]} \stackrel{\text{def}}{=} P^{[m]} \cdots P^{[m+l-1]}$ with $P^{[m,m]} = I_S$, and corresponding cumulative matrices $\hat{P}^{[m,m+l]} \stackrel{\text{def}}{=} \sum_{q=0}^{l-1} P^{[m,m+q]}$ with $\hat{P}^{[m,m]} = 0$.

For every step $k \geq 0$ and memory $\mathfrak{m}$, we define a vector $\mathbf{e}_{(\mathfrak{m})}^k(r)$, where the entry for s is defined as $e_{s,\mathfrak{m}}^k$ in the game with reward structure r, that is, the expected energy for r after k steps at state $(s, \mathfrak{m})$ of the induced DTMC. We show in Lemma 6.14 that the strategy based on ultimately periodic matrices decreases the expected energy in the periodic phase by a non-zero amount every $\mathfrak{p}$ steps. The proof of Lemma 6.14 relies on the following technical lemma.

Lemma 6.13. *Given an ultimately periodic matrix based strategy with prefix $\mathfrak{n}$ and period $\mathfrak{p}$, it holds that $\mathbf{e}_{(m \mod (\mathfrak{n},\mathfrak{p}))}^l(r) = \hat{P}^{[m,m+l]} \cdot \mathbf{r}$, for all $l \geq 0$ and $m \geq 0$.*

Proof. We show this statement by induction on l. The base case for $l = 0$ is satisfied. Now assume the statement holds for l, and we show for $l + 1$. As the strategy with memory $m \mod (\mathfrak{n}, \mathfrak{p})$ plays according to the matrix $P^{[m]}$, and increments its memory

to $m+1 \mod (\mathfrak{n},\mathfrak{p})$, it holds that

$$\begin{aligned}\mathbf{e}^{l+1}_{(m \mod (\mathfrak{n},\mathfrak{p}))}(r) &= \mathbf{r} + P^{[m]} \cdot \mathbf{e}^{l}_{(m+1 \mod (\mathfrak{n},\mathfrak{p}))}(r) \\ &= P^{[m]} \cdot \hat{P}^{[m+1,m+l+1]} \cdot \mathbf{r} \\ &= \hat{P}^{[m,m+l+1]} \cdot \mathbf{r}.\end{aligned}$$

□

Lemma 6.14. *Given an ultimately periodic matrix based strategy with prefix $\mathfrak{n}$ and period $\mathfrak{p}$, and a set E such that $A = P_E^{[\mathfrak{n},\mathfrak{n}+\mathfrak{p}]}$ is stochastic, then, for all $j \geq 0$, it holds that $[e^{j\mathfrak{p}}_{(\mathfrak{n})}(r-\varepsilon)]_E = \sum_{k=0}^{j-1} A^k \cdot [\hat{P}^{[\mathfrak{n},\mathfrak{n}+\mathfrak{p}]} \cdot \mathbf{r}]_E - j\mathfrak{p}\varepsilon$.*

Proof. Note first that $P^{[\mathfrak{n},\mathfrak{n}+j\mathfrak{p}]} = (P^{[\mathfrak{n},\mathfrak{n}+\mathfrak{p}]})^j$, that $\hat{P}^{[\mathfrak{n},\mathfrak{n}+j\mathfrak{p}]} \cdot \mathbf{1} = j\mathfrak{p}$, and that $\hat{P}^{[\mathfrak{n},\mathfrak{n}+j\mathfrak{p}]} = \sum_{k=0}^{j-1} (P^{[\mathfrak{n},\mathfrak{n}+\mathfrak{p}]})^k \cdot \hat{P}^{[\mathfrak{n},\mathfrak{n}+\mathfrak{p}]}$. Since the restriction of $P^{[\mathfrak{n},\mathfrak{n}+\mathfrak{p}]}$ to the set E is stochastic, it holds, for every vector $\mathbf{x}$, that $[P^{[\mathfrak{n},\mathfrak{n}+\mathfrak{p}]} \cdot \mathbf{x}]_E = P_E^{[\mathfrak{n},\mathfrak{n}+\mathfrak{p}]} \cdot \mathbf{x}_E$. We apply Lemma 6.13 with $l = j\mathfrak{p}$ and $m = \mathfrak{n}$, and thus get, for all $j \geq 0$, that

$$\begin{aligned}[\mathbf{e}^{j\mathfrak{p}}_{(\mathfrak{n})}(r-\varepsilon)]_E &= [\hat{P}^{[\mathfrak{n},\mathfrak{n}+j\mathfrak{p}]} \cdot (\mathbf{r} - \boldsymbol{\varepsilon})]_E \\ &= \left[\sum_{k=0}^{j-1} (P^{[\mathfrak{n},\mathfrak{n}+\mathfrak{p}]})^k \cdot \hat{P}^{[\mathfrak{n},\mathfrak{n}+\mathfrak{p}]} \cdot \mathbf{r} - j\mathfrak{p}\boldsymbol{\varepsilon}\right]_E \\ &= \sum_{k=0}^{j-1} (P_E^{[\mathfrak{n},\mathfrak{n}+\mathfrak{p}]})^k \cdot [\hat{P}^{[\mathfrak{n},\mathfrak{n}+\mathfrak{p}]} \cdot \mathbf{r}]_E - j\mathfrak{p}\boldsymbol{\varepsilon}.\end{aligned}$$

□

We now describe a situation where truncation in the definition of $\mathbf{u}^k$ do not occur.

Lemma 6.15. *For $k \geq k_0$, and $E \subseteq S_\infty$ such that $U_E^{(k+\mathfrak{p},k)}$ is stochastic, it holds that $\mathbf{u}_E^{k+\mathfrak{p}} = [\hat{U}^{(k+\mathfrak{p},k)} \cdot \mathbf{r}]_E + U_E^{(k+\mathfrak{p},k)} \cdot \mathbf{u}_E^k$.*

Proof. We show, by induction on l, the following more general statement: for all $l \geq 0$, $k \geq k_0$, and $E, E' \subseteq S_\infty$ such that $E' = \texttt{Reach}(E, U^{(k+l,k)})$, it holds that

$$\mathbf{u}_E^{k+l} = [\hat{U}^{(k+l,k)} \cdot \mathbf{r}]_E + U_{E,E'}^{(k+l,k)} \cdot \mathbf{u}_{E'}^k.$$

The base case for $l = 0$ is straightforward. Now suppose that the result holds for l, and we show it for $l+1$. Let $k \geq k_0$ and $E, E' \subseteq S_\infty$ such that $E' = \texttt{Reach}(E, U^{(k+l+1,k)})$, and let $E'' = \texttt{Reach}(E, U^{(k+l+1,k+1)})$. Note that $\texttt{Reach}(E'', U^{(k)}) = E' \subseteq S_\infty$, and hence that $E'' \subseteq S_\infty$, since all predecessors of states in S_∞ are in S_∞. As $k+1 \geq k_0$

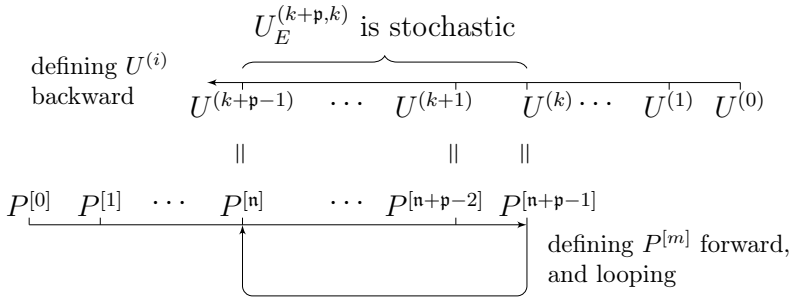


Figure 6.2: Matrices $U^{(i)}$ and $P^{[m]}$ to define the ultimately periodic matrix based strategy for Player 2 to spoil $\mathsf{EE}(r-\varepsilon)$ in the proof of Proposition 6.8.

and $E'' \subseteq S_\infty$, Remark 6.9 yields $\mathbf{u}_{E''}^{k+1} = \mathbf{r}_{E''} + U_{E'',E'}^{(k+1)} \mathbf{u}_{E'}^{k}$, and so we can conclude by

$$\begin{aligned}
\mathbf{u}_E^{k+l+1} &= [\hat{U}^{(k+l+1,k+1)} \cdot \mathbf{r}]_E + U_{E,E''}^{(k+l+1,k+1)} \cdot \mathbf{u}_{E''}^{k+1} \\
&= [\hat{U}^{(k+l+1,k+1)} \cdot \mathbf{r}]_E + U_{E,E''}^{(k+l+1,k+1)} \cdot \mathbf{r}_{E''} + U_{E,E''}^{(k+l+1,k+1)} \cdot U_{E'',E'}^{(k+1)} \cdot \mathbf{u}_{E'}^{k} \\
&= [\hat{U}^{(k+l+1,k)} \cdot \mathbf{r}]_E + U_{E,E'}^{(k+l+1,k)} \cdot \mathbf{u}_{E'}^{k},
\end{aligned}$$

where the first equality is due to the induction hypothesis. □

We complete the proof of Proposition 6.8. Suppose $S_\infty \neq \emptyset$. By Lemma 6.12, there exists a set $E \subseteq S_\infty$, and indices $k_0 \leq k < k + \mathfrak{p}$, such that $\mathtt{Reach}(U^{(k+\mathfrak{p},k)}, E) = E$. By Lemma 6.15, we have $\mathbf{u}_E^{k+\mathfrak{p}} = \mathbf{y} + A \cdot \mathbf{u}_E^k$ with $\mathbf{y} = [\hat{U}^{(k+\mathfrak{p},k)} \cdot \mathbf{r}]_E$ and $A = U_E^{(k+\mathfrak{p},k)}$.

We define an ultimately periodic matrix based Player 2 strategy σ based on the matrices $U^{(k+\mathfrak{p})}, \ldots, U^{(k+1)}$ (involved in the definition of A). The prefix of this strategy ensures that the set E is reachable from the initial state, and hence that the states of E are in the induced DTMC $\mathcal{M}^\sigma$. Let $P^{[0]}, \ldots P^{[\mathfrak{n}-1]}$ be matrices conform to $\mathcal{M}$, such that $E \cap \mathtt{Reach}(P^{[0,\mathfrak{n}-1]}, s_{\mathsf{init}}) \neq \emptyset$; for instance, we can take $P^{[i]}$ to be the matrix corresponding to choosing successors in Player 2 states with uniform probability. Then we define the periodic phase of σ with $\mathfrak{p}$ matrices by letting $P^{[\mathfrak{n}+i]} = U^{(k+\mathfrak{p}-i)}$ for $0 \leq i \leq \mathfrak{p}-1$. We illustrate the matrices used to construct σ in Figure 6.2. Note that $P^{[\mathfrak{n},\mathfrak{n}+\mathfrak{p}]} = U^{(k+\mathfrak{p},k)}$ and $\mathbf{y} \stackrel{\text{def}}{=} [\hat{U}^{(k+\mathfrak{p},k)} \cdot \mathbf{r}]_E = [\hat{P}^{[\mathfrak{n},\mathfrak{n}+\mathfrak{p}]} \cdot \mathbf{r}]_E$. Further, for states $s \in E \cap \mathtt{Reach}(\{s_{\mathsf{init}}\}, P^{[0,\mathfrak{n}-1]})$, the state $(s, \mathfrak{n})$ is in $\mathcal{M}^\sigma$.

We now show that $e_{s,\mathfrak{n}}^{j\mathfrak{p}} \to -\infty$ as $j \to \infty$, and hence that the strategy σ spoils $\mathsf{EE}(r-\varepsilon)$. From Lemma 6.14 we have $[e_{(\mathfrak{n})}^{j\mathfrak{p}}(r-\varepsilon)]_E = \sum_{k=0}^{j-1} A^k \cdot \mathbf{y} - j\mathfrak{p}\varepsilon$. It remains to show that the sequence $\sum_{k=0}^{j-1} A^k \cdot \mathbf{y}$ is upper-bounded, in order to have convergence

of $\mathbf{e}_{s,\mathfrak{n}}^{j\mathfrak{p}}$ toward $-\infty$ as $j \to \infty$. Since $\mathbf{y} = \mathbf{u}_E^{k+\mathfrak{p}} - A \cdot \mathbf{u}_E^k \leq (I_E - A) \cdot \mathbf{u}_E^k$, we have

$$\left(\sum_{i=0}^{j-1} A^i\right) \cdot \mathbf{y} \leq \left(\sum_{i=0}^{j-1} A^i\right) \cdot (I_E - A) \cdot \mathbf{u}_E^k = (I_E - A^j) \cdot \mathbf{u}_E^k \leq -A^j \cdot \mathbf{u}_E^k \leq \|\mathbf{u}_E^k\|_\infty \cdot \mathbf{1},$$

where we use for the last inequality that $\|A^j\|_\infty = 1$, since A^j is stochastic. □

Finally, in Proposition 6.17 we show that it is sufficient to consider finite memory Player 2 strategies for EE objectives. We first show in Lemma 6.16 that the precondition $S_\infty \neq \emptyset$ of Proposition 6.8 is satisfied if Player 2 can spoil the EE objective.

Lemma 6.16. *If Player 2 can spoil EE(r) in a finite PA with a one-dimensional reward structure r, then $S_\infty \neq \emptyset$.*

Proof. Fix a Player 2 strategy σ for $\mathcal{M}$, and let Δ' be the transition function of the induced DTMC $\mathcal{M}^\sigma$. We first show by induction on k that $u_s^k \leq e_{s,\mathfrak{m}}^k$ for every s and $\mathfrak{m}$. The base case for $k = 0$ is satisfied as $e_{s,\mathfrak{m}}^0 = u_s^0 = 0$. Now assume $u_s^k \leq e_{s,\mathfrak{m}}^k$ holds for some k and for every s and $\mathfrak{m}$, and we show for $k+1$. In each Player 2 state s,

$$\begin{aligned}
u_s^{k+1} &\leq r(s) + \min_{t \in \Delta(s)} u_t^k \\
&\leq r(s) + \sum_{(t,\mathfrak{m}') \in \Delta'(s,\mathfrak{m})} \Delta'((s,\mathfrak{m}),(t,\mathfrak{m}')) \cdot u_t^k \\
&\leq r(s) + \sum_{(t,\mathfrak{m}') \in \Delta'(s,\mathfrak{m})} \Delta'((s,\mathfrak{m}),(t,\mathfrak{m}')) \cdot e_{t,\mathfrak{m}'}^k \quad \text{(induction hypothesis)} \\
&= e_{s,\mathfrak{m}}^{k+1}.
\end{aligned}$$

Since Player 2 can falsify EE(r), for every v_0, there is $(s,\mathfrak{m})$ such that $e_{s,\mathfrak{m}}^k \leq v_0$ and hence $u_s^* \leq u_s^k \leq e_{s,\mathfrak{m}}^k \leq v_0$. As $\mathcal{M}$ is finite and v_0 can be taken arbitrary low, it means that there is at least one state s for which $u_s^* = -\infty$, and thus $S_\infty \neq \emptyset$. □

Proposition 6.17. *Let $\varepsilon > 0$. If a finite DU Player 1 strategy π achieves EE$(\vec{r} - \varepsilon)$ against finite Player 2 strategies, then π achieves EE$(\vec{r})$ against all Player 2 strategies.*

Proof. We show the contrapositive. Assume the strategy π loses for EE$(\vec{r})$ against an arbitrary strategy of Player 2. Then there is a coordinate r of the rewards $\vec{r}$ such that Player 2 achieves EE(r) in the induced PA $\mathcal{G}^\pi$. Then, by Lemma 6.16, $S_\infty \neq \emptyset$, and thus, by Proposition 6.8, Player 2 wins EE$(r - \varepsilon)$ for every ε, with a finite strategy. □

6.2.2 Transforming Between Pmp and EE

We now formalise the correspondence of the expected energy (EE) objectives with almost sure satisfaction of mean-payoff (Pmp) objectives. The following lemma establishes soundness of the transformation in part (i), and completeness up to ε-optimality of the strategies in part (ii). To obtain finite induced DTMCs, we use Theorem 6.2 to work with MD Player 2 strategies for Pmp CQs, and Proposition 6.17, to work with finite Player 2 strategies for EE CQs.

Proposition 6.18. *Given a finite strategy π for Player 1, the following hold:*

(i) if π achieves $\mathsf{EE}(\vec{r})$, then π achieves $\mathsf{Pmp}(\vec{r})$; and

(ii) if π is DU and achieves $\mathsf{Pmp}(\vec{r})$, then π achieves $\mathsf{EE}(\vec{r}+\vec{\varepsilon})$ for all $\varepsilon > 0$.

Proof. Instead of proving $\forall\sigma\,.\,\mathcal{G}^{(\pi,\sigma)} \models \psi \Rightarrow \forall\sigma\,.\,\mathcal{G}^{(\pi,\sigma)} \models \varphi$, we prove the stronger statement $\forall\sigma\,.\,(\mathcal{G}^{(\pi,\sigma)} \models \psi \Rightarrow \mathcal{G}^{(\pi,\sigma)} \models \varphi)$. Fix finite strategies π and σ. Let $\mathcal{D} = \mathcal{G}^{(\pi,\sigma)}$, which is a finite DTMC. By Lemma 5.2, the limit $\lim_{N\to\infty} \frac{1}{N+1}\mathsf{rew}^N(\vec{r})$ almost surely exists. For every N and path λ, we have $|\frac{1}{N+1}\mathsf{rew}^N(\vec{r})(\lambda)| \leq \max_{s\in S_\mathcal{D}} |\vec{r}(s)|$, where the maximum is taken componentwise, and so we have

$$\mathbb{E}_{\mathcal{D},s}\left[\lim_{N\to\infty} \tfrac{1}{N+1}\mathsf{rew}^N(\vec{r})\right] = \lim_{N\to\infty} \mathbb{E}_{\mathcal{D},s}\left[\tfrac{1}{N+1}\mathsf{rew}^N(\vec{r})\right] \tag{6.4}$$

by the Lebesgue dominated convergence theorem (Lemma 3.2).

Proof of (i). By Theorem 6.2 it suffices to consider MD Player 2 strategies. Assume that $\mathsf{EE}(\vec{r})$ is satisfied. Fix a finite shortfall $\vec{v}_0$ such that, for all $s \in S_\mathcal{D}$, it holds that

$$\begin{aligned}
&\forall N \geq 0\,.\,\mathbb{E}_{\mathcal{D},s}[\mathsf{rew}^N(\vec{r})] \geq \vec{v}_0 && \text{(by assumption)}\\
&\forall N \geq 0\,.\,\mathbb{E}_{\mathcal{D},s}[\tfrac{1}{N+1}\mathsf{rew}^N(\vec{r})] \geq \tfrac{\vec{v}_0}{N+1} && \text{(dividing by } N+1)\\
&\lim_{N\to\infty} \mathbb{E}_{\mathcal{D},s}[\tfrac{1}{N+1}\mathsf{rew}^N(\vec{r})] \geq 0 && \text{(taking limits)}\\
&\mathbb{E}_{\mathcal{D},s}[\lim_{N\to\infty} \tfrac{1}{N+1}\mathsf{rew}^N(\vec{r})] \geq 0. && \text{(by (6.4))}
\end{aligned}$$

From Lemma 5.2, whenever s is in a BSCC $\mathcal{B}$ of $\mathcal{D}$ (that is, $\mathbb{P}_{\mathcal{D},s}(\mathsf{F}\,\mathcal{B}) = 1$), we have $\mathsf{mp}(\vec{r})(\mathcal{B}) = \mathbb{E}_{\mathcal{D},s}[\lim_{N\to\infty} \frac{1}{N+1}\mathsf{rew}^N(\vec{r})]$. Therefore, for every BSCC $\mathcal{B}$, $\mathsf{mp}(\vec{r})(\mathcal{B}) \geq \vec{0}$. Thus, again by Lemma 5.2, $\mathsf{Pmp}(\vec{r})$ is satisfied.

Proof of (ii). Assume π is DU, and so, by Proposition 6.17, it suffices to consider finite Player 2 strategies. Fix $\varepsilon > 0$. Assume that $\mathcal{D} \models \mathsf{Pmp}(\vec{r})$, and so, by Lemma 5.2,

$\mathsf{rew}(\vec{r})(\mathcal{B}) \geq 0$ for every BSCC $\mathcal{B}$ of $\mathcal{D}$. Thus, for all states $s \in S_{\mathcal{D}}$, we have

$$\begin{aligned}
&\lim_{N\to\infty} \mathbb{E}_{\mathcal{D},s}[\tfrac{1}{N+1}\mathsf{rew}^N(\vec{r})] \geq \vec{0} && \text{(by (6.4))}\\
&\exists N_{\varepsilon,s} \geq 0 \,.\, \forall N \geq N_{\varepsilon,s} \,.\, \mathbb{E}_{\mathcal{D},s}[\tfrac{1}{N+1}\mathsf{rew}^N(\vec{r})] \geq -\vec{\varepsilon} && \text{(definition of limit)}\\
&\forall N \geq 0 \,.\, \mathbb{E}_{\mathcal{D},s}[\mathsf{rew}^N(\vec{r})] \geq -(N+1)\cdot\vec{\varepsilon} + \vec{v}_0^s &&\\
&\qquad\qquad \text{(fixing } N_{\varepsilon,s} \text{ and letting } v_{0,i}^s \stackrel{\text{def}}{=} \min_{N\leq N_{\varepsilon,s}} \mathbb{E}_{\mathcal{D},s}[\mathsf{rew}^N(r_i)]\text{)} &&\\
&\forall N \geq 0 \,.\, \mathbb{E}_{\mathcal{D},s}[\mathsf{rew}^N(\vec{r}+\varepsilon)] \geq \vec{v}_0^s \geq \vec{v}_0. && \text{(letting } v_{0,i} \stackrel{\text{def}}{=} \min_{s\in S_{\mathcal{D}}} v_{0,i}^s\text{)}
\end{aligned}$$

Since $\vec{v}_0$ is finite, $\mathcal{D}$ satisfies $\mathsf{EE}(\vec{r}+\vec{\varepsilon})$. □

6.2.3 Shortfall Characterisation

In Section 6.2.1 we used a one-dimensional version of the expected truncated energy to establish that finite strategies suffice for Player 2. To construct Player 1 strategies for an n-dimensional Pmp CQ, we have to consider all n dimensions simultaneously. We show below in Proposition 6.23 that having a lower bound on the expected truncated energy at the initial state is sufficient to be able to construct a strategy that only reaches states with lower-bounded expected energy (states with no lower bound can be avoided by the strategy). The intuition is that, by truncating positive values, divergence towards negative infinity in any BSCC that is reached with non-zero probability is sufficient to cause divergence towards negative infinity at the initial state. We go on to construct strategies from the shortfalls of the expected truncated energy, in Section 6.3, and prove that they satisfy the corresponding EE objective.

Bellman Operator. We define a Bellman operator to characterise sets of shortfalls for each state of a game, for which we use a CPO of $|S|$-dimensional vectors of subsets of $\mathbb{R}^n$, defined as follows. Given $M \geq 0$ and a set $X \subseteq \mathbb{R}^n$, define the *M-downward closure* of X by $\mathsf{dwc}(X) \cap \mathsf{Box}_M$, where $\mathsf{Box}_M \stackrel{\text{def}}{=} [-M, 0]^n$. Let $\mathcal{P}_{c,M}$ be the set of convex, closed, M-downward-closed subsets of $\mathbb{R}^n$. We define the CPO $\mathcal{C}_M$ to be the set $\mathcal{P}_{c,M}^{|S|}$, with the bottom element $\bot_M \stackrel{\text{def}}{=} \mathsf{Box}_M^{|S|}$, and which we endow with the partial order $\sqsubseteq$ defined as $Y \sqsubseteq X \Leftrightarrow \forall s \in S \,.\, \mathsf{dwc}(X_s) \subseteq \mathsf{dwc}(Y_s)$. For $D \subseteq \mathcal{C}_M$, the supremum $\sup D$ is defined via $[\sup D]_s \stackrel{\text{def}}{=} \bigcap_{X\in D} X_s$ for all $s \in S$. The intersection of convex, closed, M-downward-closed sets is itself convex, closed, M-downward-closed, and so $\sup D \in \mathcal{C}_M$ for any directed set D. Hence, $\mathcal{C}_M$ is a CPO.

We now define the Bellman operator $F_{M,\vec{r},\mathcal{G}} : \mathcal{C}_M \to \mathcal{C}_M$, for a game $\mathcal{G}$, rewards $\vec{r}$, and $M \geq 0$, as follows. For all player and stochastic states $s \in S$ of the game $\mathcal{G}$, let

$$[F_{M,\vec{r},\mathcal{G}}(X)]_s \stackrel{\text{def}}{=} \mathsf{Box}_M \cap \mathsf{dwc}\left(\vec{r}(s) + \begin{cases} \mathsf{conv}(\bigcup_{t\in\Delta(s)} X_t) & \text{if } s \in S_\Diamond \\ \bigcap_{t\in\Delta(s)} X_t & \text{if } s \in S_\Box \\ \sum_{t\in\Delta(s)} \Delta(s,t) \times X_t & \text{if } s \in S_\bigcirc \end{cases}\right).$$

If the game $\mathcal{G}$ is clear from context, we write $F_{M,\vec{r}}$ instead of $F_{M,\vec{r},\mathcal{G}}$. The operator $F_{M,\vec{r}}$ computes the shortfalls for the expected truncated energy that Player 1 can achieve in the respective state types. In $s \in S_\Diamond$, Player 1 can achieve the values in successors (union), and can randomise between them (convex hull). In $s \in S_\Box$, Player 1 can achieve only values that are in all successors (intersection), since Player 2 can pick arbitrarily. Lastly, in $s \in S_\bigcirc$, Player 1 can achieve values with the prescribed distribution. The operator $F_{M,\vec{r}}$ is closely related to the operator $F_{\mathsf{rew},\vec{r}}$ for expected total rewards in [41] but here we cut off values outside of Box_M, similarly to the controllable predecessor operator of [36] for non-stochastic games (see $F_{\mathsf{ens},M,\vec{r}}$ in Section 3.5.3).

Fixpoint of $F_{M,\vec{r}}$. We show that the fixpoint of $F_{M,\vec{r}}$ exists. A further consequence of our proof is that $F_{M,\vec{r}}$ is monotonic, and so we can approximate the fixpoint to arbitrary accuracy, see Section 6.4.1.

Proposition 6.19. *$F_{M,\vec{r}}$ is order-preserving and the least fixpoint $\mathsf{fix}(F_{M,\vec{r}})$ exists.*

Proof. The claimed properties are consequences of Scott continuity of $F_{M,\vec{r}}$ and the Kleene fixpoint theorem, Lemma 3.10. To establish Scott continuity, it is sufficient to show that, for every countable directed set D, we have that $[F_{M,\vec{r}}(\sup D)]_s = \sup([F_{M,\vec{r}}(D)]_s)$ for all $s \in S$. Fix any countable directed set $D = \{X^k \in \mathcal{C}_M \mid k \geq 0\} \subseteq \mathcal{C}_M$, and any $s \in S$. We first show intermediate results about D.

Lemma 6.20. *For finite $T \subseteq S$, $\mathsf{conv}(\bigcup_{t\in T} \bigcap_{k\geq 0} X_t^k) = \bigcap_{k\geq 0} \mathsf{conv}(\bigcup_{t\in T} X_t^k)$.*

Proof. Let $Y^k \stackrel{\text{def}}{=} \mathsf{conv}(\bigcup_{t\in T} X_t^k)$ and define $Y^\infty \stackrel{\text{def}}{=} \bigcap_{k\geq 0} Y^k$. The sets X_t^k are compact and convex, and so their convex hull Y^k is also compact and convex, by Theorem 17.2 of [114]. Moreover, Y^k is M-downward closed, so, for every k, $Y^k \in \mathcal{P}_{c,M}$. We now show the equality in the lemma. For the $\subseteq$ direction, take $\vec{y} \in \mathsf{conv}(\bigcup_{t\in T} \bigcap_{k\geq 0} X_t^k)$. Then $\vec{y} = \sum_{t\in T} \mu(t) \cdot \vec{x}_t$ for some distribution $\mu \in \mathsf{D}(T)$ and some $\vec{x}_t \in \bigcap_{k\geq 0} X_t^k$. Hence, for every k, $\vec{y} \in Y^k$, and so, $\vec{y} \in Y^\infty$. For the $\supseteq$ direction, take $\vec{y}^\infty \in Y^\infty$. We note that, for every $k \geq 0$, $\vec{y}^\infty = \sum_{t\in T} \mu_k(t) \cdot \vec{x}_t^k$ for some distribution $\mu_k \in \mathsf{D}(T)$ and

some vector $\vec{x}_t^k \in X_t^k$. The sets X^k are in $\mathcal{P}_{c,M}$, and thus compact, and so one can extract a subsequence of indices i_k such that μ_{i_k} and $\vec{x}_t^{i_k}$ converge toward limits, which we respectively denote μ and $\vec{x}_t$ for every $t \in T$. Moreover, $\lim_{k\to\infty} \vec{x}_t^{i_k} = \vec{x}_t \in Y_t^l$ for every $l \geq 0$ as Y^l is compact. Hence, $\vec{x}_t \in \bigcap_{k\geq 0} X_t^k$ for every t and we conclude $\vec{y}^\infty = \sum_{t\in T} \mu(t) \cdot \vec{x}_t \in \mathsf{conv}(\bigcup_{t\in T} \bigcap_{k\geq 0} X_t^k)$. □

Lemma 6.21. *For finite $T \subseteq S$, $\bigcap_{t\in T} \bigcap_{k\geq 0} X_t^k = \bigcap_{k\geq 0} \bigcap_{t\in T} X_t^k$.*

Proof. Reordering of countable intersections. □

Lemma 6.22. *For finite $T \subseteq S$, $\sum_{t\in T} \mu(t) \times \bigcap_{k\geq 0} X_t^k = \bigcap_{k\geq 0} \sum_{t\in T} \mu(t) \times X_t^k$, where $\mu \in D(T)$.*

Proof. For the $\supseteq$ direction, let $\vec{x} \in \bigcap_{k\geq 0} \sum_{t\in T} \mu(t) \times X_t^k$, and so, for all $k \geq 0$, there exist vectors $\vec{x}_t^k \in X_t^k$ for $t \in T$, such that $\vec{x} = \sum_{t\in T} \mu(t) \cdot \vec{x}_t^k$. We extract a subsequence of indices i_k such that $\vec{x}_t^{i_k}$ tends to a limit $\vec{x}_t$, which necessarily lies in $\bigcap_{k\geq 0} X_t^k$, by the same argument as in Lemma 6.21. Hence $\vec{x} = \sum_{t\in T} \mu(t)\vec{x}_t \in \sum_{t\in T} \mu(t) \times \bigcap_{k\geq 0} X_t^k$. The $\subseteq$ direction is straightforward. □

We now continue the proof of Proposition 6.19 by cases. For $s \in S_\Diamond$,

$$\begin{aligned}
[F_{M,\vec{r}}(\sup(D))]_s &\stackrel{\text{def}}{=} \mathsf{Box}_M \cap \mathsf{dwc}(\vec{r}(s) + \mathsf{conv}(\textstyle\bigcup_{t\in\Delta(s)} \bigcap_{k\geq 0} X_t^k)) \\
&= \mathsf{Box}_M \cap \mathsf{dwc}(\vec{r}(s) + \textstyle\bigcap_{k\geq 0} \mathsf{conv}(\bigcup_{t\in\Delta(s)} X_t^k)) && \text{(Lemma 6.20)} \\
&= \textstyle\bigcap_{k\geq 0}(\mathsf{Box}_M \cap \mathsf{dwc}(\vec{r}(s) + \mathsf{conv}(\bigcup_{t\in\Delta(s)} X_t^k))) \\
&\stackrel{\text{def}}{=} [\sup F_{M,\vec{r}}(D)]_s.
\end{aligned}$$

For $s \in S_\Box$,

$$\begin{aligned}
[F_{M,\vec{r}}(\sup(D))]_s &\stackrel{\text{def}}{=} \mathsf{Box}_M \cap \mathsf{dwc}(\vec{r}(s) + \textstyle\bigcap_{t\in\Delta(s)} \bigcap_{k\geq 0} X_t^k) \\
&= \mathsf{Box}_M \cap \mathsf{dwc}(\vec{r}(s) + \textstyle\bigcap_{k\geq 0} \bigcap_{t\in\Delta(s)} X_t^k) && \text{(Lemma 6.21)} \\
&= \textstyle\bigcap_{k\geq 0}(\mathsf{Box}_M \cap \mathsf{dwc}(\vec{r}(s) + \bigcap_{t\in\Delta(s)} X_t^k)) \\
&\stackrel{\text{def}}{=} [\sup F_{M,\vec{r}}(D)]_s.
\end{aligned}$$

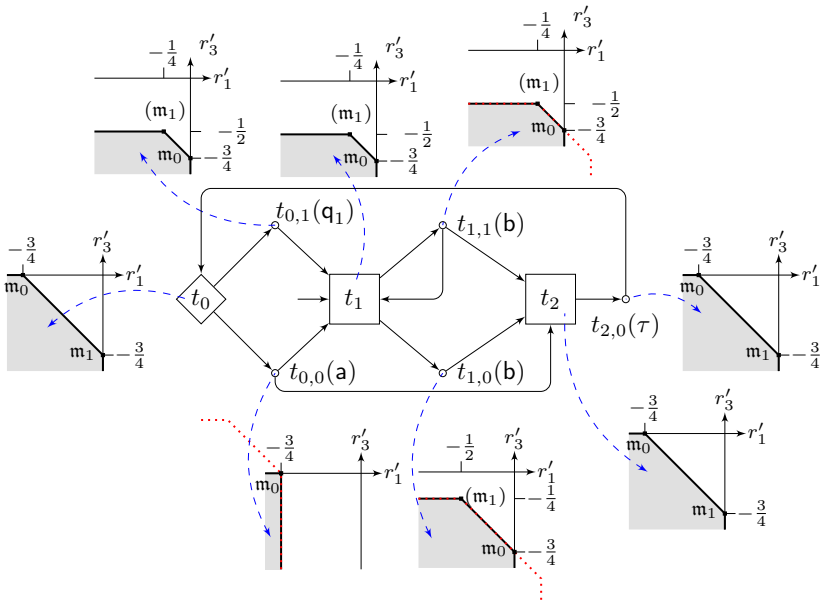


Figure 6.3: Fixpoint $\mathsf{fix}(F_{M,\vec{r},\mathcal{G}^2_{\mathsf{re}}})$ for shortfall sets in $\mathcal{G}^2_{\mathsf{re}}$ of Figure 3.8. For easier reference, moves are given state names. Each state s has an associated set $\mathsf{fix}(F_{M,\vec{r},\mathcal{G}^2_{\mathsf{re}}})(s)$ pointed to by the dashed arrows, and a memory mapping (used in Section 6.3) is annotated at the extreme points.

Finally, for $s \in S_{\bigcirc}$,

$$\begin{aligned}
[F_{M,\vec{r}}(\sup(D))]_s &\stackrel{\text{def}}{=} \mathsf{Box}_M \cap \mathsf{dwc}(\vec{r}(s) + \textstyle\sum_{t\in\Delta(s)} \Delta(s,t) \times \bigcap_{k\geq 0} X_t^k) \\
&= \mathsf{Box}_M \cap \mathsf{dwc}(\vec{r}(s) + \textstyle\bigcap_{k\geq 0} \sum_{t\in\Delta(s)} \Delta(s,t) \times X_t^k) \quad \text{(Lemma 6.22)} \\
&= \textstyle\bigcap_{k\geq 0}(\mathsf{Box}_M \cap \mathsf{dwc}(\vec{r}(s) + \sum_{t\in\Delta(s)} \Delta(s,t) \times X_t^k)) \\
&\stackrel{\text{def}}{=} [\sup F_{M,\vec{r}}(D)]_s.
\end{aligned}$$

This concludes the proof of Scott continuity for $F_{M,\vec{r}}$. Then, by Lemma 3.10, the least fixpoint exists, and is equal to $\mathsf{fix}(F_{M,\vec{r}}) = \bigcap_{k\geq 0} F^k_{M,\vec{r}}(\bot_M)$. □

Example$^{\mathsf{re}}$ 6.1. *Continuing our running example of Section 3.6, we show in Figure 6.3 the fixpoint of $F_{M,\vec{r},\mathcal{G}^2_{\mathsf{re}}}$ for the game $\mathcal{G}^2_{re}$. Recall the reward structures (defined on actions) $r_1(\mathsf{a}) = 1$, $r_3(\mathsf{b}) = 1$, $c(\mathsf{a}) = c(\mathsf{b}) = 1$, and zero otherwise (for the actions used in $\mathcal{G}^2_{re}$). The objective we are interested in is φ^2_{re} with ratioE objectives, that is, $\mathsf{ratioE}^{\leq}(r_1/c)(v_1) \wedge \mathsf{ratioE}^{\leq}(r_3/c)(v_3)$. We let the target be $(v_1, v_3) = (\frac{1}{4}, \frac{3}{4})$, which is achievable ε-optimally, as can be seen from the Pareto set in Figure 4.9. We use Theorem 5.18 to convert φ^2_{re} to the CQ $\mathsf{Pmp}(-r_1 + v_1 \cdot c) \wedge \mathsf{Pmp}(-r_3 + v_3 \cdot c)$; note*

that the signs of the rewards were inverted. We define new reward structures r'_1 and r'_3 by

$$\begin{aligned} r'_1(a) &= -r_1(a) + v_1 \cdot c(a) &&= -1 + \tfrac{1}{4} \cdot 1 = -\tfrac{3}{4}, \\ r'_1(b) &= -r_1(b) + v_1 \cdot c(b) &&= 0 + \tfrac{1}{4} \cdot 1 = \tfrac{1}{4}, \\ r'_3(a) &= -r_3(a) + v_3 \cdot c(a) &&= 0 + \tfrac{3}{4} \cdot 1 = \tfrac{3}{4}, \\ r'_3(b) &= -r_3(b) + v_3 \cdot c(b) &&= -1 + \tfrac{3}{4} \cdot 1 = -\tfrac{1}{4}, \end{aligned}$$

and zero otherwise. By Proposition 6.18, we need to consider the expected energy objective $\mathsf{EE}(r'_1, r'_3)$, and so we compute the fixpoint $\mathsf{fix}(F_{M,\vec{r},\mathcal{G}^2_{\mathsf{re}}})$ for the rewards $\vec{r} = (r'_1, r'_3)$, which we denote by X^*. Note that this sequence of transformations ensures that a strategy constructed from X^* achieves the original specification φ^2.

We now explain the fixpoint X^*. Since t_2 has only one move enabled, and since the τ-transition is followed by a Dirac, we have $X^*_{t_2} = X^*_{t_{2,0}} = X^*_{s_0}$. Similarly, $X^*_{t_{0,1}} = X^*_{t_1}$. Since $r'_1(b) = \frac{1}{4}$ and $r'_3(b) = -\frac{1}{4}$, the set $X^*_{t_{1,0}}$ is $X^*_{t_2}$ shifted by $(\frac{1}{4}, -\frac{1}{4})$, and then cut by Box_M; and the set $X^*_{t_{1,0}}$ is the uniformly weighted Minkowski sum of $X^*_{t_2}$ and $X^*_{t_1}$, shifted by $(\frac{1}{4}, -\frac{1}{4})$, and then cut by Box_M. Since $r'_1(a) = -\frac{3}{4}$ and $r'_3(a) = \frac{3}{4}$, the set $X^*_{t_{0,0}}$ is the uniformly weighted Minkowski sum of $X^*_{t_1}$ and $X^*_{t_2}$, shifted by $(-\frac{3}{4}, \frac{3}{4})$, and then cut by Box_M. For $X^*_{t_{1,0}}$, $X^*_{t_{1,1}}$, and $X^*_{t_{0,0}}$, the sets before cutting by Box_M are shown in red (dotted) in Figure 6.3. Since $X^*_{t_{1,1}}$ is a subset of $X^*_{t_{1,0}}$, we have that, for the Player 2 state t_1, $X^*_{t_1} = X^*_{t_{1,0}}$ due to the intersection operation of $F_{M,\vec{r}}$. Finally, $X^*_{t_0}$ is the convex union of $X^*_{t_{0,0}}$ and $X^*_{t_{0,1}}$.

In order to obtain a complete synthesis algorithm, we now show that whenever the expected energy (EE) objective is achievable, the fixpoint of $F_{M,\vec{r}}$ is nonempty for some box size M. Thus, if it is known that Pmp is achievable, by employing Proposition 6.18 (ii), we can compute $F_{M,\vec{r}}$ for a large enough M and construct an ε-optimal strategy for EE that is winning for Pmp by Proposition 6.18 (i).

Proposition 6.23. *Let $\mathcal{G}$ be a game with rewards $\vec{r}$. For every $\varepsilon > 0$, if $\mathsf{EE}(\vec{r} - \varepsilon)$ is achievable by a finite DU strategy, then $[\mathsf{fix}(F_{M,\vec{r},\mathcal{G}})]_{s_{\mathsf{init}}} \neq \emptyset$ for some $M \geq 0$.*

Proof. We first show two intermediate lemmas. In Lemma 6.24, we show that we can consider the fixpoints $\mathsf{fix}[F_{M,\vec{r},\mathcal{M}}]_s$ for PAs $\mathcal{M}$, which exist due to Proposition 6.19, and in Lemma 6.25 we reduce the problem to the study of the one-dimensional expected truncated energy, which we used earlier in Proposition 6.8 and Lemma 6.16.

Lemma 6.24. *Given a game $\mathcal{G}$ and a strategy π, if $[\mathsf{fix}(F_{M,\vec{r},\mathcal{G}^\pi})]_s \neq \emptyset$ for all states s of $\mathcal{G}^\pi$, then $[\mathsf{fix}(F_{M,\vec{r},\mathcal{G}})]_{s_{\mathsf{init}}} \neq \emptyset$.*

Proof. By a straightforward induction on k, for every $k \geq 0$ and every state $(s, \mathfrak{d}_\lambda)$ of $\mathcal{G}^\pi$, we have that $[F^k_{M,\vec{r},\mathcal{G}^\pi}(\bot_M)]_{(s,\mathfrak{d}_\lambda)} \subseteq [F^k_{M,\vec{r},\mathcal{G}}(\bot_M)]_{\mathsf{last}(\lambda)}$. Hence, $[\mathsf{fix}(F_{M,\vec{r},\mathcal{G}^\pi})]_{(s,\mathfrak{d}_\lambda)} \subseteq [\mathsf{fix}(F_{M,\vec{r},\mathcal{G}})]_{\mathsf{last}(\lambda)}$, for every state $(s, \mathfrak{d}_\lambda)$ of $\mathcal{G}^\pi$. Since, in particular, $[\mathsf{fix}(F_{M,\vec{r},\mathcal{G}^\pi})]_t \neq \emptyset$ for all t in the support of the initial distribution of $\mathcal{G}^\pi$, we have $[\mathsf{fix}(F_{M,\vec{r},\mathcal{G}})]_{s_{\mathsf{init}}} \neq \emptyset$, concluding the proof. □

Lemma 6.25. *Given a PA $\mathcal{M}$ with rewards $\vec{r}$ and a state s_o of $\mathcal{M}$, if $\mathsf{fix}[F_{M,\vec{r},\mathcal{M}}]_{s_o} = \emptyset$ for every $M < \infty$, then there exists i such that $u^*_{s_o} = -\infty$ for the reward r_i.*

Proof. Fix a PA $\mathcal{M} = \langle S, (S_\Box, S_\bigcirc), \varsigma, \mathcal{A}, \chi, \Delta \rangle$ with rewards $\vec{r}$, and let $s_o \in S$. We prove the lemma by contraposition: we assume that $u^*_{s_o} > -\infty$ for rewards r_i for all i, and show that there is an M for which $\mathsf{fix}[F_{M,\vec{r},\mathcal{M}}]_{s_o} \neq \emptyset$. We consider a multi-dimensional version of the truncated energy sequence defined in (6.1), and we get that the fixpoint of the multi-dimensional truncated energy, as $k \to \infty$, is

$$\vec{u}^*_s = \begin{cases} \min(\vec{0}, \vec{r}(s) + \min_{t \in \Delta(s)} \vec{u}^*_t) & \text{if } s \in S_\Box \\ \min(\vec{0}, \vec{r}(s) + \sum_{t \in \Delta(s)} \Delta(s,t) \cdot \vec{u}^*_t) & \text{if } s \in S_\bigcirc, \end{cases}$$

for every $s \in S$, where the minima are taken componentwise. Observe that, if $\vec{u}^*_s$ is finite for a state s, then $\vec{u}^*_t$ is finite for all successors t of s. Since all states of the PA are reachable from the initial state, $\vec{u}^*_s$ has no infinite coordinate, for every state s. Therefore, there is a global bound M, such that $\vec{u}^*_s \in \mathsf{Box}_M$ for every s. We now show that $Z \in \mathcal{C}_M$, defined by $Z_s \stackrel{\text{def}}{=} \mathsf{Box}_M \cap \mathsf{dwc}(\vec{u}^*_s)$, is a fixpoint of $F_{M,\vec{r},\mathcal{M}}$, and hence that the least fixpoint of $F_{M,\vec{r},\mathcal{M}}$ is nonempty. Taking the downward-closure gives

$$\mathsf{dwc}(\vec{u}^*_s) = \begin{cases} \mathbb{R}_{\leq 0} \cap (\vec{r}(s) + \bigcap_{t \in \Delta(s)} \mathsf{dwc}(\vec{u}^*_t)) & \text{if } s \in S_\Box \\ \mathbb{R}_{\leq 0} \cap (\vec{r}(s) + \sum_{t \in \Delta(s)} \Delta(s,t) \times \mathsf{dwc}(\vec{u}^*_t)) & \text{if } s \in S_\bigcirc, \end{cases}$$

and hence

$$Z_s = \begin{cases} \mathsf{Box}_M \cap (\vec{r}(s) + \bigcap_{t \in \Delta(s)} \mathsf{dwc}(\vec{u}^*_t)) & \text{if } s \in S_\Box \\ \mathsf{Box}_M \cap (\vec{r}(s) + \sum_{t \in \Delta(s)} \Delta(s,t) \times \mathsf{dwc}(\vec{u}^*_t)) & \text{if } s \in S_\bigcirc. \end{cases}$$

Since $\vec{u}^*_t \in \mathsf{Box}_M$, Z_t is nonempty, and we have

$$\begin{aligned} \vec{r}(s) + \textstyle\bigcap_{t \in \Delta(s)} \mathsf{dwc}(\vec{u}^*_t) &= \mathsf{dwc}(\vec{r}(s) + \textstyle\bigcap_{t \in \Delta(s)} Z_t) && \text{for } s \in S_\Box, \\ \vec{r}(s) + \textstyle\sum_{t \in \Delta(s)} \Delta(s,t) \times \mathsf{dwc}(\vec{u}^*_t) &= \mathsf{dwc}(\vec{r}(s) + \textstyle\sum_{t \in \Delta(s)} \Delta(s,t) \times Z_t) && \text{for } s \in S_\bigcirc. \end{aligned}$$

This implies that $Z = F_{M,\vec{r},\mathcal{M}}(Y)$, and hence that $\mathsf{fix}[F_{M,\vec{r},\mathcal{M}}]_{s_o} \sqsubseteq Z_{s_o}$. We thus conclude from $Z_{s_o} \neq \emptyset$ that $\mathsf{fix}[F_{M,\vec{r},\mathcal{M}}]_{s_o} \neq \emptyset$. □

We now conclude the proof of Proposition 6.23. Fix a game $\mathcal{G}$ with rewards $\vec{r}$ and $\varepsilon > 0$. We show the contrapositive, that is, Player 1 does not have a finite DU strategy achieving $\mathsf{EE}(\vec{r} - \varepsilon)$ whenever, for every $M \geq 0$, $[\mathsf{fix}(F_{M,\vec{r},\mathcal{G}})]_{s_{\mathsf{init}}} = \emptyset$. Assume that $[\mathsf{fix}(F_{M,\vec{r},\mathcal{G}})]_{s_{\mathsf{init}}} = \emptyset$ for all $M \geq 0$, and let π be an arbitrary finite DU strategy. By Lemma 6.24, $[\mathsf{fix}(F_{M,\vec{r},\mathcal{G}^\pi})]_{s_o} = \emptyset$ for some state s_o of $\mathcal{G}^\pi$. Thus, by Lemma 6.25, there is an index i such that $u^*_{s_o} = -\infty$ for the reward r_i, and hence $S_\infty \neq \emptyset$. We conclude, using Proposition 6.8 that Player 2 can spoil $\mathsf{EE}(r - \varepsilon)$ in the PA $\mathcal{G}^\pi$. □

6.3 Strategy Construction

We construct strategies for $\mathsf{Pmp}(\vec{r})$ that operate by maintaining finite shortfalls for the expected energy. In Proposition 6.23 we showed that the fixpoint $\mathsf{fix}(F_{M,\vec{r}})$ is nonempty if the corresponding EE objective is ε-achievable, and so our strategy construction uses approximations to $\mathsf{fix}(F_{M,\vec{r}})$. We justify the correctness of our construction using a mapping from memory elements of the strategy to shortfall vectors.

6.3.1 Geometric Interpretation of Strategies

We map each memory element of a strategy to a vector, and if we can find an appropriate mapping, the geometric interpretation allows us to prove that the strategy achieves an EE objective. Then, in our strategy construction, we find such vectors from computing an approximation of $\mathsf{fix}(F_{M,\vec{r}})$, which exists due to Proposition 6.23.

Given a game $\mathcal{G} = \langle S, (S_\Diamond, S_\Box, S_\bigcirc), s_{\mathsf{init}}, \mathcal{A}, \chi, \Delta \rangle$, a strategy $\pi = \langle \mathfrak{M}, \pi_{\mathsf{c}}, \pi_{\mathsf{u}}, \pi_{\mathsf{d}} \rangle$, and an n-dimensional reward structure $\vec{r}$, a *memory mapping* is a partial function $f_\pi : \mathfrak{M} \times S \to \mathbb{R}^n_{\leq 0}$. We typically write $f_\pi(\mathfrak{m}, s) = \vec{\mathfrak{m}}_s$. A memory mapping is *$\varepsilon$-consistent* for $\vec{v}_0 \in \mathbb{R}^n_{\leq 0}$, if $\sum_{\mathfrak{m} \in \mathfrak{M}} \vec{\mathfrak{m}}_{s_{\mathsf{init}}} \cdot \pi_{\mathsf{d}}(s_{\mathsf{init}})(\mathfrak{m}) \geq \vec{v}_0$, and for all $s \in S$, $\mathfrak{m} \in \mathfrak{M}$,

$$\begin{array}{lll} \sum_{t \in \Delta(s)} \pi_{\mathsf{c}}(s, \mathfrak{m})(t) \cdot \sum_{\mathfrak{m}' \in \mathfrak{M}} \pi_{\mathsf{u}}(\mathfrak{m}, t)(\mathfrak{m}') \cdot \vec{\mathfrak{m}}'_t \geq \vec{\mathfrak{m}}_s - \vec{r}(s) - \varepsilon & \text{if } s \in S_\Diamond \\ \max_{t \in \Delta(s)} \quad \sum_{\mathfrak{m}' \in \mathfrak{M}} \pi_{\mathsf{u}}(\mathfrak{m}, t)(\mathfrak{m}') \cdot \vec{\mathfrak{m}}'_t \geq \vec{\mathfrak{m}}_s - \vec{r}(s) - \varepsilon & \text{if } s \in S_\Box \\ \sum_{t \in \Delta(s)} \Delta(s, t) \cdot \quad \sum_{\mathfrak{m}' \in \mathfrak{M}} \pi_{\mathsf{u}}(\mathfrak{m}, t)(\mathfrak{m}') \cdot \vec{\mathfrak{m}}'_t \geq \vec{\mathfrak{m}}_s - \vec{r}(s) - \varepsilon & \text{if } s \in S_\bigcirc. \end{array}$$

Lemma 6.26. *Given a strategy π with a memory mapping f_π, if f_π is ε-consistent for some $\vec{v}_0 < -\infty$, then π achieves $\mathsf{EE}(\vec{r} + \varepsilon)$.*

Proof. Fix a game $\mathcal{G} = \langle S, (S_\Diamond, S_\Box, S_\bigcirc), s_{\mathsf{init}}, \mathcal{A}, \chi, \Delta \rangle$ with rewards $\vec{r} : S \to \mathbb{R}^n$, let $\pi = \langle \mathfrak{M}, \pi_{\mathsf{c}}, \pi_{\mathsf{u}}, \pi_{\mathsf{d}} \rangle$ be a Player 1 strategy, and let f_π be an ε-consistent memory mapping for $\vec{v}_0 < -\infty$. For each state s of $\mathcal{G}$, we write $\vec{\mathfrak{m}}_s = f_\pi(\mathfrak{m}, s)$. Let σ be a Player 2 strategy, let $\mathcal{D} \stackrel{\text{def}}{=} \mathcal{G}^{(\pi,\sigma)}$ be the induced DTMC, and let s_o be a state of $\mathcal{D}$, which is of the form $s_o = (p_o, \vec{\mathfrak{m}}_{p_o}, \mathfrak{n})$, for some memory $\mathfrak{n}$ of Player 2. We show that $\mathbb{E}_{\mathcal{D},s_o}[\mathsf{rew}^N(\vec{r})] \geq \vec{\mathfrak{m}}_{p_o} - N\varepsilon$ for all $N \geq 0$, by showing that, at every step N, the memory of π maps to a non-negative vector above $\vec{\mathfrak{m}}_{p_o} - \mathbb{E}_{\mathcal{D},s_o}[\mathsf{rew}^N(\vec{r})] - N\varepsilon$.

Let $V_N : \Omega_\mathcal{D} \to \mathbb{R}^n$ be the random variable that assigns $\vec{\mathfrak{m}}_s$ to a path $\lambda = s_0 s_1 \ldots$ for which $s_N = (s, \vec{\mathfrak{m}}_s, \mathfrak{n})$. Since $\mathbb{E}_{\mathcal{D},s_o}[V_N] \leq \vec{0}$ for all $N \geq 0$, it is sufficient to show, for all states s_o of $\mathcal{D}$, that

$$\mathbb{E}_{\mathcal{D},s_o}[V_N] \geq \vec{\mathfrak{m}}_{p_o} - \mathbb{E}_{D,s_o}[\mathsf{rew}^N(\vec{r})] - N\varepsilon, \tag{6.5}$$

in order to conclude that $\mathbb{E}_{\mathcal{D},s_o}[\mathsf{rew}^N(r \stackrel{\rightarrow}{+} \varepsilon)] \geq \vec{0}$, and thus that $\mathcal{D}$ satisfies $\mathsf{EE}(\vec{r} + \vec{\varepsilon})$. We show (6.5) by induction on the length N of paths $\Omega_\mathcal{D}$. In the base case, for $N = 0$, we have $\mathbb{E}_{\mathcal{D},s_o}[V_0] = \vec{\mathfrak{m}}_{p_o}$, corresponding to the memory at the initial state s_o. For the induction step, assume that $\mathbb{E}_{\mathcal{D},s_o}[V_N] \geq \vec{\mathfrak{m}}_{p_o} - \mathbb{E}_{\mathcal{D},s_o}[\mathsf{rew}^N(\vec{r})] - N\varepsilon$. Let W_N be the set of all finite paths of length N in $\mathcal{D}$. Let $\lambda' \in W_N$, which is of the form $\lambda' = \lambda(s, \vec{\mathfrak{m}}_s, \mathfrak{n})$. We have

$$\mathbb{E}_{\mathcal{D},s_o}[V_{N+1}|\lambda'] = \begin{cases} \sum_{t \in \Delta(s)} \pi_{\mathsf{c}}(s, \mathfrak{m})(t) \cdot \sum_{\mathfrak{m}' \in \mathfrak{M}} \pi_{\mathsf{u}}(\mathfrak{m}, t)(\mathfrak{m}') \cdot \vec{\mathfrak{m}}'_t & \text{if } s \in S_\Diamond \\ \sum_{t \in \Delta(s)} \sigma_{\mathsf{c}}(s, \mathfrak{n})(t) \cdot \sum_{\mathfrak{m}' \in \mathfrak{M}} \pi_{\mathsf{u}}(\mathfrak{m}, t)(\mathfrak{m}') \cdot \vec{\mathfrak{m}}'_t & \text{if } s \in S_\Box \\ \sum_{t \in \Delta(s)} \Delta(s, t) \cdot \sum_{\mathfrak{m}' \in \mathfrak{M}} \pi_{\mathsf{u}}(\mathfrak{m}, t)(\mathfrak{m}') \cdot \vec{\mathfrak{m}}'_t & \text{if } s \in S_\bigcirc. \end{cases}$$

Therefore, by the ε-consistency of the memory mapping f_π, we have

$$\mathbb{E}_{\mathcal{D},s_o}[V_{N+1}|\lambda'] \geq \vec{\mathfrak{m}}_s - \vec{r}(s) - \varepsilon. \tag{6.6}$$

Further, evaluating expectations over paths in W_N yields

$$\mathbb{E}_{\mathcal{D},s_o}[\mathsf{rew}^{N+1}(\vec{r})] - \mathbb{E}_{\mathcal{D},s_o}[\mathsf{rew}^N(\vec{r})] = \textstyle\sum_{\lambda' \in W_N} \vec{r}(s) \cdot \mathbb{P}_{\mathcal{D},s_o}(\lambda'), \tag{6.7}$$

$$\mathbb{E}_{\mathcal{D},s_o}[V_N] = \textstyle\sum_{\lambda' \in W_N} \mathbb{P}_{\mathcal{D},s_o}(\lambda') \cdot \vec{\mathfrak{m}}_s. \tag{6.8}$$

We can now establish (6.5) as follows:

$$\begin{aligned}
\mathbb{E}_{\mathcal{D},s_o}[V_{N+1}] &= \textstyle\sum_{\lambda' \in W_N} \mathbb{E}_{\mathcal{D},s_o}[V_{N+1}|\lambda'] \cdot \mathbb{P}_{\mathcal{D},s_o}(\lambda') && \text{(law of total probability)} \\
&\geq \textstyle\sum_{\lambda' \in W_N} (\vec{\mathfrak{m}}_s - \vec{r}(s) - \varepsilon) \cdot \mathbb{P}_{\mathcal{D},s_o}(\lambda') && \text{(by Equation (6.6))} \\
&= \mathbb{E}_{\mathcal{D},s_o}[V_N] - (\mathbb{E}_{\mathcal{D},s_o}[\mathsf{rew}^{N+1}(\vec{r})] - \mathbb{E}_{\mathcal{D},s_o}[\mathsf{rew}^{N}(\vec{r})]) - \varepsilon \\
& && \text{(by Equations (6.7) and (6.8))} \\
&\geq \vec{\mathfrak{m}}_{p_o} - \mathbb{E}_{\mathcal{D},s_o}[\mathsf{rew}^{N+1}(\vec{r})] - (N+1) \cdot \varepsilon. && \text{(induction hypothesis)}
\end{aligned}$$

This concludes the proof by induction. □

Example[re] 6.2 (Memory Mapping)**.** *We explain the concept of memory mappings on the fixpoint shown in Figure 6.3. Let the memory of π be $\mathfrak{M} = \{\mathfrak{m}_0, \mathfrak{m}_1\} \times S$, that is, memory is specific to the states, for convenience of presentation. The memory mapping f_π is shown by assigning the extreme points of the sets in the fixpoint to memory elements. For example, at t_0, $f_\pi(\mathfrak{m}_0, t_0) = (-\frac{3}{4}, 0)$ and $f_\pi(\mathfrak{m}_1, t_0) = (0, -\frac{3}{4})$. Note that in Figure 6.3 some memory elements are shown in parentheses, and we explain in Example 6.3 that we may not need all memory elements for every state.*

6.3.2 SU Strategy Construction

We now show how to construct Player 1 strategies given $X \in \mathcal{C}_M$, rewards $\vec{r}$, and $\varepsilon \geq 0$, and we formalise in Proposition 6.27 the sufficient conditions for achieving $\mathsf{EE}(\vec{r} + \varepsilon)$. For any point $\vec{p} \in X_s$, $s \in S$, there is some $\vec{q} \geq \vec{p}$ that can be obtained by a convex combination of extreme points $\mathsf{C}(X_s)$, and so the strategy we construct uses $\mathsf{C}(X_s)$ as memory, randomising to attain the convex combination $\vec{q}$. Denote by $T_X \subseteq S$ the set of states $s \in S$ for which $[F_{M,\vec{r}}(X)]_s \neq \emptyset$.

Definition 6.2. *Given $X \in \mathcal{C}_M$, $\varepsilon \geq 0$, $\mathcal{G} = \langle S, (S_\Diamond, S_\Box, S_\bigcirc), s_{\mathsf{init}}, \mathcal{A}, \chi, \Delta \rangle$, and $\vec{r} : S \to \mathbb{R}^n$, define $\pi(X, \vec{r}, \varepsilon) = \langle \mathfrak{M}, \pi_{\mathsf{c}}, \pi_{\mathsf{u}}, \pi_{\mathsf{d}} \rangle$, where*

- *the memory is $\mathfrak{M} \stackrel{\text{def}}{=} \bigcup_{s \in T_X} \{(s, \vec{p}) \,|\, \vec{p} \in \mathsf{C}(X_s)\}$;*
- *the initial distribution π_{d} is defined by $\pi_{\mathsf{d}}(s) \stackrel{\text{def}}{=} (s, \vec{q}_0^{\,s})$ for any $s \in T_X$ and some arbitrary $\vec{q}_0^{\,s} \in \mathsf{C}(X_s)$; and*
- *the memory update π_{u} and next move function π_{c} are defined as follows: at state s with memory $(s, \vec{p})$, for all $t \in \Delta(s)$, pick n vectors $\vec{q}_i^{\,t} \in \mathsf{C}(X_t^k)$ for*

$1 \leq i \leq n$ *and distributions* $\beta^t \in \mathsf{D}(\{1,\dots,n\})$*, such that*

$$\begin{aligned}
&\textit{for } s \in S_\Diamond\textit{:} && \exists \alpha \in \mathsf{D}(\Delta(s) \cap T_X)\,.\, \textstyle\sum_{t\in\Delta(s)} \alpha(t) \cdot \sum_i \beta^t(i) \cdot \vec{q}_i^{\,t} \geq \vec{p} - \vec{r}(s) - \varepsilon,\\
&\textit{for } s \in S_\Box\textit{:} && \forall t \in \Delta(s)\,.\, \textstyle\sum_i \beta^t(i) \cdot \vec{q}_i^{\,t} \geq \vec{p} - \vec{r}(s) - \varepsilon,\\
&\textit{for } s \in S_\bigcirc\textit{:} && \textstyle\sum_{t\in\Delta(s)} \Delta(s,t) \cdot \sum_i \beta^t(i) \cdot \vec{q}_i^{\,t} \geq \vec{p} - \vec{r}(s) - \varepsilon,
\end{aligned}$$

and let, for all $t \in \Delta(s) \cap T_X$*,*

$$\begin{aligned}
\pi_u((s,\vec{p}),t)(t,\vec{q}_i^{\,t}) &\stackrel{\text{def}}{=} \beta^t(i) && \textit{for all } 1 \leq i \leq n,\\
\pi_c(s,(s,\vec{p}))(t) &\stackrel{\text{def}}{=} \alpha(t) && \textit{if } s \in S_\Diamond.
\end{aligned}$$

We now show that the strategy $\pi(X,\vec{r},\varepsilon)$ is well defined if $F_{M,\vec{r}}(X) + \varepsilon \sqsubseteq X$ and $[F_{M,\vec{r}}(X)]_{s_{\mathsf{init}}} \neq \emptyset$, and that it achieves $\mathsf{EE}(\vec{r}+\varepsilon)$, by defining an ε-consistent memory mapping for M from the sets $X \in \mathcal{C}_M$.

Proposition 6.27. *Let* $X \in \mathcal{C}_M$ *and* $\varepsilon \geq 0$*. If* $F_{M,\vec{r}}(X)+\varepsilon \sqsubseteq X$ *and* $[F_{M,\vec{r}}(X)]_{s_{\mathsf{init}}} \neq \emptyset$*, then* $\pi(X,\vec{r},\varepsilon)$ *achieves* $\mathsf{EE}(\vec{r}+\varepsilon)$*.*

Proof. Fix a game $\mathcal{G} = \langle S, (S_\Diamond, S_\Box, S_\bigcirc), s_{\mathsf{init}}, \mathcal{A}, \chi, \Delta\rangle$, let $X \in \mathcal{C}_M$, and let $\varepsilon \geq 0$. such that $F_{M,\vec{r}}(X) + \varepsilon \sqsubseteq X$ and $[F_{M,\vec{r}}(X)]_{s_{\mathsf{init}}} \neq \emptyset$. We first show that the strategy $\pi(X,\vec{r},\varepsilon)$ is well-defined. Recall that $T_X \subseteq S$ is the set of states $s \in S$ for which $[F_{M,\vec{r}}(X)]_s \neq \emptyset$, and so $s_{\mathsf{init}} \in T_X$, and, if $s \in T_X \cap (S_\Box \cup S_\bigcirc)$, then, for every $t \in \Delta(s)$, $[F_{M,\vec{r}}(X)]_t + \varepsilon \sqsubseteq X_t \neq \emptyset$, and hence $t \in T_X$.

For any $s \in T_X$, depending on the type of s (that is, Player 1, Player 2, or move), we define an auxiliary set Z_s without the cut-off by Box_M. We then show that we can find the required distributions α and β^t, and the extreme points for every point in Z_s, and prove that for all extreme points $\vec{p}$ of X_s we have $\vec{p} - \varepsilon$ in Z_s for $k \geq 0$, allowing us to show well-definedness of the strategy. Take $s \in T_X$.

- **Case** $s \in S_\Diamond$**.** Let $Z_s \stackrel{\text{def}}{=} \vec{r}(s) + \mathsf{conv}(\bigcup_{t\in\Delta(s)\cap T_X} X_t)$. Take any $\vec{p}' \in Z_s$. There are distributions $\alpha \in \mathsf{D}(\Delta(s) \cap T_X)$, $\beta^t \in \mathsf{D}(\{1,\dots,n\})$, and points $\vec{q}_i^{\,t} \in \mathsf{C}(X_t)$ for $t \in \Delta(s) \cap T_X$, such that $\sum_t \alpha(t) \cdot \sum_i \beta^t(i) \cdot \vec{q}_i^{\,t} \geq \vec{p}' - \vec{r}(s)$.

- **Case** $s \in S_\Box$**.** Let $Z_s \stackrel{\text{def}}{=} \mathsf{dwc}(\vec{r}(s) + \bigcap_{t\in\Delta(s)} X_t)$. Take any $\vec{p}' \in Z_s$. For any $t \in \Delta(s)$, there are distributions $\beta^t \in \mathsf{D}(\{1,\dots,n\})$, and points $\vec{q}_i^{\,t} \in \mathsf{C}(X_t)$ such that $\sum_i \beta_i^t \cdot \vec{q}_i^{\,t} \geq \vec{p}' - \vec{r}(s)$.

- **Case** $s \in S_\bigcirc$**.** Let $Z_s \stackrel{\text{def}}{=} \vec{r}(s) + \sum_{t\in\Delta(s)} \Delta(s,t) \times X_t$. Take any $\vec{p} \in Z_s$. Due to the Minkowski sum, there are distributions $\beta^t \in \mathsf{D}(\{1,\dots,n\})$, and points $\vec{q}_i^{\,t} \in \mathsf{C}(X_t)$ such that $\sum_{t\in\Delta(s)} \Delta(s,t) \cdot \sum_i \beta_i^t \cdot \vec{q}_i^{\,t} \geq \vec{p}' - \vec{r}(s)$.

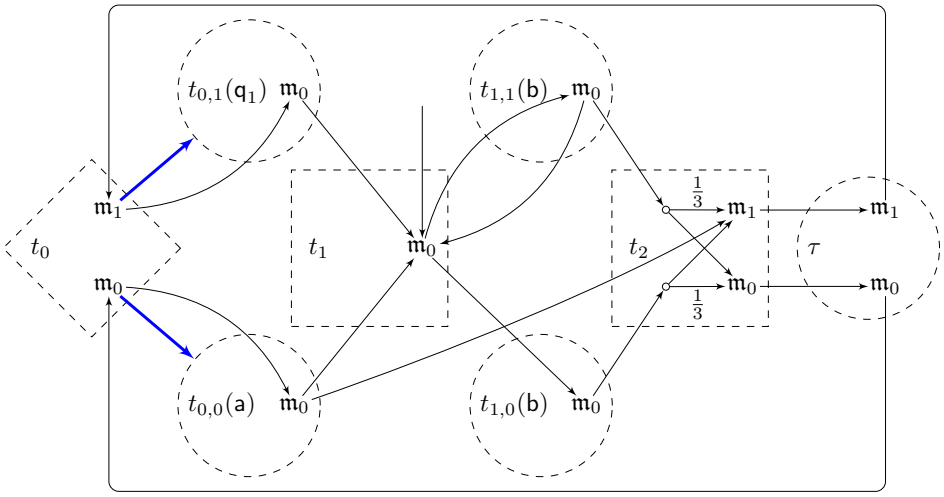


Figure 6.4: Strategy constructed from the fixpoint $\mathsf{fix}(F_{M,\vec{r},\mathcal{G}^2_{\mathsf{re}}})$ in Figure 6.3.

Note that, if two sets satisfy $A \sqsubseteq B$, they also satisfy $A - \varepsilon \sqsubseteq B - \varepsilon$. We have $F_{M,\vec{r}}(X) + \varepsilon \sqsubseteq X$, and so $\mathsf{dwc}(Z_s) \cap \mathsf{Box}_M = [F_{M,\vec{r}}(X)]_s \sqsubseteq X_s - \vec{\varepsilon}$, for all $s \in T_X$. Then, for any point $\vec{p} \in \mathsf{C}(X_s)$, it holds that $\vec{p} - \varepsilon \in \mathsf{dwc}(Z_s) \cap \mathsf{Box}_M$. We now let f_π be the memory mapping for $\pi(X, \vec{r}, \varepsilon)$, defined by $f_\pi(s, \vec{p}) \stackrel{\text{def}}{=} \vec{p}$ for all $s \in S$ and $\vec{p} \in \mathsf{C}(X_s)$. It is ε-consistent for $\vec{q}_0^s \geq -M$ by definition. We conclude that $\pi(X, \vec{r}, \varepsilon)$ achieves $\mathsf{EE}(\vec{r} + \varepsilon)$ by Lemma 6.26. □

Example$^{\mathsf{re}}$ 6.3 (Strategy Construction)**.** *We show how to construct a winning strategy $\pi^2 = \pi(X, \vec{r}, 0)$ from the fixpoint $X^* = \mathsf{fix}(F_{M,\vec{r},\mathcal{G}^2_{\mathsf{re}}})$ (shown in Figure 6.3), for the game $\mathcal{G}^2_{\mathsf{re}}$ of our running example in Section 3.6. Since we operate with the true fixpoint we can set $\varepsilon = 0$, as $F_{M,\vec{r}}(X^*) \sqsubseteq X^*$ for the preconditions of Proposition 6.27, and we construct the strategy according to the memory mapping shown in Figure 6.3. Since the initial state of $\mathcal{G}^2_{\mathsf{re}}$ is t_1, we only construct the strategy constructed specific to this state, and therefore only use memory elements that are reachable from the initial state t_1. In Figure 6.3 the unused memory elements are shown in parentheses.*

We show the constructed strategy π^2 in Figure 6.4. The strategy is initialised to $\mathfrak{m}_0$ at t_1, corresponding to the point $(0, -\frac{3}{4})$ in the fixpoint $\mathsf{fix}(F_{M,\vec{r}})(t_1)$. For illustration, we explain the memory update from $\mathfrak{m}_0$ at $t_{1,0}$. The memory update randomises between the memory elements of t_2, assigning probability $\frac{1}{3}$ to $\mathfrak{m}_0$ and probability $\frac{2}{3}$ to $\mathfrak{m}_1$, which is due to the memory mapping: the strategy needs to maintain $(-\frac{1}{4}, -\frac{1}{2})$, because the point associated with $\mathfrak{m}_0$ at $t_{1,0}$ is $(0, -\frac{3}{4})$, and the shift that is incurred at $t_{1,0}$ from the b action is $(\frac{1}{4}, -\frac{1}{4})$ (see Example 6.1). Note

that the strategy plays the EE *objective, and so does not cut off the positive values, as is done to compute X^*. To find the distribution between the memory elements at t_2, we solve*

$$\alpha \cdot \overbrace{(-\tfrac{3}{4}, 0)}^{\mathfrak{m}_0} + (1-\alpha) \cdot \overbrace{(0, -\tfrac{3}{4})}^{\mathfrak{m}_1} = (-\tfrac{1}{4}, -\tfrac{1}{2}),$$

and get $\alpha = \frac{1}{3}$, corresponding to the desired distribution of memory elements at t_2.

6.3.3 Finite DU Strategies

We construct SU strategies in Section 6.3.2. Since we use Proposition 6.18 (ii) in order to show completeness of our method, we show in this section that if Player 1 achieves $\mathsf{Pmp}(\vec{r})$ with an arbitrary strategy, then it also has a finite ε-optimal DU strategy. The intuition is to restart the strategy after intervals that are long enough so that the probability of paths violating the required mean-payoff threshold approaches zero. We first show the following technical result.

Lemma 6.28. *Let $(X_n)_{n\geq 0}$ be a sequence of real-valued random variables such that $\mathbb{P}(\underline{\lim}_{n\to\infty} X_n \geq v) = 1$. For every $\delta > 0$, it holds that $\mathbb{P}(X_n < v - \delta) \to 0$ as $n \to \infty$.*

Proof. Assume that $\mathbb{P}(\underline{\lim}_{n\to\infty} X_n \geq v) = 1$. Fix $\delta > 0$. Let $A_n \stackrel{\text{def}}{=} \bigcup_{m\geq n}\{e \mid X_m(e) < v - \delta\}$. Since A_n is a non-increasing sequence of events as $n \to \infty$, it holds that $\mathbb{P}(A_n) \to \mathbb{P}(\bigcap_{n\geq 0} A_n)$, which is zero by hypothesis of the lemma. Hence $P(X_n < v-\delta)$ also tends to zero as $n \to \infty$, since $P(X_n < v - \delta) \leq P(A_n)$. □

Theorem 6.29. *If Player 1 achieves $\mathsf{Pmp}(\vec{r})$, then, for every $\varepsilon > 0$, Player 1 has a finite DU strategy to achieve $\mathsf{Pmp}(\vec{r} + \varepsilon)$.*

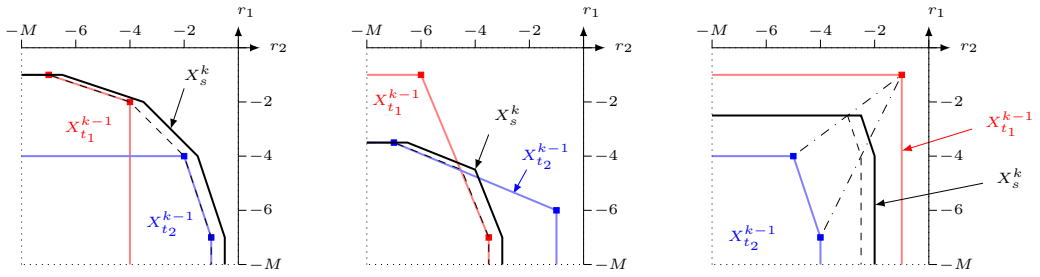
Proof. Let $\mathcal{G}$ be a game, let π be a Player 1 strategy achieving $\mathsf{Pmp}(\vec{r})$, and let $\mathcal{M} = \mathcal{G}^\pi$. Denote by $S_\mathcal{M}$ and $S_\mathcal{G}$ the respective states spaces of $\mathcal{M}$ and $\mathcal{G}$. Without loss of generality, we assume that the memory of π is the set $\Omega^{\mathsf{fin}}_\mathcal{G}$ of paths in $\mathcal{G}$, and so $\mathcal{M}$ is an infinite tree where each state is identified uniquely by a path $\lambda \in \Omega^{\mathsf{fin}}_\mathcal{G}$. Consider the set $S_{\mathcal{M},\mathcal{G}} \stackrel{\text{def}}{=} \{\mathsf{last}(\lambda) \in S_\mathcal{G} \mid \lambda \in S_\mathcal{M}\}$ of states of the game that appear in some state of $\mathcal{M}$. For every $\lambda \in S_\mathcal{M}$, $\mathbb{P}^\sigma_{\mathcal{M},\lambda}(\mathsf{mp}(\vec{r}) \geq 0) = 1$ holds for all Player 2 strategies σ. Consider, for each state $s \in S_{\mathcal{M},\mathcal{G}}$, a path $\lambda_s \in S_\mathcal{M}$ with $\mathsf{last}(\lambda_s) = s$, which uniquely identifies a state in $\mathcal{M}$ (note that given s, λ_s is not unique, but it suffices to pick an arbitrary one). Then, for every Player 2 strategy σ, it holds that $\mathbb{P}^\sigma_{\mathcal{M},\lambda_s}(\underline{\lim}_{N\to\infty} \frac{1}{N+1}\mathsf{rew}^N(\vec{r}) \geq 0) = 1$, and hence, by Lemma 6.28, the quantity

$p_{s,h,\sigma} \stackrel{\text{def}}{=} \mathbb{P}^{\sigma}_{\mathcal{M},\lambda_s}(\frac{1}{h+1}\mathsf{rew}^h(\vec{r}) \leq -\varepsilon/2)$ tends to 0. We define $p_{h,\sigma} \stackrel{\text{def}}{=} \max_s p_{s,h,\sigma}$, and let p_h be the maximum $p_{s,h,\sigma}$ over all MD Player 2 strategies σ. As the maxima are taken over finite sets, we have that $p_h \to 0$ as $h \to \infty$.

Now construct the finite DU Player 1 strategy π_h that plays as follows: starting from $s \in S_{\mathcal{M},\mathcal{G}}$, it initialises its memory to λ_s and plays π for h steps; then, from whatever state $t \in S_{\mathcal{M},\mathcal{G}}$ it arrived at, it resets its memory to λ_t and plays π for a further h steps, and so on. Fix any MD strategy of Player 2, and a BSCC $\mathcal{B}$ of the induced DTMC $\mathcal{D} = \mathcal{G}^{\pi_h,\sigma}$. Given a state $s \in S_{\mathcal{M},\mathcal{G}}$, let $\tilde{s} = (s, \lambda_s, \mathfrak{n})$ be the corresponding state of $\mathcal{D}$ (where $\mathfrak{n}$ is the only memory element of σ.) Since π_h plays unique memory elements (by unrolling the history) except when resetting in states of $S_{\mathcal{M},\mathcal{G}}$, $\mathcal{B}$ must contain at least one state $\tilde{s}_0$ with $s_0 \in S_{\mathcal{M},\mathcal{G}}$. Note that $\mathcal{B}$ is finite. By Remark 3.6, we then have $\mathbb{P}_{\mathcal{D}}(\mathsf{mp}(\vec{r}) = \mathsf{mp}(\vec{r})(\mathcal{B})) = 1$, so it suffices to find a lower-bound for $\mathsf{mp}(\vec{r})(\mathcal{B})$, which is equivalent to $\lim_{N\to\infty} \frac{1}{N+1}\mathbb{E}_{\mathcal{D},\tilde{s}_0}[\mathsf{rew}^N(\vec{r})]$. We have constructed π_h so that every h steps a state in $S_{\mathcal{M},\mathcal{G}}$ is encountered, and hence it holds, for every $k \geq 0$, that $\mathbb{E}_{\mathcal{D},\tilde{s}_0}[\mathsf{rew}^{kh}(\vec{r})] \geq k \cdot \min_{s\in S_{\mathcal{M},\mathcal{G}}} \mathbb{E}_{\mathcal{D},\tilde{s}}[\mathsf{rew}^h(\vec{r})]$. From a state $s \in S_{\mathcal{M},\mathcal{G}}$, with probability less than $p_{h,\sigma}$, the reward accumulated is at least $-h\rho^*$, where $\rho^* = \max_{s\in S_{\mathcal{G},i}} |r_i(s)|$. Further, with probability greater than $1 - p_{h,\sigma}$ the reward accumulated is at least $-h\varepsilon$. Therefore, for every state $s \in S_{\mathcal{M},\mathcal{G}}$, $\mathbb{E}_{\mathcal{D},\tilde{s}}[\mathsf{rew}^h(\vec{r})] \geq -p_{h,\sigma}\rho^* - (1 - p_{h,\sigma})h\varepsilon \geq -p_h\rho^* - h\varepsilon$. Hence, $\mathbb{E}_{\mathcal{D},\tilde{s}}[\mathsf{rew}^{kh}(\vec{r})] \geq -kh(p_h\rho^* + \varepsilon)$. Dividing by $kh + 1$ and letting k go towards infinity, we get that $\mathbb{E}_{\mathcal{D},\tilde{s}_0}[\mathsf{mp}(\vec{r})] = \lim_k \frac{1}{kh+1}\mathbb{E}_{\mathcal{D},\tilde{s}_0}[\mathsf{rew}^{kh}(\vec{r})] \geq -p_h\rho^* - \varepsilon$. We therefore have, for every BSCC $\mathcal{B}$ of $\mathcal{D}$, that $\mathsf{mp}(\vec{r})(\mathcal{B}) \geq -p_h\rho^* - \varepsilon$, and hence, by Remark 5.3, Player 1 achieves $\mathsf{Pmp}(\vec{r} + p_h\rho^* + \varepsilon)$ against all MD Player 2 strategies. Then, by Theorem 6.2, Player 1 achieves $\mathsf{Pmp}(\vec{r} + p_h\rho^* + \varepsilon/2)$ against all Player 2 strategies. Since $p_h \to 0$, we can find h large enough so that $p_h\rho^* \leq \varepsilon/2$, and hence have $\mathsf{Pmp}(\vec{r} + \varepsilon)$ against every σ. □

6.4 Strategy Synthesis Algorithm

In this section we develop our algorithm to construct winning strategies for Pmp CQs. In the proof of Proposition 6.27 we constructed winning strategies for EE objectives if we are given $X \in \mathcal{C}_M$ such that $F_{M,\vec{r}}(X) + \varepsilon \sqsubseteq X$ and $[F_{M,\vec{r}}(X)]_{s_{\mathsf{init}}} \neq \emptyset$. Our algorithm computes an approximation X to $\mathsf{fix}(F_{M,\vec{r}})$ that satisfies the required conditions, and uses the transformations in Proposition 6.18 to obtain a strategy for Pmp.



(a) Player 1 state s. (b) Player 2 state s. (c) Move $s = (a, \mu)$; μ uniform.

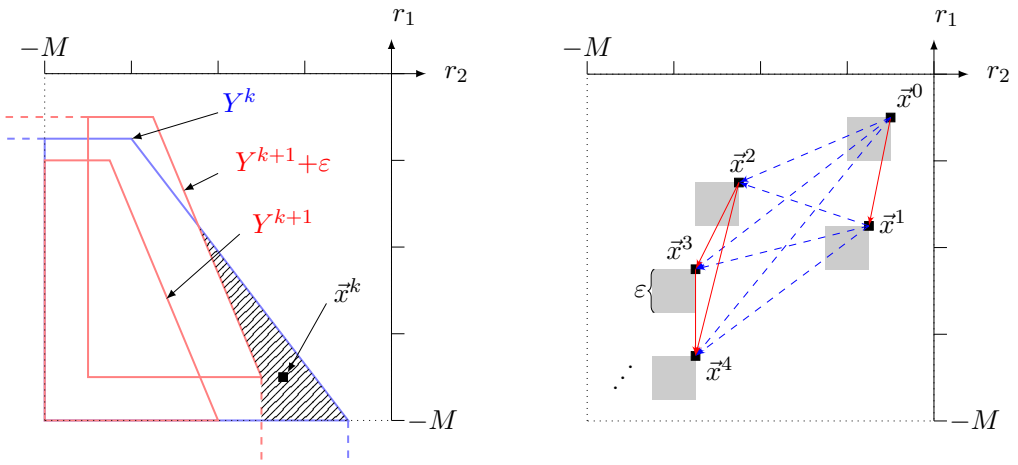
Figure 6.5: One step of the fixpoint computation for expected truncated energy shortfalls for a state s with successors t_1, t_2, and rewards $r_1(s) = 0.5$ and $r_2(s) = 0$.

6.4.1 Fixpoint Computation

From the Kleene fixpoint theorem (Lemma 3.10), we can obtain the least fixpoint $\mathsf{fix}(F_{M,\vec{r}})$ as the limit of the sequence $(F^k_{M,\vec{r}}(\bot_M))_{k\geq 0}$ as $k \to \infty$. For notational convenience, we let $X^k \stackrel{\text{def}}{=} F^k_{M,\vec{r}}(\bot_M)$. Since, for finite k, X^k has a finite number of extreme points, which we use as strategy memory in $\pi(X, \vec{r}, \varepsilon)$, we obtain finite SU strategies. Note that our strategy construction also works for $\mathsf{fix}(F_{M,\vec{r}})$, where we can take $\varepsilon = 0$, but it is not currently known if $\mathsf{fix}(F_{M,\vec{r}})$ is finitely representable or computable in general.

Example 6.4 (Fixpoint Computation)**.** *We illustrate a single step of our fixpoint computation in Figure 6.5. Consider a state s with two successors t_1 and t_2, and the three cases of $s \in S_\Diamond$, $s \in S_\Box$ and $s \in S_\bigcirc$. We let the rewards be $r_1(s) = 0.5$ and $r_2(s) = 0$, and let the box size $M = 8$. The figure shows, for the three state types, the sets X^k_s at iteration k, which are computed from $X^{k-1}_{t_1}$ and $X^{k-1}_{t_2}$, the sets corresponding to the successors at iteration $k-1$.*

Bounding the Number of Steps. To bound the number of steps k necessary for $X^{k+1} + \varepsilon \sqsubseteq X^k$, we use Lemma 6.30 (related to Ramsey's theorem) to bound the length of paths in a graph. A *graph* $G = (V, E)$ consists of a finite set V of *nodes* and a set $E \subseteq V \times V$ of *edges*. A graph is *linearly-ordered complete* if, for some strict linear order $\succ$ on V, $(v, w) \in E$ if and only if $v \succ w$. An *n-colouring* of a graph (V, E) is a function $E \to \{1, \ldots, n\}$, assigning one of n colours to each edge. A *monochromatic directed path of length N* is a sequence of nodes $v_1, \ldots, v_N$ such that $(v_i, v_{i+1}) \in E$ for all $1 \leq i < N$, and such that each node v_i has the same colour.



(a) The hatched region is the set $Y^k \cap (\mathsf{Box}_M \backslash \mathsf{dwc}(Y^{k+1} + \varepsilon))$, where $\vec{x}^k$ has to be.

(b) The red (solid) and blue (dashed) arrows represent distance greater than ε in dimensions r_1 and r_2, respectively.

Figure 6.6: Illustrations for Lemma 6.31 for two dimensions r_1 and r_2.

Lemma 6.30 (Theorem 4.5.2 of [116])**.** *Let $G = (V, E)$ be a linearly-ordered complete graph over m nodes, with an n-colouring of its edges. Then G contains a monochromatic directed path of length $\lfloor \sqrt{m/n - 2} \rfloor - 1$.*

We first consider a single state in Lemma 6.31, and use an inductive argument on the number of states to find the bound for all states in Proposition 6.33.

Lemma 6.31. *Let $(Z^k)_{k\geq 0}$ be a sequence over $\mathcal{P}_{c,M}(\mathsf{Box}_M)$ such that $Z^k \sqsubseteq Z^{k+1}$ for every $k \geq 0$. For every $I \subseteq \mathbb{N}$ such that $|I| \geq k^* \stackrel{\text{def}}{=} n \cdot ((\lceil \frac{M}{\varepsilon} \rceil + 1)^2 + 2)$, there exists $k \in I$ such that $Z^{k+1} + \varepsilon \sqsubseteq Z^k$.*

Proof. Fix a sequence $(Z_k)_{k\geq 0}$ that is non-decreasing for $\sqsubseteq$, and fix $I \subseteq \mathbb{N}$ such that $|I| \geq k^*$. Assume towards a contradiction that for every $k \in I$, $Z^{k+1} + \varepsilon \not\sqsubseteq Z^k$. Note first that if two sets satisfy $Z \not\sqsubseteq X$, then there exists $\vec{x} \in X \setminus \mathsf{dwc}(Z)$. Hence the hypothesis $Z^{k+1} + \varepsilon \not\sqsubseteq Z^k$ for every $k \in I$ implies the existence of a sequence $(\vec{x}^k)_{k\in I} \in Z^k \setminus \mathsf{dwc}(Z^{k+1} + \varepsilon)$ of points, shown in Figure 6.6 (a). Note that, for all $j < k$, there exists a coordinate $c(j,k)$ for which $x^j_{c(j,k)} - x^k_{c(j,k)} > \varepsilon$. Assume otherwise, that is, $\vec{x}^j - \varepsilon \leq \vec{x}^k$ for $j < k$. Then $\vec{x}^j - \varepsilon \in \mathsf{dwc}(Z^k)$, and, since $Z^k \sqsubseteq Z^{j+1}$, we deduce $\vec{x}^j \in \mathsf{dwc}(Z^{j+1} + \varepsilon)$, contradicting the definition of the sequence $(\vec{x}^k)_{k\leq m}$.

Now consider the linearly-ordered complete graph over nodes I, and with edges (j,k) for $j < k$ and $j, k \in I$. Endow the edges of this graph with the n-colouring c given above, that is, there is one colour per dimension of the M-polytope, see

Figure 6.6 (b). By Lemma 6.30, there exists a monochromatic path $j_1 \to j_2 \to \cdots \to j_l$ of length $l = \lfloor\sqrt{|I|/n-2}\rfloor - 1 \geq \lceil\frac{M}{\varepsilon}\rceil$, and thus by denoting c the colour of this path it holds that $x_c^{j_1} > x_c^{j_2} + \varepsilon > \ldots > x_c^{j_l} + l\varepsilon \geq -M + \frac{M}{\varepsilon}\varepsilon \geq 0$, a contradiction. □

Lemma 6.32. *Let U a be a finite set, let ϕ be a predicate over $U \times \mathbb{N}$, and let K be a positive integer. The implication "$\Phi_1 \Rightarrow \Phi_2$" holds, where*

Φ_1 *is "for every $s \in U$ and every $I \subseteq \mathbb{N}$ such that $|I| \geq K$, there exists $i \in I$, such that $\phi(s,i)$ holds;" and*

Φ_2 *is "for every $I \subseteq \mathbb{N}$ such that $|I| \geq K^{|U|}$, there exists $i \in I$ such that, for every $s \in U$, $\phi(s,i)$ holds."*

Proof. We show the result by induction on the cardinality of U. If U is empty the result is true. Now assume that the implication "$\Phi_1 \Rightarrow \Phi_2$" holds for sets U' of cardinality c, and let $U = U' \cup \{t\}$ be of cardinality $c+1$. Let ϕ be a predicate over $U \times \mathbb{N}$ and let K be a positive integer, such that Φ_1 is satisfied for U. Let $I \subseteq \mathbb{N}$ such that $|I| \geq K^{|U|}$. We want to find an index i such that $\phi(s,i)$ holds for all $s \in U$. We partition I into K parts $I_1, \ldots, I_K$, each containing at least $K^{|U|-1}$ elements. Since Φ_1 is satisfied for U, it is also satisfied for $U \setminus \{t\}$, and so, by the induction hypothesis, for every I_k there is an index $i_k \in I_k$ such that, for every $s \in U \setminus \{t\}$, $\phi(s,i_k)$ holds. The set $\{i_1, \ldots, i_K\}$ contains K elements and hence we can apply Φ_1 (which holds for U by assumption), and extract one i such that also $\phi(t,i)$ is true. Hence, i is such that for every $s \in U$, $\phi(s,i)$ is true, concluding the induction step. □

Proposition 6.33. *Let M, $\varepsilon > 0$, and let $(X^k)_{k\geq 0}$ be a sequence over $\mathcal{C}_M$ such that $X^k \sqsubseteq X^{k+1}$ for every $k \geq 0$. There exists $k \leq k^{**} \stackrel{\text{def}}{=} \left[n((\lceil\frac{M}{\varepsilon}\rceil + 1)^2 + 2)\right]^{|S|}$, such that $X^{k+1} + \varepsilon \sqsubseteq X^k$.*

Proof. Fix M and $\varepsilon > 0$. Let $\mathcal{G}$ be a game with state space S. Let $(X^k)_{k\geq 0}$ be a sequence over $\mathcal{C}_M$ that is non-decreasing for $\sqsubseteq$. We apply Lemma 6.32 with $U = S$, $K = k^*$, and with the predicate $X_s^{k+1} + \varepsilon \sqsubseteq X_s^k$ for ϕ, noting that Φ_1 is satisfied by Lemma 6.31, and that Φ_2 is the statement we set out to prove. □

6.4.2 Algorithm

We now summarise our synthesis algorithm for Pmp CQs. We first invoke our co-NP decision procedure from Section 6.1. If the specification is achievable, we fix $\vec{r}' = \vec{r} + \frac{\varepsilon}{2}$,

and so by Proposition 6.23, there is an M such that for all $k \geq 0$, $[F^k_{M,\vec{r}'}(\bot_M)]_{s_{\mathsf{init}}} \neq \emptyset$. For a fixed M, we compute the fixpoint approximation for $X^k = F^k_{M,\vec{r}'}(\bot_M)$, until either $F_{M,\vec{r}'}(X^k) + \varepsilon \sqsubseteq X^k$ or $[F_{M,\vec{r}'}(X^k)]_{s_{\mathsf{init}}} = \emptyset$, where the maximum required value of k is bounded by Proposition 6.33. However, since we do not know the value of M a-priori, we iteratively increase M, for example, starting M at $M_{\min} = 2$, and increasing M quadratically every time $[F_{M,\vec{r}'}(X^k)]_{s_{\mathsf{init}}} = \emptyset$. We state in Theorem 6.34 the correctness of our algorithm, and give the pseudocode in Section 7.2.3.

Theorem 6.34. *The above algorithm terminates, returning a finite ε-optimal strategy for $\mathsf{Pmp}(\vec{r})$ if it is achievable, and indicating if it is unachievable.*

Proof. The case when $\mathsf{Pmp}(\vec{r})$ is not achievable is covered by Corollary 6.6. Suppose $\mathsf{Pmp}(\vec{r})$ is achievable. By Theorem 6.29, $\mathsf{Pmp}(\vec{r} + \frac{\varepsilon}{8})$ is achievable by a finite DU strategy, and so, by Proposition 6.18 (ii), $\mathsf{EE}(\vec{r} + \frac{\varepsilon}{4})$ is achievable by a finite DU strategy. Applying Proposition 6.23 with $\vec{r}' \stackrel{\text{def}}{=} \vec{r} + \frac{\varepsilon}{4} + \varepsilon'$ and $\varepsilon' = \frac{\varepsilon}{4}$, there exists an M such that $[\mathsf{fix}(F_{M,\vec{r}'})]_{s_{\mathsf{init}}}$ is nonempty. As $[F^k_{M,\vec{r}'}(\bot_M)]_{s_{\mathsf{init}}} = X^k_{s_{\mathsf{init}}} \sqsubseteq [\mathsf{fix}(F_{M,\vec{r}'})]_{s_{\mathsf{init}}}$, we have $X^k_{s_{\mathsf{init}}} \neq \emptyset$. Further, due to the bound M, after a finite number of iterations, k, we have $X^k + \frac{\varepsilon}{2} \sqsubseteq X^{k-1}$, by Proposition 6.33. Then, by Proposition 6.27, the finite SU strategy $\pi(X^{k-1}, \vec{r} + \frac{\varepsilon}{2}, \frac{\varepsilon}{2})$, constructed for the reward $\vec{r}' = \vec{r} + \vec{\frac{\varepsilon}{2}}$, satisfies $\mathsf{EE}(\vec{r} + \vec{\varepsilon})$. By Proposition 6.18 (i), this strategy also satisfies $\mathsf{Pmp}(\vec{r})(-\vec{\varepsilon})$. □

Value and Policy Iteration. Our algorithm uses a *value iteration* approach to first find the achievable targets for the EE objective, using the Belman operator $F_{M,\vec{r}}$, and then constructs a strategy from the resulting sets of shortfalls. An alternative approach is *policy iteration*, which involves starting with an arbitrary strategy π^k, computing the achievable targets under π^k, and then using an *improvement step* to find a strategy π^{k+1}, until a strategy is found achieving the desired target. While both value and policy iteration yield winning strategies when they exist, one is usually outperforming the other in practice. In our setting, the use of policy iteration requires to overcome two key difficulties. Firstly, strategies of $\mathsf{Player\ 1}$ in general require memory and randomisation, which means that the set of strategies to iterate over is unbounded, and evaluating the outcome of any particular strategy requires to reason over the induced PA. Secondly, obtaining an *improvement step* requires to understand how changing a given strategy affects the outcome of the game, which is exacerbated by the observation that setting the probability to pick a move to zero changes the topology of the induced PA, and hence can affect the end components. We therefore

adopted a value iteration approach, as we can approximate the achievable targets iteratively, and then construct strategies with the appropriate memory updates.

6.5 Summary

In this chapter we have presented our synthesis method for conjunctions of almost sure satisfaction of mean-payoff (Pmp) objectives. We first showed that the achievability problem for Pmp CQs is in co-NP if Player 1 can play arbitrary strategies. We introduced expected (truncated) energy objectives, which we used as a technical tool in our strategy construction, and gave a transformation between Pmp objectives and expected energy (EE) objectives. We then gave a Bellman operator, $F_{M,\vec{r}}$, to characterise the shortfalls of the expected truncated energy in a stochastic game, which we defined over a CPO of compact, closed subsets of a continuous space, reflecting the continuous nature introduced by stochasticity. Using this Bellman operator, we developed a finite fixpoint approximation for the shortfalls of the expected truncated energy, together with a relative stopping criterion, $F^{k+1}_{M,\vec{r}}(\bot_M) + \varepsilon \sqsubseteq F^{k}_{M,\vec{r}}(\bot_M)$. From the approximations to the fixpoint, we then constructed finite SU strategies that are ε-optimal for Pmp.

Our proofs for the Bellman operator $F_{M,\vec{r}}$ use finite DU strategies, since this allows us to induce finite PAs using Definition 3.5. However, the strategies we construct in Section 6.3.2 are (finite) SU strategies. By showing in Theorem 6.29 that we can ε-approximate arbitrary winning strategies using finite DU strategies, our algorithm, which we summarise in Section 6.4.2, is sound and complete for ε-optimal strategies.

Several directions for further investigation arise. In this chapter we focused on keeping the mean-payoffs above a threshold with probability one. In [71] it is shown that, if we know the states where Player 1 has an almost sure winning strategy (the *almost sure winning region*), we can construct strategies for an arbitrary probability threshold, that is, $\mathbb{P}(\mathsf{mp}(r) \geq v) \geq \theta$ for $\theta \in [0, 1]$. It is to be investigated whether the same is possible for conjunctions, that is, whether we can construct strategies for $\mathbb{P}(\bigwedge_i \mathsf{mp}(r_i) \geq v_i) \geq \theta$ from knowing the winning region for $\mathbb{P}(\bigwedge_i \mathsf{mp}(r_i) \geq v_i) = 1$. A further question is to construct strategies for *percentile* queries of the form $\bigwedge_i \mathbb{P}(\mathsf{mp}(r_i) \geq v_i) \geq \theta_i$, where each objective could have a different probability threshold. For MDPs, the synthesis problem for percentile queries with mean-payoffs is discussed in [34]. In the context of our assume-guarantee framework, percentile queries motivate the investigation of further rules, where the probability thresholds may change through composition.

Further, some questions of complexity classification remain open. For Pmp CQs, our algorithm iteratively finds a box size M that is sufficiently large to separate the states where the energy diverges from the states where the energy is bounded. We do not currently know how to pick M so that it is large enough a-priori, even in the cases where we know that such a value exists from deciding the achievability problem. Knowing M would also establish a complexity bound for the synthesis problem of finite ε-optimal strategies. The complexity of Emp synthesis remains an open problem; two key characteristics in this case are that the games are not determined (Proposition 1 of [40]), and that it is not sufficient to consider MD strategies for Player 2, see for example the (non-stopping) game in Figure 1 of [40]. To approach the complexity classification for the Emp achievability problem, the strictly simpler problem of target reachability in stopping games can be considered first [124].

Chapter 7

Tool Implementation

Contents

In this chapter we present PRISM-games 2.0, a tool for strategy synthesis from the multi-objective queries discussed in this thesis [86]. The tool implements our algorithms from Chapters 5 and 6 for strategy synthesis within the assume-guarantee framework of Chapter 4, and thus provides comprehensive support for compositional modelling and analysis of stochastic games.

PRISM-games 2.0 is based on the PRISM-games tool, which supports the verification of turn-based multi-player stochastic games with single-objective rPATL objectives [38], and includes non-compositional strategy synthesis [39]. We extend the native PRISM modelling language to support fully compositional game models. A composed game, which we call here the *top-level system*, may consist of several *subsystems* (the component games), and each subsystem comprises one or more *modules*. The assignment of actions to players can be done individually per subsystem, using a syntax inspired by input/output automata [46], marking outputs (controlled by Player 1) with an exclamation mark (!) and inputs (controlled by Player 2) with a question mark (?). We also extend the property specification language, so that each component game can be assigned its local goal specification. Further, the individual objectives of multi-objective queries can be named, so that it is clear which objectives are related across subsystems, as is required, for example, in the (Asym) rule. By visualising Pareto sets in the graphical user interface (GUI), the user can conveniently

explore which targets are achievable globally, and how to instantiate the local targets. Finally, we implement the (monolithic) synthesis algorithms for Pmp, Pratio, Emp, and ratioE objectives developed in this thesis, as well as for Erew objectives based on our work in [41, 43].

In Section 7.1 we describe the modelling and property specification language used in the tool, highlighting the new features for assume-guarantee strategy synthesis and multi-objective property specification. Then, in Section 7.2, we briefly discuss how the main synthesis algorithms are implemented. We also give details of the fixpoint computations of the $F_{M,\vec{r}}$ operator for expected energy defined in Section 6.2, and the $F_{\mathsf{rew},M}$ operator for expected total rewards defined in Section 3.5.3, and we explain the approximations that were introduced in order to improve performance. Finally, in Section 7.3, we give a walkthrough of the tool's functionality.

PRISM-games 2.0 is available at `http://www.prismmodelchecker.org/games/`, and is open source under the GPL licence. Earlier prototype implementations were presented and used in [10, 11, 43, 63]. We describe several case studies in Chapter 8, demonstrating the effectiveness and versatility of PRISM-games 2.0.

7.1 Modelling and Property Specification Language

PRISM-games 2.0 offers a flexible framework for compositional modelling of stochastic games, extending the native PRISM language, which is based on the reactive modules formalism of [1]. Below, we describe our modelling language extensions, and refer to Figure 7.1 as an example, which models our running example of Section 3.6.

Top-Level System. PRISM-games 2.0 supports the modelling of games by composing several smaller games. In a composed game $\mathcal{G} = \texttt{S1} \parallel \texttt{S2} \parallel \cdots$, we call $\mathcal{G}$ the *top-level system*, and each of the component games `S1`, `S2`, ... a *subsystem*. Subsystems `S1`, `S2`, ..., modelling normal form games, are composed to the top-level system using the game composition operator from Definition 4.3, which is written as

```
system
  "S1" || "S2" || ···
endsystem
```

PRISM-games 2.0 requires the top-level system to be specified first in the file, just after the keyword `smg`, which designates the model as a stochastic (multi-player) game. In Figure 7.1, for example, the preamble (in the box on the top left) defines the top-level system consisting of the subsystems `S1` and `S2`.

```
smg

system
   "S1" || "S2"
endsystem
```

```
system "S1"
   G1
endsystem
module G1

   s : [0..2] init 1;

   [d!] s=0 → (s'=1);
   [q1!] s=0 → (s'=1);

   [a?] s=1 → (s'=2);
   [b?] s=1 → 0.5 : (s'=1)
            + 0.5 : (s'=2);

   [] s=2 → (s'=0);
   [a?] s=2 → (s'=2);

endmodule
```

```
system "S2"
   G2
endsystem
module G2

   t : [0..2] init 1;

   [a!] t=0 → 0.5 : (t'=1)
            + 0.5 : (t'=2);
   [q1!] t=0 → (t'=1);

   [b?] t=1 → (t'=2);
   [b?] t=1 → 0.5 : (t'=1)
            + 0.5 : (t'=2);

   [] t=2 → (t'=0);

endmodule
```

```
rewards "r1"
   [a] true : 1;
endrewards

rewards "r2"
   [d] true : 1;
   [b] true : 1;
endrewards

rewards "r3"
   [b] true : 1;
endrewards

rewards "c"
   [a] true : 1;
   [b] true : 1;
endrewards
```

Figure 7.1: A PRISM-games 2.0 model of the composed game from the running example in Section 3.6, including the reward structures used to define the specifications, see Figure 7.2.

Subsystems. Each subsystem consists of one or more *modules*, and each module is itself a game. PRISM-games 2.0 composes modules using the *module composition*, which synchronises on shared actions, but does not distinguish between players. The module composition is the standard parallel composition in PRISM and PRISM-games, and is performed by first interpreting the modules as PAs, composing them using the PA composition [119], and then assigning the states of the composition to players. Thus, the module composition requires that Player 1 states only compose with Player 1 states, and symmetrically for Player 2 states.

Definition 7.1. *Given games* $\mathcal{G}^i = \langle S^i, (S^i_\Diamond, S^i_\Box, S^i_\bigcirc), s^i_{init}, \mathcal{A}^i, \chi^i, \Delta^i \rangle$, $i \in I$, *the* module composition *is* $\langle S, (S_\Diamond, S_\Box, S_\bigcirc), s_{init}, \mathcal{A}, \chi, \Delta \rangle$, *derived from the composed PA* $\langle S, (S_\Diamond \cup S_\Box, S_\bigcirc), s_{init}, \mathcal{A}, \chi, \Delta \rangle = \|_{i \in I} \langle S^i, (S^i_\Diamond \cup S^i_\Box, S^i_\bigcirc), s^i_{init}, \mathcal{A}^i, \chi^i, \Delta^i \rangle$, *such that* $\vec{s} \in S_\Diamond$ *if and only if* $s_i \in S^i_\Diamond$ *for some* i, *and* $\vec{s} \in S_\Box$ *if and only if* $s_i \in S^i_\Box$ *for some* i.

The module composition does not support assume-guarantee synthesis, which requires that no Player 1 states are put in parallel, see the discussion of compatibility and

strategy composition in Sections 4.1.3 and 4.1.4. Modules `G1`, `G2`, ..., are composed to a *subsystem* `S1` using the construct

```
system  "S1"
  G1 || G2 || ···
endsystem
```

Note that each subsystem must carry a name (here `S1`), which is used to define the top-level system. In our example in Figure 7.1, each component consists of only one module, that is, `S1` is made up of `G1`, and `S2` is made up of `G2`.

Modules. Modules are the basic building blocks in the modelling of games. The state space of a module is determined by a set of *variables*, each of which has a *domain*, and may (optionally) be initialised to a value in the domain using the `init` keyword. For example, in Figure 7.1, the variable `s` in the module `G1` is defined over the domain [0..2], corresponding to the set of integers from 0 to 2, and is initialised to 2 using the statement `init` 2. The transition and labelling functions of a module are specified using a set of *guarded commands*, which are of the form

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

and we explain in the following the components of a guarded command. Firstly, the label `a` is either an action $a \in \mathcal{A}$, or empty for τ, and these labels define the labelling function χ of the game. If no τ-transition is present in a subsystem, the transformation to normal form is performed automatically. Secondly, when composing games, often the same action name in the model needs to be assigned to different players in different subsystems, when it is controlled by one game, and regarded as external to the other, for instance, when modelling interfaces. Using $* \in \{!, ?\}$ in $[\texttt{a}*]$, transitions can be assigned to Player 1 by !, or to Player 2 by ?. Since we are concerned with turn-based games, no state can have outgoing transitions labelled by both ! and ?. Within a subsystem, no action can be assigned to more than one player, which is only a slight restriction in practice over the more general game composition in Definition 4.3. Thirdly, the *guard* `guard` is a predicate that has to be satisfied for the transition to be enabled. Finally, the probabilistic *update* $p_1 : (\texttt{update}_1) + p_2 : (\texttt{update}_2) \cdots$ specifies that with probability p_i the variables in the next state are determined by $\texttt{update}_i$, which is a list of updates of the form `s'`$=v$, meaning that `s` is updated to the value v. We require that $\sum_i p_i = 1$. For example, in Figure 7.1, the command

$$[\texttt{b}?]\ \texttt{s}{=}1 \rightarrow 0.5 : (\texttt{s'}{=}1) + 0.5 : (\texttt{s'}{=}2);$$

defines Player 2 transitions labelled with `b` from states where `s` = 1 to states where `s` is updated with equal probability to either 1 or 2, and all other variables stay the same. A guarded command enables a transition in each state where the guard holds.

Rewards. Models can be augmented with reward structures, using the syntax

```
rewards "r"
  [a] guard : value;
  ⋮
endrewards
```

This reward structure is named `r`, and it contains a list of reward assignments as guarded statements: the (optional) label [a] specifies an action $a \in \mathcal{A}$, the guard `guard` restricts the states the reward is applied to, and `value` defines the reward. Reward structures defined on actions only contain assignments of the form [a] `true : value`, that is, with trivially satisfied guard, since the reward must not depend on the state via the guard. For example, in Figure 7.1, the reward structures are given in the box on the right, where we define `r1`, `r2`, `r3`, and `c` on actions.

Specification Language. PRISM-games 2.0 supports specifications expressed as Boolean combinations of objectives, as discussed in Section 3.4.3. As in other versions of PRISM, specifications are called *properties*. We now describe the syntax for defining multi-objective queries: the syntax for the various objectives is given in the following table, where we take `r` and `c` to be reward structures satisfying the requirements of the respective objectives.

Objective	PRISM-games 2.0 syntax
$\mathsf{Pmp}(r)(v)$	`P>=1 [ R(path){"r"}>=v [ S ] ]`
$\mathsf{Pratio}(r/c)(v)$	`P>=1 [ R(path){"r"}/{"c"}>=v [ S ] ]`
$\mathsf{Emp}(r)(v)$	`R{"r"}>=v [ S ]`
$\mathsf{ratioE}(r/c)(v)$	`R{"r"}/{"c"}>=v [ S ]`
$\mathsf{Erew}(r)(v)$	`R{"r"}>=v [ C ]`

A target `v` can either be fixed as a number within the objective, or left as a named *constant*. Below, in Section 7.2.4, we explain how the constants allow us to compute compositional Pareto sets. The letters `S` and `C` stand for mean-payoff (dividing by the path length) and cumulative, respectively. Ratio objectives take the keyword `C`, as no division by the path length is performed.

To express Boolean combinations of objectives, PRISM-games 2.0 provides the infix symbols `&` (conjunction), `|` (disjunction), `!` (negation), and `=>` (implication).

```
const double v1 = 1/4;
const double v2 = 9/8;
const double v3 = 3/4;

"phi1" : <<1>> (R{"r1"}/{"c"}<=v1 [ S ] => R{"r2"}/{"c"}>=v2 [ S ])
"phi2" : <<1>> (R{"r1"}/{"c"}<=v1 [ S ] & R{"r3"}/{"c"}<=v3 [ S ])
"psi"  : comp("phi1", "phi2")
"phi"  : <<1>> (R{"r2"}/{"c"}>=v2 [ S ] & R{"r3"}/{"c"}<=v3 [ S ])
```

Figure 7.2: Properties for the PRISM-games 2.0 model in Figure 7.1, for the running example in Section 3.6.

For example, $(O_1 \,|\, !O_2)\,\&\,O_3$ expresses the specification $(O_1 \vee \neg O_2) \wedge O_3$. Arbitrary Boolean combinations can be expressed, which are converted automatically to conjunctive normal form (CNF) in order to apply the reduction to CQs from Theorem 5.17. Each property can receive a label, for example `phi` in "`phi`" : `not`(O_1), and so properties can be reused to define other properties. Strict inequalities in objectives are ignored (after issuing a warning), and interpreted as non-strict inequalities. For example, $\mathsf{ratio}\mathsf{E}^{\leq}(r_1/c)(v_1) \rightarrow \mathsf{ratio}\mathsf{E}^{\leq}(r_3/c)(v_3)$ is equivalent to $\mathbb{E}_{\mathcal{D}}[\mathsf{mp}(r_1)]/\mathbb{E}_{\mathcal{D}}[\mathsf{mp}(c)] > v_1 \vee \mathbb{E}_{\mathcal{D}}[\mathsf{mp}(-r_1)]/\mathbb{E}_{\mathcal{D}}[\mathsf{mp}(c)] \geq -v_1$, in an (induced) DTMC $\mathcal{D}$, but is interpreted as $\mathbb{E}_{\mathcal{D}}[\mathsf{mp}(r_1)]/\mathbb{E}_{\mathcal{D}}[\mathsf{mp}(c)] \geq v_1 \vee \mathbb{E}_{\mathcal{D}}[-\mathsf{mp}(r_1)]/\mathbb{E}_{\mathcal{D}}[\mathsf{mp}(c)] \geq -v_1$.

For our assume-guarantee framework (synthesis and Pareto set computation), PRISM-games 2.0 offers the keyword `comp` in order to specify a property for each component game individually. For example, for a game $\mathcal{G}^1 \parallel \mathcal{G}^2$, we can write `comp`$(\varphi^1, \varphi^2)$ to indicate that we want to analyse the model compositionally, using φ^1 for $\mathcal{G}^1$ and φ^2 for $\mathcal{G}^2$. A property not using the `comp` keyword is interpreted as a global property and used with the full product game.

Example$^{\mathsf{re}}$ 7.1. *In Figure 7.2, we list the properties for our running example of Section 3.6. The figure shows the properties file, which starts by defining the constants `v1`, `v2` and `v3` to be used in the properties. If no values are specified, they are queried in the GUI before the synthesis algorithm is invoked. Note that properties are evaluated on the top-level system, and so "`phi1`" and "`phi2`" on their own would mean evaluating $\mathcal{G}^1_{re} \parallel \mathcal{G}^2_{re} \models \varphi^1_{re}$ and $\mathcal{G}^1_{re} \parallel \mathcal{G}^2_{re} \models \varphi^2_{re}$, respectively, which is not the intention in the example. Rather, we use the compositional property "`psi`", which, by virtue of the `comp` keyword, evaluates $\mathcal{G}^1_{re} \models \varphi^1_{re}$ and $\mathcal{G}^2_{re} \models \varphi^2_{re}$, correctly assigning the local specifications to components. The global property φ_{re} is given in "`phi`".*

7.2 Implementation Details

In this section we describe the implementation of our strategy synthesis methods. We begin by describing in Section 7.2.1 the data structures underlying our implementation. In Section 7.2.2 we then give details about our fixpoint computation for the $F_{M,\vec{r}}$ and $F_{\mathsf{rew},\vec{r}}$ operators. In Section 7.2.3 we discuss how the transformations between the various multi-objective properties from Chapter 5 are implemented in order to synthesise strategies according to Chapter 6. Finally, in Section 7.2.4 we present how we approximate and visualise Pareto sets, both monolithically and compositionally.

7.2.1 Data Structures

We explain the most important data structures used in our implementation.

Explicit Game Representation. The current implementation relies on an explicit-state representation of the stochastic game models, building on the PRISM "explicit" engine. We extend the representation to accommodate games that are composed of several components, by adding a composition procedure implementing Definition 4.3, the (optional) compatibility check of Definition 4.4, and the normal form transformation of Definition 4.2. Further, to support strategy composition via Definition 4.5, we augment the data structure for stochastic games so that from each state $\vec{s} = (s^1, s^2, \ldots)$ we can reconstruct the states $s^1, s^2, \ldots$ of the individual components, and which component controls a Player 1 state, in order to reconstruct the components involved in the transitions (that is, the set $\Gamma(a, \vec{t})$ for the successor $\vec{t}$).

Representation of Polytopes. When performing the finite-step fixpoint approximation for expected energy objectives in Section 6.2.3, as well as for expected total rewards in Section 3.5.3, the sets we operate on are convex and closed polytopes. We use the Parma Polyhedra Library (PPL) [3] to perform the set-theoretic operations of convex union and intersection, and hence we use the polytope representation provided by the library. The PPL represents each polytope by both its vertices and the set of bounding hyperplanes, called the Motzkin double description method [3]. The vertex representation also allows for *rays*, which we use for the downward closure operation: a ray is a vector $\vec{y}$ such that, for any point $\vec{x}$ in a polytope X, any point $\vec{x} + \alpha \cdot \vec{y}$ is also in X, for any $\alpha > 0$. Both representations are equally expressive, but differ in how efficient operations are performed: intersection of polytopes is more efficient using hyperplanes (by taking the union of the bounding hyperplanes); convex union is more efficient using the vertices (by taking the union of the vertices).

The PPL automatically performs transformations between the representations, and can minimise the representation size. This representation of the polytopes is symbolic in the sense that we represent a polytope, an infinite set of points, by a finite set of vertices and hyperplanes. A similar approach in a discrete topology is representing sets as *antichains* [36], that is, by the extreme points of the sets.

Weighted Minkowski Sum. The weighted Minkowski sum operation for the stochastic states is not directly supported by the PPL, and we implement it using the vertex representation. We explain the process for two polytopes P_1 and P_2. Let $P_i \stackrel{\text{def}}{=} \{\vec{x} \in \mathbb{R}^n \mid \exists \vec{y} \in \mathbb{R}^{m_i}_{\geq 0} \,.\, V_i\vec{y} = \vec{x} \wedge \vec{1}^T\vec{y} = 1\}$ for $i \in \{1, 2\}$, where V_i is an $n \times m_i$ matrix defining the vertices of the polytope. The idea behind computing their weighted Minkowski sum is the following. First, lift the space to dimension $n+1$ and place P_1 and P_2 at a distance of $-\frac{1}{1-\alpha}$ and $\frac{1}{\alpha}$ from the origin respectively. Their convex hull, shown with dashed lines in Figure 7.3, can then be computed as

$$\{\vec{x} \in \mathbb{R}^{n+1} \mid \exists \vec{y} \in \mathbb{R}^{m_1+m_2+1} \,.\, \begin{bmatrix} V_1 & V_2 \\ \vec{1}^T/\alpha & -\vec{1}^T/(1-\alpha) \end{bmatrix} \vec{y} = \vec{x} \wedge \vec{1}^T\vec{y} = 1\}.$$

Constraining this polytope to $x_{n+1} = 0$, we get $\vec{1}^T\vec{y}_1 = \alpha$ and $\vec{1}^1\vec{y}_2 = 1 - \alpha$, and hence we define $\alpha\vec{z}_1 = \vec{y}_1$ and $(1-\alpha)\vec{z}_2 = \vec{y}_2$, so that $\vec{x} = \alpha V_1\vec{z}_1 + (1-\alpha)V_2\vec{z}_2$ and $\vec{1}^T\vec{z}_1 = \vec{1}^T\vec{z}_2 = 1$. This corresponds to computing the weighted Minkowski sum $\alpha \times P_1 + (1-\alpha) \times P_2$, as illustrated by the hatched polytope in Figure 7.3. Computing the convex hull and constraining to $x_{n+1} = 0$ are supported by the PPL. This method extends to more than two polytopes in a similar fashion by introducing an extra dimension per polytope [76].

Note that the weighted Minkowski sum is the most computationally expensive operation that we have to implement, as the number of vertices of the polytope $\alpha P_1 + (1-\alpha)P_2$ is $\mathcal{O}(|P_1| \cdot |P_2|)$, see Theorem 4.1.1 of [136]. In contrast, the number of vertices of the polytopes $P_1 \cap P_2$ and $\mathsf{conv}(P_1 \cup P_2)$ is $\mathcal{O}(|P_1| + |P_2|)$ in both cases. The performance of our synthesis algorithms is therefore dependent on the outgoing branching degree of the states.

Strategies. We represent SU strategies by simply encoding the next choice function and memory update functions as maps. In the assume-guarantee framework, the synthesised strategies are not explicitly composed, but the individual strategies are stored separately. When simulating a composed game under a composed strategy, the memory update is performed at each step only for the strategies corresponding to the involved components in the set $\Gamma(a, \vec{t})$ defined in Section 4.1.4, and the next choice

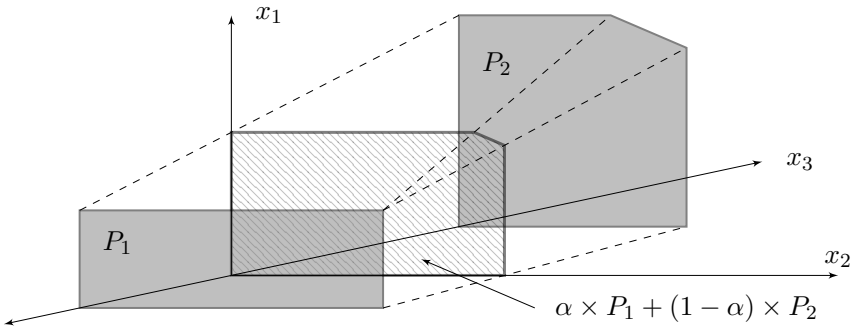


Figure 7.3: Illustrating the computation of the weighted Minkowski sum. P_1 is shifted from the origin by $x_3 = -\frac{1}{1-\alpha}$, P_2 is shifted from the origin by $x_3 = \frac{1}{\alpha}$.

Table 7.1: Summary of algorithm parameters used in PRISM-games 2.0.

Parameter	Set by User	Description
ε	✓	termination accuracy
a_0	✓	baseline rounding accuracy
a_k		rounding accuracy for iteration k
η	✓	increase factor for rounding accuracy
$M_{\min}$, $M_{\max}$	✓	minimum and maximum box size
$k_{\max}$	✓	maximum iteration count for conjunctions
$d_{\max}$	✓	maximum iteration count for disjunctions
x_{init}		initial weight vector
$\mathsf{acc}_{\text{init}}$		initial weight vector accuracy

in a Player 1 state $\vec{s}$ is determined by the strategy controlling $\vec{s}$. Thus, as mentioned above, when composing games, we keep track of the underlying structure of the states and moves, in order to be able to extract $\Gamma(a, \vec{t})$, and the components s_i of states $\vec{s}$.

7.2.2 Fixpoint Computation

At the heart of our implementation is the shortfall fixpoint computation of the $F_{M,\vec{r}}$ operator from Section 6.2.1 for Pmp objectives, as well as the Pareto set approximation for Erew objectives using the $F_{\mathsf{rew},\vec{r}}$ operator from Section 3.5.3. We give the details of our implementation, focusing on how to make the strategy synthesis more efficient. The parameters used in the algorithms are summarised in Table 8.1.

Long-Run Objectives. For strategy synthesis of Pmp objectives we compute the sequence $X^0, X^1, X^2, \ldots$,using $X^k = F^k_{M,\vec{r}}(\bot_M)$, for which we have $X^k \sqsubseteq X^{k+1}$ for all $k \geq 0$. From Proposition 6.27 we can construct ε-optimal strategies for Pmp objectives from the sets X^k, if $X^{k+1} + \varepsilon \sqsubseteq X^k$ and $X^k_{s_{\mathsf{init}}} \neq \emptyset$. This stopping criterion ensures that

the expected energy does not degrade by more than ε per step, yielding a degradation of at most ε in the mean-payoff. The same computation is used for strategy synthesis of Pratio, Emp and ratioE objectives, via the transformations described in Chapter 5.

Expected Total Rewards. For non-negative rewards in stopping games, the operator $F_{\mathsf{rew},\vec{r}}$ of Section 3.5.3 defines a $\subseteq$-monotonic sequence $(F^k_{\mathsf{rew},\vec{r}}(\mathsf{dwc}(\vec{0})))_{k\geq 0}$, and we show in Theorem 4 of [41] that we can compute ε-approximate Pareto sets for Erew after a finite number of iterations of $F_{\mathsf{rew},\vec{r}}$. To achieve a target $\vec{v} \in F^{k+1}_{\mathsf{rew},\vec{r}}(\mathsf{dwc}(\vec{0}))$, under the condition that $F^k_{\mathsf{rew},\vec{r}}(\mathsf{dwc}(\vec{0})) \subseteq F^{k+1}_{\mathsf{rew},\vec{r}}(\mathsf{dwc}(\vec{0}))$, Theorem 3 of [43] shows a strategy construction, which is a special case of our Definition 6.2, where choice functions are Dirac distributions. This latter requirement is, however, not necessary.

If $\vec{r}$ assigns negative rewards, then the sequence $(F^k_{\mathsf{rew},\vec{r}}(\mathsf{dwc}(\vec{0})))_{k\geq 0}$ may not be non-decreasing with respect to $\subseteq$. We handle negative rewards by starting at the lowest possible expected total reward for each objective i, assuming that these rewards are bounded, that is, we define $\bot \in (\mathcal{P}(\mathbb{R}^n))^{|S|}$ by

$$\bot_s \stackrel{\text{def}}{=} \{\vec{x} \in \mathbb{R}^n \mid \forall i \,.\, x_i \leq \inf_{\pi} \inf_{\sigma} \mathbb{E}^{\pi,\sigma}_{\mathcal{G},s}[\mathsf{rew}(r_i)]\},$$

for all $s \in S$. Letting $Y^k \stackrel{\text{def}}{=} F^k_{\mathsf{rew},\vec{r}}(\bot)$, the sequence $(Y^k)_{k\geq 0}$ is non-decreasing with respect to $\subseteq$, and we can construct strategies for any $\vec{v} \in Y^{k+1}$ as follows.

Theorem 7.1. *Let $Y \in (\mathcal{P}(\mathbb{R}^n))^{|S|}$, and let $\vec{v} \in F_{\mathsf{rew},\vec{r}}(Y)$, for $\vec{r} : S \to \mathbb{R}^n$. If $Y \subseteq F_{\mathsf{rew},\vec{r}}(Y)$, then the strategy $\pi(F_{\mathsf{rew},\vec{r}}(Y), \vec{r}, 0)$ of Definition 6.2 satisfies $\mathsf{Erew}(\vec{r})(\vec{v})$.*

Proof. A direct consequence of Theorem 3 of [43] and the above discussion. □

For stopping games, $\inf_\pi \inf_\sigma \mathbb{E}^{\pi,\sigma}_{\mathcal{G},s}[\mathsf{rew}(r_i)]$ in the definition of $\bot = Y^0$ is bounded, and so can be computed using the MDP methods of PRISM. Since the only difference between $F_{\mathsf{rew},\vec{r}}$ and $F_{M,\vec{r}}$ is the restriction to Box_M, so we unify the implementation of the two operators: we perform the same operations on the polytopes, and additionally intersect with Box_M in the case of strategy synthesis of long-run objectives.

Note that an alternative method for handling negative rewards is to introduce an ε-degradation in the strategy construction, which is the approach in [124].

Gauss-Seidel Update. The following discussion is presented for the sequence $(X^k)_{k\geq 0}$, but applies likewise to $(Y^k)_{k\geq 0}$. During the fixpoint computation with $F_{M,\vec{r}}$, we compute X^{k+1}_s from X^k_s for all states $s \in S$, before computing X^{k+2}_s from X^{k+1}_s for all states s, and so on, and similarly for the computation of Y^k using $F_{\mathsf{rew},\vec{r}}$. This

process can be accelerated by allowing in-place updates, also called *Gauss-Seidel updates*. That is, when X_s^{k+1} is computed from X^k, the result is stored in the same memory location as X_s^k. Subsequently, after X_t^{k+1} has been computed, and X_s^{k+1} is being computed for $s \neq t$, where t is a successor of s, then X_t^{k+1} is used instead of X_t^k. Correctness of this method follows from the monotonicity of the operator $F_{M,\vec{r}}$, shown in Proposition 6.19, and respectively from the monotonicity of the operator $F_{\mathsf{rew},\vec{r}}$, shown in [43]. Without the Gauss-Seidel update, we can interpret the fixpoint computation as a backward computation exploring the tree of paths through the game in a strictly step-by-step fashion. The Gauss-Seidel update allows for some of these steps to be skipped, and thus the same depth in the tree can be explored with fewer steps. Due to this skipping of steps, the Gauss-Seidel update is not applicable to our method of approximating Pareto sets for Emp and ratioE objectives (see Section 5.5.1).

Stopping Criteria. For the synthesis of Pmp CQs, we can construct a strategy when the fixpoint computation of $F_{M,\vec{r}}$ satisfies $X^{k+1} + \varepsilon \sqsubseteq X^k$ and $X^k_{s_{\mathsf{init}}} \neq \emptyset$, which is a relative stopping criterion. For synthesis of Erew CQs, the stopping criterion for $F_{\mathsf{rew},\vec{r}}$ relies on the monotonicity of the operator. Then, a strategy achieving $\vec{v}$ is sufficient, even if the Pareto set contains a point $\vec{w} > \vec{v}$. We thus obtain the (absolute) stopping criterion $\vec{v} \in Y^{k+1}_{s_{\mathsf{init}}}$, and, for completeness of the algorithm for ε-optimal strategies, we add the (relative) stopping criterion $Y^k \not\subseteq Y^{k-1} + \varepsilon$. To compute Pareto sets for Erew CQs, we terminate when $Y^{k+1} \subseteq Y^k + \varepsilon$, which is a relative stopping criterion. Pareto sets for Emp CQs are computed using the Erew Pareto set algorithm, and hence use the same stopping criterion. Note that we are not interested in achieving specific shortfalls for EE computed by $F_{M,\vec{r}}$.

Rounding. The following discussion is presented for the sequence $(X^k)_{k\geq 0}$, but applies likewise to $(Y^k)_{k\geq 0}$. The number of extreme points $|\mathsf{C}(X_s^k)|$ of the sets X_s^k may increase exponentially with k, due to the weighted Minkowski sum and convex union operators. To mitigate this complexity, at every step k, we can fix some accuracy $a_k \gg 0$, and round down each coordinate i of each extreme point of X_s^k to a multiple of $b_k \stackrel{\text{def}}{=} \frac{\rho^*}{a_k}$, where ρ^* is the largest magnitude reward in the game, as in Lemma 5.20, and where a_k is the *accuracy* for step k. Formally, we define the rounding $\lfloor X_s^k \rfloor_{b_k} \stackrel{\text{def}}{=} \{\vec{y} \in \mathbb{R}^n \mid \exists \vec{x} \in X_s^k \,.\, \forall i \,.\, y_i = \lfloor x_i \rfloor_{b_k}\}$, and thus compute the rounded sequence $\tilde{X}^{k+1} \stackrel{\text{def}}{=} \lfloor F_{M,\vec{r}}(\tilde{X}^k) \rfloor_{b_k} \cup \tilde{X}^k$. When computing $\tilde{X}^{k+1}$ from $\tilde{X}^k$, we must round *down* (and in the case of $\tilde{Y}^{k+1}$ take the union with $\tilde{Y}^k$), in order to ensure that the resulting sequence is monotonic with respect to $\sqsubseteq$, that is, $\tilde{X}_s^0 = \bot_M \sqsubseteq \tilde{X}_s^1 \sqsubseteq \tilde{X}_s^2 \sqsubseteq \cdots$. To select the

```
$SU.strat − v0.1
startstrategy
States :
3
InitState :
1
Init :
{0 = 1.0}
Next :
0 0 {0 = 1.0}
0 1 {1 = 1.0}
MemUpdStates :
0 0 0 {0 = 1.0}
0 1 1 {0 = 1.0}
1 0 0 {0 = 1.0}
1 0 1 {0 = 1.0}
2 0 0 {0 = 1.0}
2 1 0 {1 = 1.0}
MemUpdMoves :
0 0 0 1 {0 = 1.0}
0 0 0 2 {1 = 1.0}
0 1 0 1 {0 = 1.0}
1 0 0 2 {0 = 1/3, 1 = 2/3}
1 1 0 1 {0 = 1.0}
1 1 0 2 {0 = 2/3, 1 = 1/3}
2 0 0 0 {0 = 1.0}
2 0 1 0 {1 = 1.0}
Info :
maximum C-iterations:  100
relative termination threshold:  0.00001
endstrategy
```

Figure 7.4: Strategy computed for the game in Figure 7.1 under property φ^2_{re} of Section 3.6, as exported by PRISM-games 2.0. We approximate decimals as fractions, and memory reachable due to rounding errors is removed. This is the strategy shown in Figure 6.4.

accuracies a_k, we start by selecting $a_0 \gg 0$, together with some *increase factor* $\eta \geq 1$, and let $a_k = \eta \cdot a_{k-1}$ for all $k > 0$. This leads to a dynamically increased accuracy, with the benefit that the polytope computation is accelerated initially by performing large steps towards convergence, and, as the sets grow, their accuracy is improved. Setting $\eta = 1$ disables this dynamically increased accuracy. With this approach, we obtain under-approximations of the Pareto frontiers with a small number of points that are gradually refined by allowing more points in the polytopes. Note that some parts of the implementation use floating point operations, limiting the accuracy.

Strategy Construction. We implement the strategy construction of Definition 6.2, which we parameterise for Pmp and Erew objectives using Proposition 6.27 and Theorem 7.1, respectively. Using the implemented transformations of Chapter 5, we obtain strategies for all objective types studied in this thesis. We find the distributions α and β^t of Definition 6.2 by converting the conditions imposed on them by

the memory mapping into linear programs, which can be efficiently solved. During the strategy construction, we only consider the reachable memory elements, which yields more compact strategies, and reduces the computation time. In the strategy of Example 6.3, shown in Figure 6.4, we have already demonstrated this idea. We solve the linear programs to a limited accuracy, meaning that rounding errors are introduced, which may change the memory elements that are accessed, but, by the same argument as in the strategy discretisation in Section 6.3.3, this does not affect the quality of the strategy beyond a bounded deviation.

Example$^{\mathsf{re}}$ 7.2 (Synthesised Strategy)**.** *In Figure 7.4, we show an example strategy computed by PRISM-games 2.0 as exported to a file. The strategy is the one shown in Figure 6.4, synthesised for $\mathcal{G}^2_{re}$ under φ^2_{re} in our running example of Section 3.6. The output consists of seven sections. Under the heading* `States`*, the memory size is given. PRISM-games 2.0 models have a specific initial state, and the strategy is specific to this state, given under the heading* `InitState`*. Under* `Next` *the next choice function is given: $\pi_c(s, \mathfrak{m})(t) = p$ is a line of the form "s $\mathfrak{m}$ $i = p$", where i stands for the ith move t from s. Under* `MemUpdStates` *the next memory update function at states is given: $\pi_u(\mathfrak{m}, t)(\mathfrak{n}) = p$ is a line of the form "s $\mathfrak{m}$ i $\mathfrak{n} = p$", where s is the state that $\mathfrak{m}$ is associated with, and i stands for the ith move t from s. Similarly, under* `MemUpdMoves` *the next memory update function at moves is given: $\pi_u(\mathfrak{m}, u)(\mathfrak{n}) = p$ is a line of the form "s i $\mathfrak{m}$ $u = p$", where i is the ith move t at state s, and $\mathfrak{m}$ is associated with the move t. Finally, under the heading* `Info` *additional information about the strategy is presented; here, for example, the accuracy ε is recorded.*

7.2.3 Synthesis Algorithms

We now give the pseudo code for the main algorithms implemented in PRISM-games 2.0, both for strategy synthesis and Pareto set computation. We summarise the algorithms in Figure 7.5, which we refer to throughout this section for further explanation.

First, the synthesis algorithm for Pmp CQs, as developed in Section 6.4, is given as Algorithm 1. It is based on the $F_{M,\vec{r}}$ operator, where we initialise the box size M to some $M_{\min} > 1$. For the sake of user convenience, we also implement an upper bound on the box size, $M_{\max}$, and on the number of fixpoint iterations, $k_{\max}$. The same algorithm is applicable to Pratio CQs, and for CM games to Emp and ratioE CQs, using the transformations of Chapter 5. We do not implement the decision procedure from Section 6.1, because even if it reports that the specification is achievable, for

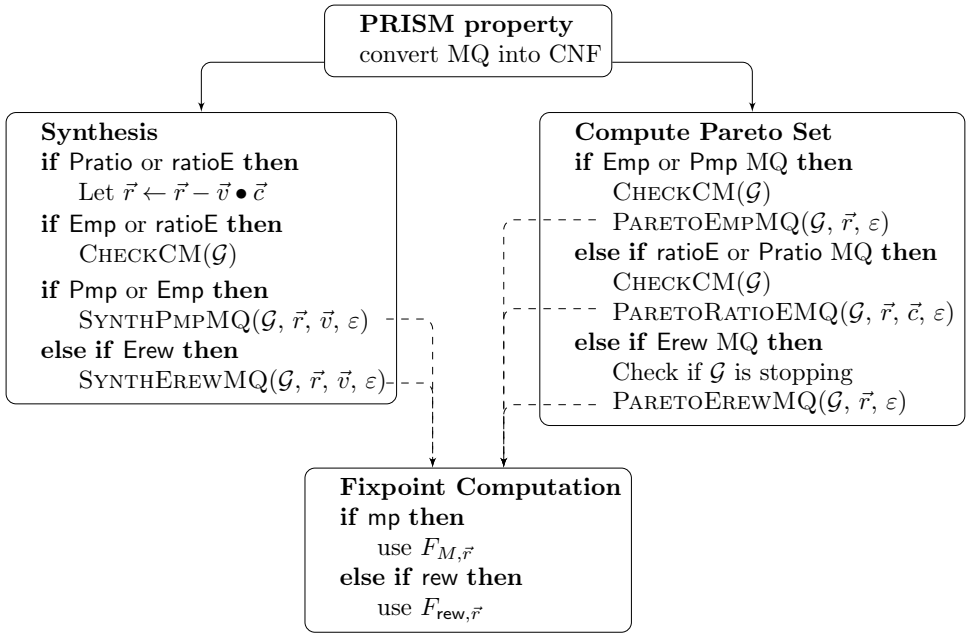


Figure 7.5: Algorithms implemented in PRISM-games 2.0. The dashed arrows indicate subroutine calls.

Algorithm 1 Strategy Synthesis for CQ $\bigwedge_{i=1}^{n} \mathsf{Pmp}(r_i)(v_i)$

function SynthPmpCQ($\mathcal{G}$, $\vec{r}$, $\vec{v}$, ε)
  $M \leftarrow M_{\min}$;
  while $M < M_{\max}$ **do**
    $\pi \leftarrow$ SynthPmpCQ($\mathcal{G}$, $\vec{r}$, $\vec{v}$, ε, M);
    if $\pi \neq$ `null` **then return** π;
    $M \leftarrow M^2$;

function SynthPmpCQ($\mathcal{G}$, $\vec{r}$, $\vec{v}$, ε, M)
  $\vec{r}' \leftarrow \vec{r} - \vec{v} + \frac{\varepsilon}{2}$; $k \leftarrow 0$; $X^0 \leftarrow \bot_M$;
  do
    $X^{k+1} \leftarrow F_{M,\vec{r}'}(X^k)$; $k{+}{+}$;
  while $X^k + \frac{\varepsilon}{2} \not\sqsubseteq X^{k-1}$ and $k < k_{\max}$ ▷ convergence condition
  if $X^k_{s_{\mathsf{init}}} \neq \emptyset$ **then**
    return $\pi(X^{k-1}, \vec{r}', \frac{\varepsilon}{2})$;
  return `null`; ▷ if diverged to empty sets

efficiency reasons rounding is typically used, which limits the accuracy. Finally, the algorithm for Erew CQs, based on the $F_{\mathsf{rew},\vec{r}}$ operator, is shown as Algorithm 2.

In the rest of this section we focus on strategy synthesis for general MQs, which we assume here are given in CNF. Consider a specification $\psi = \bigwedge_{i=1}^{n} \bigvee_{j=1}^{m} \mathsf{Emp}(r_{i,j})(v_{i,j})$.

Algorithm 2 Strategy Synthesis for CQ $\bigwedge_{i=1}^{n} \mathsf{Erew}(r_i)(v_i)$

```
1: function SynthErewCQ(𝒢, r⃗, v⃗, ε)
2:     Let k ← 0; Y^0 ← ⊥;
3:     do
4:         Y^{k+1} ← F_{rew,r⃗}(Y^k); k++;
5:         if v⃗ ∈ Y^k_{s_init} then                          ▷ stopping criterion
6:             return ConstructStrategy(Y^k, 0);
7:     while Y^k ⋢ Y^{k−1} + ε and k < k_max                  ▷ convergence condition
8:     return null;                                          ▷ if converged without achieving v⃗
```

By Theorem 5.17, we can convert this MQ into a CQ $\varphi = \bigwedge_{i=1}^{n} \mathsf{Emp}(\vec{x}_i \cdot \vec{r}_i)(\vec{x}_i \cdot \vec{v}_i)$, so that, if a winning strategy exists for φ, the same strategy is winning for ψ. The caveat is that we need to select the weights $\vec{x}_1, \ldots, \vec{x}_n$ appropriately in order for φ to be achievable. Since the weights cannot be known a-priori, we develop a heuristic to iterate over them, as already outlined in [11] and Section 5.5.

Weight Selection. For a MQ in CNF with n conjuncts of m objectives each, the basic idea is to iterate over weight vectors $\vec{x}$ of dimension $n \cdot m$, gridding the entire search space, as suggested in Lemma 5.20. We observe that, due to the linearity of expectations, a CQ $\bigwedge_{i=1}^{n} \mathsf{Emp}(\alpha_i \cdot r_i)(\alpha_i \cdot v_i)$ has the same achievability result for any $\vec{\alpha} \in \mathbb{R}^n_{>0}$. Hence, for each conjunct i, only the direction of the weight vectors $\vec{x}_i$ is relevant, and so we reduce the search space from $n \cdot m$ to $n \cdot (m-1)$ dimensions. We show the algorithm for selecting weights as Algorithm 3. The NextWeight function takes the current weight vector $\vec{x}$, the accuracy acc, and a parameter $\vec{E}$ indicating which dimensions are currently excluded from being iterated over (according to the mentioned reduction of the search space). $\vec{E}$ is an n-dimensional tuple of indices between 1 and m, which initially is set to $(1, 1, \ldots, 1)$, that is, initially we grid all dimensions except the first for every conjunct. We also assume that acc is initialised to $\mathsf{acc}_{\text{init}} \stackrel{\text{def}}{=} \frac{1}{2}$, and that $\vec{x}$ is initialised to $\vec{x}_{\text{init}} \stackrel{\text{def}}{=} ((\frac{3}{4}, \frac{3}{4}, \frac{3}{4}, \ldots), (\frac{3}{4}, \frac{3}{4}, \frac{3}{4}, \ldots), \ldots)$. In lines 2–3, the next weight is selected by gridding the next available dimension with the accuracy acc. If all dimensions (except the ones excluded by $\vec{E}$) are exhaustively gridded, we change the excluded dimensions in lines 5–6, unless all dimensions were already excluded under this setting of accuracy; in this case, we increase the accuracy in line 8, and reset $\vec{E}$. In line 9, since $\vec{E}$ has changed, we re-initialise the weight $\vec{x}$ with the new accuracy.

We refer to Figure 7.6 to illustrate the weight selection algorithm. The pattern of selected vectors is shown for $n = 1$ and $m = 2$, that is, for a disjunction consisting of two objectives. Figure 7.6 shows the weight vectors $\vec{x}$ as dots for the accuracies

Algorithm 3 Weight Vector Selection

function NEXTWEIGHT($\vec{x}$, acc, $\vec{E}$)
  if there are indices (i, j) such that $E_i \neq j$ and $x_{i,j} >$ acc **then**
    pick the lexicographically least such pair (i, j) and set $x_{i,j} \leftarrow x_{i,j} -$ acc;
  else
    if there are indices (i, j) such that $E_i = j$ and $j < m - 1$ **then**
      pick the lexicographically least such pair (i, j) and set $E_i \leftarrow j + 1$;
    else
      acc $\leftarrow$ acc$/2$; $\vec{E} \leftarrow (1, 1, \ldots, 1)$; ▷ increase accuracy
    for all indices (i, j), let $x_{i,j} \leftarrow 1 -$ acc$/2$;
  return $(\vec{x}, \text{acc}, \vec{E})$;

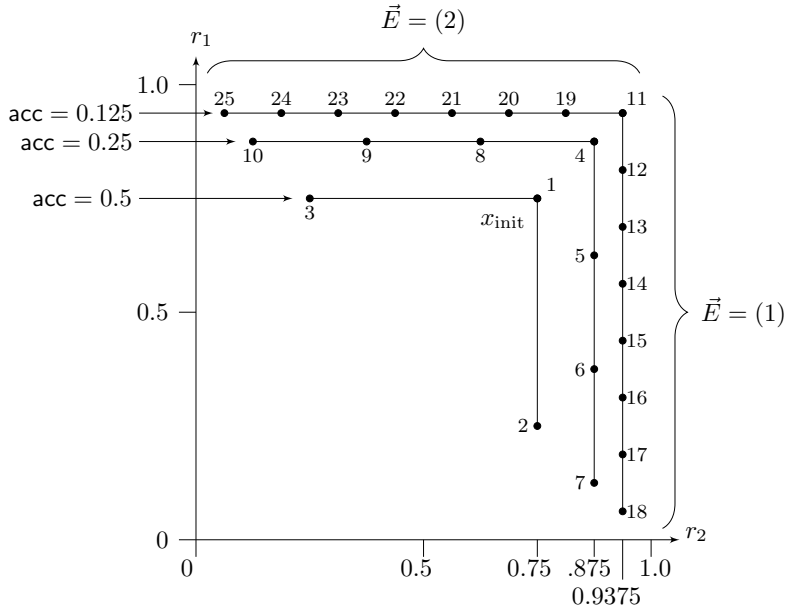


Figure 7.6: The first 25 weight vectors selected using Algorithm 3 for $n = 1$ and $m = 2$, with the sequence number written next to the weights.

acc $= 0.5$, acc $= 0.25$ and acc $= 0.125$, and demonstrates that Algorithm 3 selects weights in a way that grids the space of weight vectors with increasingly refined accuracy, and realises the τ-discretisation of $\mathsf{Weights}_{\mathsf{MQ}}(\vec{\rho}^*)$ of Lemma 5.20.

Synthesis of MQs. Using our weight selection algorithm, we now show how to synthesise Boolean combinations of objectives. We show the algorithm for Emp MQs as Algorithm 4, which is applicable to CM games. We use the notation $[q_i]_{i=1}^n$ to stand for the vector $\vec{q}$ with $[\vec{q}] = q_i$. For ratioE MQs in CM games, the same algorithm can be applied. Note that, when iteratively searching over weight vectors, we have to iterate

over the size M of Box_M as the outermost iteration, since we are not guaranteed that the algorithm terminates for any choice of M, see the discussion of our algorithm in Section 6.4. For MQs of Erew objectives in stopping games, the same method for synthesis via our weight selection is applicable, which we give as Algorithm 5.

Algorithm 4 Strategy Synthesis for MQ $\bigwedge_{i=1}^{n} \bigvee_{j=1}^{m} \mathsf{Emp}(r_{i,j})(v_{i,j})$

```
1: function SynthEmpMQ(𝒢, r⃗, v⃗, ε)
2:     x⃗ ← x⃗_init; acc ← acc_init; E⃗ ← (1, 1, ..., 1); M ← M_min;
3:     while M < M_max do
4:         for d ← 0; d < d_max; d++ do
5:             (x⃗, acc, E⃗) ← NextWeight(x⃗, acc, E⃗)
6:             π ← SynthPmpCQ(𝒢, [x⃗_i · r⃗_i]_{i=1}^n, [x⃗_i · v⃗_i]_{i=1}^n, ε, M)
7:             if π ≠ null then return π;
8:         M ← M^2;
```

Algorithm 5 Strategy Synthesis for MQ $\bigwedge_{i=1}^{n} \bigvee_{j=1}^{m} \mathsf{Erew}(r_{i,j})(v_{i,j})$

```
1: function SynthErewMQ(𝒢, r⃗, v⃗, ε)
2:     x⃗ ← x⃗_init; acc ← acc_init; E⃗ ← (1, 1, ..., 1);
3:     for d ← 0; d < d_max; d++ do
4:         (x⃗, acc, E⃗) ← NextWeight(x⃗, acc, E⃗)
5:         π ← SynthErewCQ(𝒢, [x⃗_i · r⃗_i]_{i=1}^n, [x⃗_i · v⃗_i]_{i=1}^n, ε)
6:         if π ≠ null then return π;
```

7.2.4 Pareto Set Computation

We show our algorithms for approximating Pareto sets, and explain how we visualise higher dimensional Pareto sets.

Pareto Sets for CQs. For Erew objectives, the algorithm is shown as Algorithm 6. For stopping games, we may leave the parameter $k_{\max}$ to its default value of (conceptually) ∞, since the Pareto computation is guaranteed to converge (Theorem 4 of [41]). The Pareto set computation for Emp CQs, as discussed in Section 5.5, is based on computing approximations of Pareto sets for Erew CQs, and dividing each vector in the resulting sets by the number of computation steps. Hence, we use the ParetoErewCQ algorithm to approximate Pareto sets for Emp CQs in CM games by bounding the number of iterations with a finite $k_{\max}$, leading to bounded Pareto sets also in non-stopping games. We show the algorithm as Algorithm 7. Note that in CM games the Pareto sets for Pmp CQs are equivalent to those for Emp CQs, but

PRISM-games 2.0 does not support computing Pareto sets for Pmp CQs in general games.

Algorithm 6 Pareto Set Computation for CQ $\bigwedge_{i=1}^{n} \mathsf{Erew}(r_i)(v_i)$

function ParetoErewCQ($\mathcal{G}$, $\vec{r}$, ε)
  Let $k \leftarrow 0$; $Y^0 \leftarrow \bot$;
  do
    $Y^{k+1} \leftarrow F_{\mathsf{rew},\vec{r}}(Y^k)$; $k{+}{+}$;
  while $Y^k \not\subseteq Y^{k-1} + \varepsilon$ and $k < k_{\max}$ ▷ convergence condition
  return Y^k;

Algorithm 7 Pareto Set Computation for CQ $\bigwedge_{i=1}^{n} \mathsf{Emp}(r_i)(v_i)$

function ParetoEmpCQ($\mathcal{G}$, $\vec{r}$, ε)
  $Y \leftarrow$ ParetoErewCQ($\mathcal{G}$, $\vec{r}$, ε);
  return $\frac{1}{k_{\max}} \times Y$;

For ratioE CQs, we approximate Pareto sets by gridding the space and testing for every point if it is achievable, see Section 5.5.2. Since the Pareto sets for ratioE CQs are convex, we can make this method slightly more efficient by including the convex downward closure of the points that were found achievable, and excluding the upward closure of the points that were found not achievable. We show the algorithm as Algorithm 8. The gridding is implemented using the weight selection of Algorithm 3.

Algorithm 8 Pareto Set Computation for CQ $\bigwedge_{i=1}^{n} \mathsf{ratioE}(r_i/c_i)(v_i)$

function ParetoRatioECQ($\mathcal{G}$, $\vec{r}$, $\vec{c}$, ε)
  Initialise $P = \emptyset$, $Q = \emptyset$
  for $\vec{v} \in \mathbb{R}^n$ finitely gridded **do**
    if $\vec{v} \in P \cup Q$ **then**
      continue; ▷ value already subsumed
    if SynthPmpCQ($\mathcal{G}$, $\vec{r} - \vec{c} \bullet \vec{v}$, $\vec{0}$, ε) $\neq$ `null` **then**
      $P \leftarrow \mathsf{dwc}(\mathsf{conv}(P \cup \{\vec{v}\}))$;
    else
      $Q \leftarrow \mathsf{upc}(Q \cup \{\vec{v}\})$;
  return P;

Pareto Sets for MQs. To compute Pareto sets for MQs, every point $\vec{v} \in \mathbb{R}^n$ in a CQ Pareto set with weights $\vec{x}_1, \ldots, \vec{x}_n \in \mathbb{R}^m_{\geq 0}$ corresponds to intersection of half-spaces $\vec{x}_i \cdot \vec{z} \leq v_i$, and the union over all choices of weight vectors is the ε-approximation of the corresponding MQ Pareto set. We show the algorithm to compute Pareto sets

Algorithm 9 Pareto Set Computation for MQ $\bigwedge_{i=1}^{n} \bigvee_{j=1}^{m} \mathsf{Emp}(r_{i,j})(v_{i,j})$

1: **function** ParetoEmpMQ($\mathcal{G}$, $\vec{r}$, $\vec{v}$, ε)
2: $\quad\vec{x} \leftarrow \vec{x}_{\text{init}}$; $\mathsf{acc} \leftarrow \mathsf{acc}_{\text{init}}$; $\vec{E} \leftarrow (1, 1, \ldots, 1)$; $P \leftarrow \emptyset$
3: $\quad$**for** $d \leftarrow 0$; $d < d_{\max}$; $d{+}{+}$ **do**
4: $\quad\quad(\vec{x}, \mathsf{acc}, \vec{E}) \leftarrow$ NextWeight($\vec{x}$, acc, $\vec{E}$)
5: $\quad\quad P_{\vec{x}} \leftarrow$ ParetoEmpCQ($\mathcal{G}$, $[\vec{x}_i \cdot \vec{r}_i]_{i=1}^{n}$, ε)
6: $\quad\quad P \leftarrow P \cup \mathsf{conv}(\bigcup_{\vec{v} \in P_{\vec{x}}} \bigcap_{i=1}^{n} \{\vec{u} \in \mathbb{R}^{n \times m} \mid \vec{x}_i \cdot \vec{u}_i \leq v_i\})$;
7: $\quad$**return** P;

Algorithm 10 Pareto Set Computation for MQ $\bigwedge_{i=1}^{n} \bigvee_{j=1}^{m} \mathsf{ratioE}(r_{i,j}/c_{i,j})(v_{i,j})$

1: **function** ParetoRatioEMQ($\mathcal{G}$, $\vec{r}$, $\vec{v}$, ε)
2: $\quad\vec{x} \leftarrow \vec{x}_{\text{init}}$; $\mathsf{acc} \leftarrow \mathsf{acc}_{\text{init}}$; $E \leftarrow (1, 1, \ldots, 1)$; $P \leftarrow \emptyset$
3: $\quad$**for** $d \leftarrow 0$; $d < d_{\max}$; $d{+}{+}$ **do**
4: $\quad\quad(\vec{x}, \mathsf{acc}, \vec{E}) \leftarrow$ NextWeight($\vec{x}$, acc, $\vec{E}$)
5: $\quad\quad Q_{\vec{x}} \leftarrow$ ParetoRatioECQ($\mathcal{G}$, $\vec{\rho}$, $\vec{\gamma}$, ε)
6: $\quad\quad P \leftarrow P \cup \mathsf{conv}(\bigcup_{\vec{\kappa} \in Q_{\vec{x}}} \bigcap_{i=1}^{n} \{\vec{u} \in \mathbb{R}^{n \times m} \mid \vec{x}_i \cdot \vec{u}_i \leq \vec{x}_i \cdot \vec{w}_i(\kappa_i)\})$;
7: $\quad$**return** P;

for Emp MQs as Algorithm 9, which is based on Equation (5.7). In stopping games, the same algorithm can be used for MQs of Erew objectives. The corresponding algorithm for ratioE MQs is shown as Algorithm 9, which is based on Equation (5.9). Since we evaluate the union of the convex Pareto sets computed for CQs, the Pareto sets obtained for the MQs might not be convex, and so, internally, we store the Pareto sets as lists of convex polytopes, ensuring no set in this list is fully contained in another.

Visualisation. In order to pick a target conveniently, as in step (S3) of Section 4.4, we implement a visualisation of Pareto set approximations. Pareto sets of two dimensions can be straightforwardly plotted in a coordinate system. For higher dimensions, we use a method based on displaying two-dimensional slices of the Pareto set to the user, also advocated in [88]. The user can select two dimensions to plot against each other, and the other dimensions will be either *sliced* at a given value, or *projected*. Projecting out a dimension i of $X \subseteq \mathbb{R}^n$ means computing $\{\vec{y} \in \mathbb{R}^{n-1} \mid \exists \vec{x} \in X \,.\, (y_1, \ldots, y_{n-1}) = (x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n)\}$. A single slicing operation of a set $X \subseteq \mathbb{R}^n$ means fixing a dimension i to a value v, that is, $\{\vec{y} \in \mathbb{R}^n \mid \exists \vec{x} \in X \,.\, x_i = v \wedge \forall j \neq i \,.\, y_j = x_j\}$, and projecting out i. We illustrate the visualisation of Pareto sets in Figure 7.10, where a three-dimensional Pareto set is shown with its first dimension projected out.

Compositional Pareto Sets. To obtain compositional Pareto sets, we use the approach of Section 4.3. The operations of lifting, intersecting and projecting of the local Pareto sets are performed using the PPL. The key challenge here is to ensure that the correct dimensions are associated with each other. For example, in the (ASYM) rule, one component has a specification $\mathsf{ratioE}(r_1/c)(v_1) \rightarrow \mathsf{ratioE}(r_2)(v_2)$, and the other component has a specification $\mathsf{ratioE}(r_1/c)(v_1)$, and we want the compositional Pareto set to contain all values of v_2, such that there exist values for v_1 that make the specifications achievable (recall that we under-approximate Pareto sets for ratioE MQs using Eratio MQs.) In PRISM-games 2.0, we use named *constants* for this purpose, which specify which dimensions are intersected. An example is the properties file in Figure 7.2. If constant names are not used, a warning is issued that the corresponding dimensions cannot be associated for intersection. An example of a compositional Pareto set is shown in Figure 7.10 in the tool demonstration below.

7.3 Tool Demonstration

We give a brief overview of the usage of PRISM-games 2.0 [86]. More details, download and installation instructions are available at `http://www.prismmodelchecker.org/games/`. We demonstrate the operation of the tool on the running example of Section 3.6, focusing on the multi-objective and compositional features. The following is a step-by-step walkthrough of the tool. All file paths are relative to the installation directory of PRISM-games 2.0.

1. After installing, the PRISM-games 2.0 GUI is started by executing `xprism`, or by navigating to the installation directory and executing `./bin/xprism`.

2. We now load the model of a stochastic game. When selecting from the top menu bar the item *Model→Open model ...*, a dialogue opens, in which we select the file `examples/compositional/comp_example.prism`. The model is given in Figure 7.1, representing the games in Figure 3.8, and Figure 7.7 shows the tool in the *Model* tab.

3. We now load a properties file with the specifications we are interested in, by selecting from the menu bar the item *Properties→Open properties list ...*, and in the dialogue choosing `examples/compositional/comp_example.props`. PRISM-games 2.0 automatically switches to the *Properties* tab, shown in Figure 7.8, displaying the properties we intend to check.

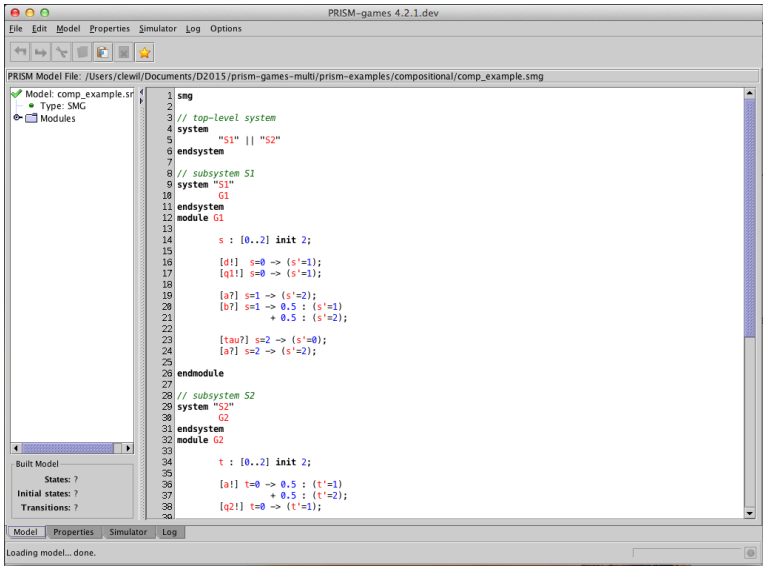


Figure 7.7: Screenshot of PRISM-games 2.0 showing the compositional model.

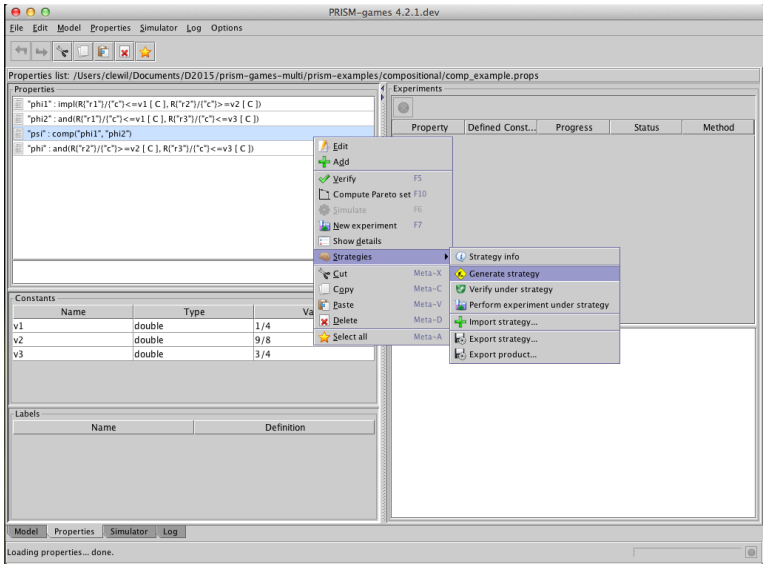


Figure 7.8: Screenshot of PRISM-games 2.0 showing the menus for strategy generation.

4. The properties labelled "`phi1`" and "`phi1`" are the local specifications for the components `S1` and `S2`, and we check them using the compositional property "`psi`". Right-clicking on "`psi`" opens a menu, in which we select *Strategies*→*Generate Strategy*, see Figure 7.8. After PRISM-games 2.0 finishes the computation, a dialogue appears showing the result of (in our case successful) strategy generation.

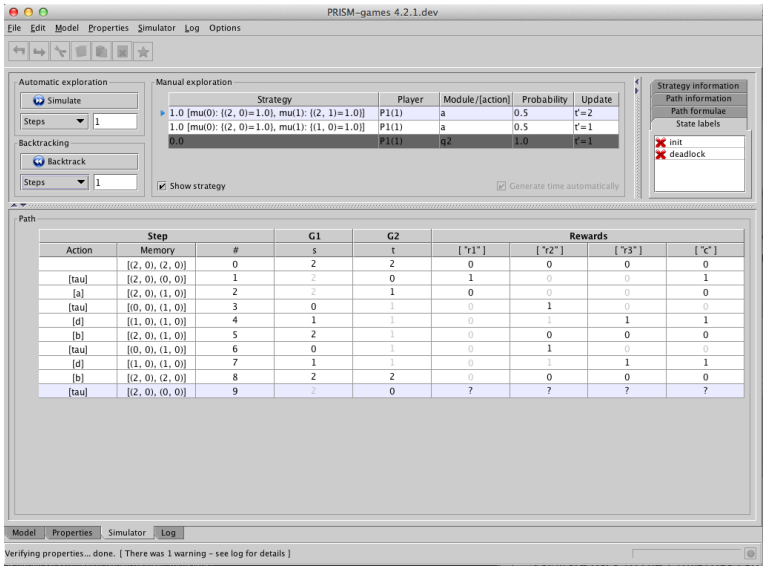


Figure 7.9: Screenshot of PRISM-games 2.0 showing the game simulated under the computed strategy.

5. We can inspect details of the computation by navigating to the *Log* tab, shown at the bottom of the screen. The log also includes the resulting strategies for each component.

6. We now inspect the model under the strategy by simulating it. From the menu bar we select *Simulator*→*New path*, and the *Simulator* tab automatically appears. Figure 7.9 shows the simulation of the composed game for a few steps. In the lower half of the screen, the table shows, in its respective columns: (1) the action of the move chosen by the strategy; (2) the memory of the composed strategy; (3) the step number along the path; (4-5) the state of the component games; and (6-9) the reward in the step (action plus state, but here we work with rewards defined on actions). Further, in the upper half of the screen, the table shows in each row a successor state, and in its respective columns: (1) the memory update of the synthesised strategy, and, if a Player 1 state, the choice; (2) the player controlling the state, and, if a Player 1 state, the game in which Player 1 controls the state; (3) the label of the transition; (4) the probability of the move to lead to the successor; and (5) the update of the variables when transitioning to the successor. Note that rows which the strategy never selects are disabled, because only choices in the reachable state space are computed.

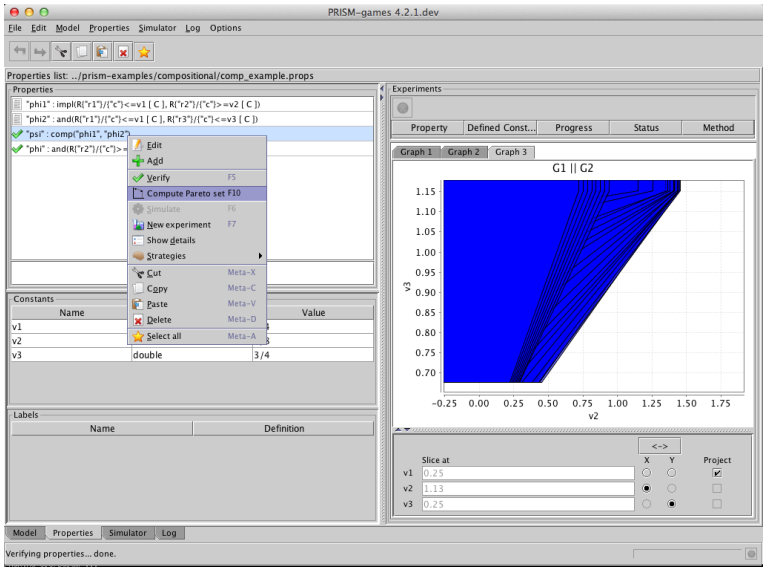


Figure 7.10: Screenshot of PRISM-games 2.0 showing an approximation of the compositional Pareto set.

7. We now synthesise a strategy directly on the composed system. In the properties tab, by right-clicking on the "`phi`" property, a menu opens in which we select *Strategies→Generate Strategy*.

8. We now compute compositional Pareto set approximations. In the properties tab, by right-clicking on "`psi`", a menu opens in which we select *Compute Pareto set*. After completion, three Pareto sets (approximations) are shown on the right of the screen. *Graph 1* and *Graph 2* are for `S1` under "`phi1`" and for `S2` under "`phi2`", respectively. *Graph 3* shows the compositional Pareto set for "`psi`", before projecting it for the global property "`psi`". By default the dimensions `v1` and `v2` are selected. We can select the dimensions `v2` and `v3` using the radio buttons, and checking the box for *Project* associated with `v1`. The resulting Pareto set is the approximation of the compositional Pareto set, shown in Figure 7.10.

9. To compute the Pareto set for the global property directly, we right-click on "`phi`" and select *Compute Pareto set*, to generate a new graph, *Graph 4*, which is an approximation of the global Pareto set for "`phi`".

7.4 Summary

We implemented the assume-guarantee strategy synthesis framework in PRISM-games 2.0. The tool provides a versatile modelling environment for stochastic games composed of multiple components. For monolithic games (that is, component games), our main contributions are threefold: (i) we added support for the analysis of multi-objective queries consisting of Emp, ratioE, Pmp, Pratio and Erew objectives; (ii) Pareto sets are displayed to give an indication of the achievable targets, even for higher dimensions, via slicing and projecting; and (iii) strategies can be synthesised, exported, and inspected using the simulator that also displays the stochastic memory updates. We highlight the following main contributions for the assume-guarantee framework: (i) we extended the modelling language of PRISM-games so that stochastic games can be defined as a composition of several component games, with independent player assignments; (ii) we added the keyword `comp` to the property specification language, in order to specify local properties for the individual components, and invoke the compositional routines; and (iii) in compositional models, Pareto sets can be computed compositionally, to instantiate the local targets in assume-guarantee rules.

Several steps in the strategy synthesis are implemented using numerical operations with finite accuracy. For example, computing the distributions in the strategy construction is performed using linear programming. However, using the (optional) rounding operations, for Erew objectives, at each step we still obtain an under-approximation to the Pareto set, but convergence is slower, while for Pmp objectives, rounding means that, at every iteration of the $F_{M,\vec{r}}$ operator, the sets might be smaller by some $\delta > 0$. For Pmp the sets thus could diverge even if the specification is achievable, but, if convergent, the constructed strategies achieve the desired target.

There are several directions for future development. A key challenge is to improve the performance of the algorithm implementations. Compositional analysis itself addresses some scalability issues, and we use the PPL for efficient polytope computations. However, a symbolic representation of the state space, rather than just the polytopes, could speed up the fixpoint computations. More fundamentally, it would be interesting to investigate other ways of computing ε-optimal strategies, for example via limiting the memory size a-priori and developing a linear program characterisation for the now bounded-memory strategies. Given that MD Player 2 strategies suffice for Pmp, another approach would be to consider counterexample-guided abstraction refinement (CEGAR) [30]. Finally, for the Pareto set computations, alternative approximation algorithms could be investigated, in particular for the ratioE objectives.

Chapter 8

Case Studies

Contents

We demonstrate in this chapter our assume-guarantee strategy synthesis framework on four case studies. The case studies evince the applicability of our methods, and further demonstrate the effectiveness of our tool implementation from Chapter 7. We give two case studies that use the assume-guarantee framework to obtain strategies compositionally and demonstrate scalability. To comprehensively display the synthesis capabilities of our tool, these case studies use multi-objective ratios of rewards, and we give two further case studies with conjunctions of expected total reward objectives. However, care must be taken when employing total rewards in the context of assume-guarantee synthesis under fairness, since they apply mainly to the transient states in a game (see the discussion in Section 3.4.1). In [11] we gave an example with total rewards for our assume-guarantee framework, and noted that this requires all components to synchronise on entering terminal states, since otherwise there are no fair strategies for Player 2 and the global property is trivially satisfied.

Firstly, in Section 8.1, synthesis for multi-dimensional total expected rewards is explored in a case study concerning controller synthesis for autonomous urban vehicles [43]. The scenario, modelled as a stochastic game, is that of a car moving over road segments in a potentially adverse environment, with the objective of maximising safety, while also maximising the likelihood to arrive at the intended goal location

with the highest comfort on the roads. Secondly, in Section 8.2, we consider path planning for unmanned aerial vehicles interacting with a human operator [63]. In this model, we also particularly study the effects of probability and nondeterminism, using the probability of the operator delegating the flight choices to the UAV. Thirdly, in Section 8.3, synthesis for multi-dimensional ratios of rewards in a compositional setting is demonstrated in a case study for controlling the power distribution network on a more electric aircraft [10]. The system is partitioned into several components for increased reliability and scalability, which is exploited by our assume-guarantee strategy synthesis method. Employing quantitative specifications (via rewards) and a model based on stochastic games allows us to achieve improved performance over previous studies [140]. Lastly, in Section 8.4, we present an investigation of a compositional model for temperature control in several rooms, in which controllers have to maintain a given setpoint in the presence of outside temperature uncertainty and potentially changing room dynamics, for example, from people entering the room or windows being opened. The analyses of the case studies were carried out using our PRISM-games 2.0 tool presented in Chapter 7, on a 2.80 GHz x86 CPU with 32 GB of memory. This chapter is mainly based on [10, 43, 63].

8.1 Autonomous Urban Driving

Autonomous driving has the potential to fundamentally transform our transportation infrastructure. We present a case study of autonomous vehicle navigation in an urban setting, which is motivated by the DARPA Urban Challenge 2007 (henceforth referred to as the Challenge), a competition for autonomous cars to navigate safely and effectively in an urban setting [51]. In the Challenge, cars were exposed to a set of traffic situations, and had to plan routes, avoid hazards, and cope with the presence of other vehicles. This section is mainly based on the work in [43].

8.1.1 Problem Setting

We identify desired functions of vehicle controllers from the Challenge, and focus on a scenario of a car driving in an urban context. We model the problem as a two-player stochastic game, where Player 1 represents the car controls, and Player 2 represents the environment. The game setting allows us to model the nondeterministic, adversarial nature of hazards selected by the environment, and the car reacts to the hazards according to the synthesised strategy. The topology of the roads is obtained

Table 8.1: Model parameters for the autonomous driving case study.

Hazard	Abbreviation	Weight (ζ)	Reaction	Accident Probability
pedestrian	p	0.05	brake	0.01
			honk	0.04
			change lane	0.03
jam	j	0.1	honk	0.01
			U-turn	0.02
obstacle	o	0.02	change lane	0.02
			U-Turn	0.02

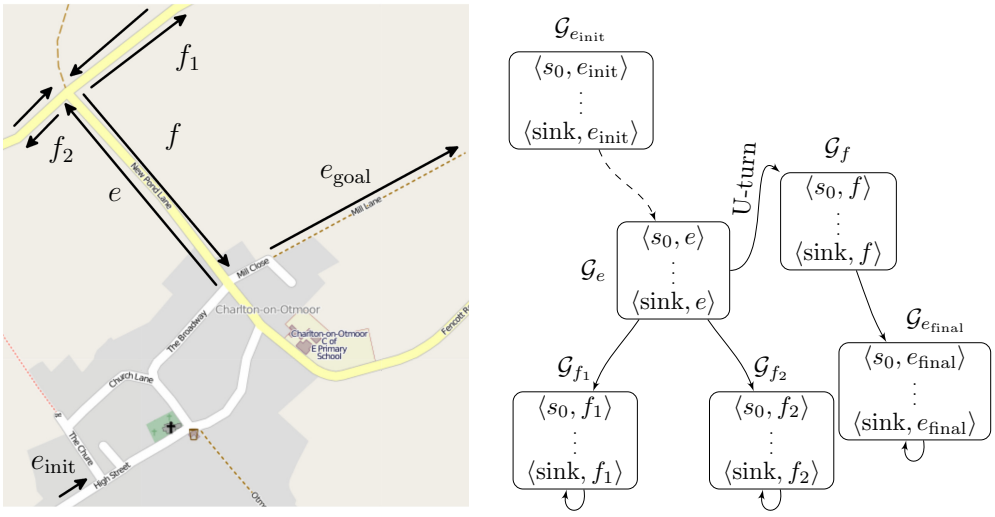
by importing maps from OpenStreetMap [103]. Probabilities are used to model the relative frequency of events in a given road segment and between different road segments. Further, considering multiple objectives enables the exploration of trade-offs when constructing controllers.

Our model is based on the Challenge event guidelines and technical evaluation criteria [51]. Each road segment is modelled as part of a stochastic game between the environment that selects from a set of available hazards, and the car that selects reactions to the hazards, as well as making route selection (steering) decisions. Hazards occur with certain probabilities depending on the properties of the road segment the car is on, and each decision the car takes is only successful with a given probability. We consider three types of hazards with the corresponding reactions the car can take (see Table 8.1): for example, if a pedestrian is on the road, the controller can choose to brake, honk, or change the lane, each with a different probability of an accident occurring. The probabilities in the model are understood to be parameters obtained from statistical observations; for example, certain road types are more prone to accidents. Finally, we model the road quality using rewards.

8.1.2 Model

We represent the road network as a directed graph $G = (V, E)$, where each edge $e \in E$ represents a road segment, see Figure 8.1 (a). To each edge $e \in E$, we associate an *edge-game* $\mathcal{G}_e$, parameterised by the properties of the corresponding road segment (for example, length, quality of the road, number of lanes).

Edge Games. An example illustrating an edge-game $\mathcal{G}_e$ is shown in Figure 8.2, where we use the abbreviations for hazards introduced in Table 8.1. The states of $\mathcal{G}_e$ are of the form $\langle s, e \rangle$, where s can be seen as an offset into the state space of $\mathcal{G}_e$; when the edge e is clear from context, we simply write s. For each $\mathcal{G}_e$, in state s_0, a set of



(a) Map of Charlton-on-Otmoor, UK, with overlaid graph of the road network (only partially shown).

(b) Connection of edge-games. Roads that are not one-way have U-turn connections (only one U-turn is shown for the sake of clarity). The dashed arrow abstracts several intermediate edge-games.

Figure 8.1: Illustrating the graph $G = (V, E)$ representing the road network and the corresponding edge-game connections.

at most two hazards is selected probabilistically, from which Player 2 can then choose. We put the distribution over hazards before the Player 2 choice, since otherwise it is always possible to select a pedestrian hazard, even if there may be no pedestrian present on the current road segment. To each hazard $h \in \{\mathsf{p}, \mathsf{j}, \mathsf{o}\}$ we associate a tuning parameter ζ_h, given in Table 8.1. We model that hazards are more likely the longer the road segment e, and we use $\text{len}(e)$ for the length in metres of e. The weights ζ_h tune the relative frequency of hazard occurrence. To achieve a valid probability distribution, we use the hyperbolic tangent as a sigmoid function. Thus, we define the probability of a set of hazards $\{h_1, h_2\}$ by $p_{h_1 h_2} \stackrel{\text{def}}{=} \tanh(\zeta_{h_1} \cdot \zeta_{h_2} \cdot \text{len}(e))/K$, and of a single hazard h by $p_h \stackrel{\text{def}}{=} \tanh(\zeta_h \cdot \text{len}(e))/K$, where $K = 6$ is the number of nonempty sets of at least two hazards. Finally, the empty set of hazards is chosen with the residual probability, denoted p_{none}. Once a set of possible hazards is chosen in s_0, and Player 2 has selected a specific one in s_1, s_2 and s_3, Player 1 must select an appropriate reaction in s_4, s_5 and s_6. Then the game enters either the terminal accident state "acc", or a Player 1 state "sink", where the next edge can be chosen. If the reaction is not appropriate in the given road segment (for instance, changing

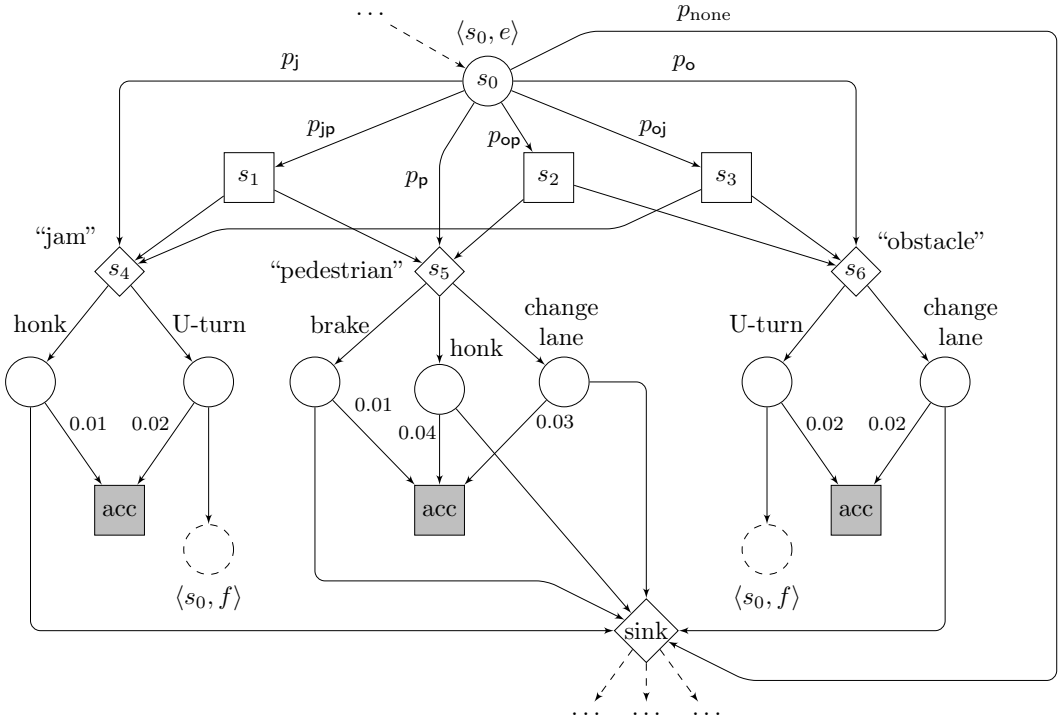


Figure 8.2: Edge-game $\mathcal{G}_e$ with reverse edge f of e. States of the form $\langle s, e\rangle$ are abbreviated by s. For simplicity, we omit stochastic states with Dirac distributions. Hazards and reactions are indicated using notation from Table 8.1. After a U-turn, the initial state $\langle s_0, f\rangle$ of the reverse edge gadget $\mathcal{G}_f$ is entered (note that this state is not part of $\mathcal{G}_e$, and shown twice here for clarity of the diagram.)

lane in a single lane road), a terminal "violation" is entered instead with probability 1 (not shown in Figure 8.2), which the car is assumed to naturally try to avoid.

Connecting Games. From the edge-games $\mathcal{G}_e$ and the graph G, a game $\mathcal{G}$ is constructed that connects the games $\mathcal{G}_e$ as shown in Figure 8.1 (b). In a sink, for instance, $\langle \text{sink}, e\rangle$, Player 1 chooses which edge to go to next and enters the corresponding local initial state, for example, $\langle s_0, f_1\rangle$. Also, in a U-turn, the edge-game $\mathcal{G}_f$ of the reverse edge f of e is entered at its initial state, for instance, $\langle s_0, f\rangle$. If a road segment is the goal e_{goal}, or does not have any successors in G, the local sink for the corresponding game is made into a terminal state. Thus, $\mathsf{Term} = \{\langle \text{sink}, e_{\text{goal}}\rangle\} \cup \{\langle \text{sink}, e\rangle \mid \nexists v'' \in V \,.\, e = (v, v') \wedge (v', v'') \in E\} \cup \{\langle \text{acc}, e\rangle \mid e \in E\} \cup \{\langle \text{violation}, e\rangle \mid e \in E\}$. Note that the above construction results in a stopping game.

8.1.3 Analysis

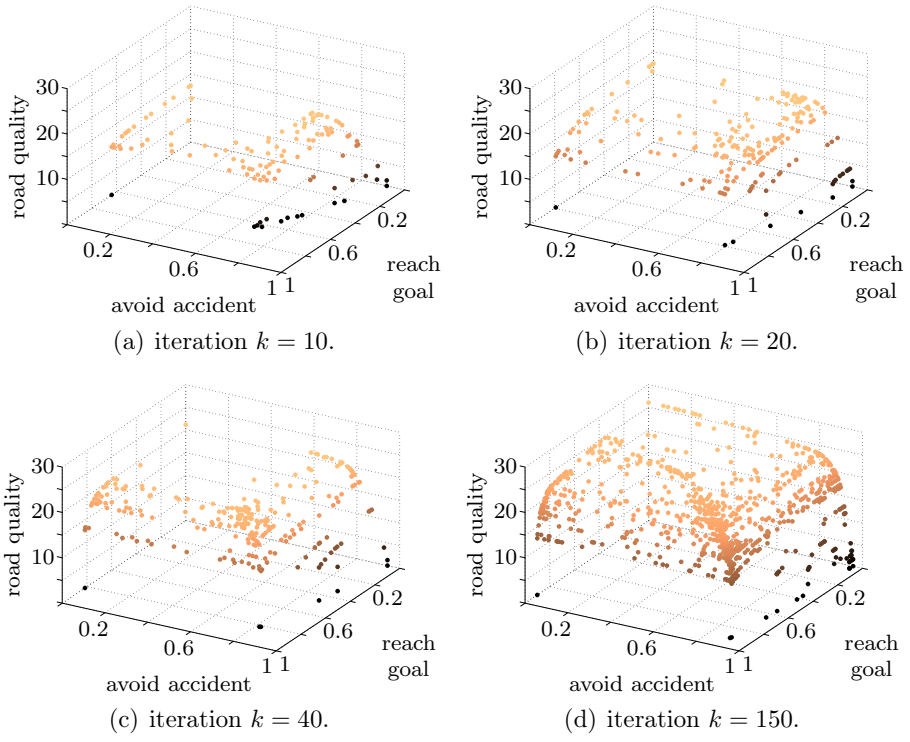
We study three objectives, derived from the requirements of the Challenge. We fix the initial state to be $\langle s_0, e_{\text{init}} \rangle$, and consider the following objectives, where $\mathsf{Reach}(T)(v)$ stands for the *reachability objective* defined as $\mathbb{P}_{\mathcal{D}}(\mathsf{F}\, T) \geq v$.

Target Location Reachability. From the initial location, the car has to reach a target location at a particular orientation with probability v_{tlr}, that is, we have the objective $O_{\text{tlr}} \equiv \mathsf{Reach}(T_{\text{tlr}})(v_{\text{tlr}})$, where $T_{\text{tlr}} \stackrel{\text{def}}{=} \{\langle \text{sink}, e_{\text{goal}} \rangle\}$ is the set of targets corresponding to reaching the goal on the edge e_{goal}. Note that the orientation of the car is implicit, as two-way streets are modelled as separate edges. On a high level, reachability at a correct orientation is a primary goal also in the Challenge.

Accident Avoidance. Further, the car has to avoid accidents, which we encode by requiring at least a probability v_{aa} of reaching terminal states where no accident happened, that is, $T_{\text{aa}} \stackrel{\text{def}}{=} \mathsf{Term} \setminus \{\langle \text{acc}, e \rangle \mid e \in E\}$. Note that a traffic rule violation is not considered an accident, as it is not safety-critical. Hence, we have the objective $O_{\text{aa}} \equiv \mathsf{Reach}(T_{\text{aa}})(v_{\text{aa}})$. This safety objective is another primary goal of the Challenge.

Road Quality. Finally, we require the car to use roads with some minimum quality requirements, that is, we have the objective $O_{\text{rq}} \equiv \mathsf{Erew}(r_{\text{rq}})(v_{\text{rq}})$, where r_{rq} is the reward structure corresponding to road quality, and v_{rq} is the required threshold. We define r_{rq} according to the road type and length extracted from the map data. Hence, each edge e is assigned a value $\text{rval}(e)$, and the reward function r_{rq} is defined by $r_{\text{rq}}(\langle \text{sink}, e \rangle) \stackrel{\text{def}}{=} \text{rval}(e) \cdot \text{len}(e)$, and zero otherwise. Likewise, in the Challenge, cars must be able to navigate over different road types, and select adequate roads.

Fixpoint Computation. We consider the conjunctive query $\varphi = O_{\text{tlr}} \wedge O_{\text{aa}} \wedge O_{\text{rq}}$, and explain the fixpoint computation. In Figure 8.3 we show the rounded polytopes $Y^k = F^k_{\mathsf{rew},\vec{r}}(\bot)$ computed for the initial state of the game for Charlton-on-Otmoor for several values of k. To decrease the number of extreme points we have to consider, we use rounding and the following dynamic accuracy adaptation. Starting from the baseline accuracy $a_0 = 50$, we dynamically increase the accuracy a_k by the factor $\eta = 1.2$ after N_k steps, while at the same time multiplying the number N_k of steps until the next increase by η, starting at $N_0 = 5$. In the long run, this yields an additive increase in the accuracy of $\frac{a_0(\lambda-1)}{N_0}$ per step. With this approach, we obtain under-approximations of the Pareto frontiers with a small number of points that are gradually refined by allowing more points in the polytopes. Further, we use Gauss-Seidel updates, which increase the convergence speed but not the time per iteration. In Figure 8.4 we show performance indicators of the value iteration of Figure 8.3.



(a) iteration $k = 10$. (b) iteration $k = 20$.

(c) iteration $k = 40$. (d) iteration $k = 150$.

Figure 8.3: Successive (under-)approximations of the Pareto sets in $\langle s_0, e_{\text{init}} \rangle$ of $\mathcal{G}$ for Charlton-on-Otmoor, UK (figure adapted from [43]).

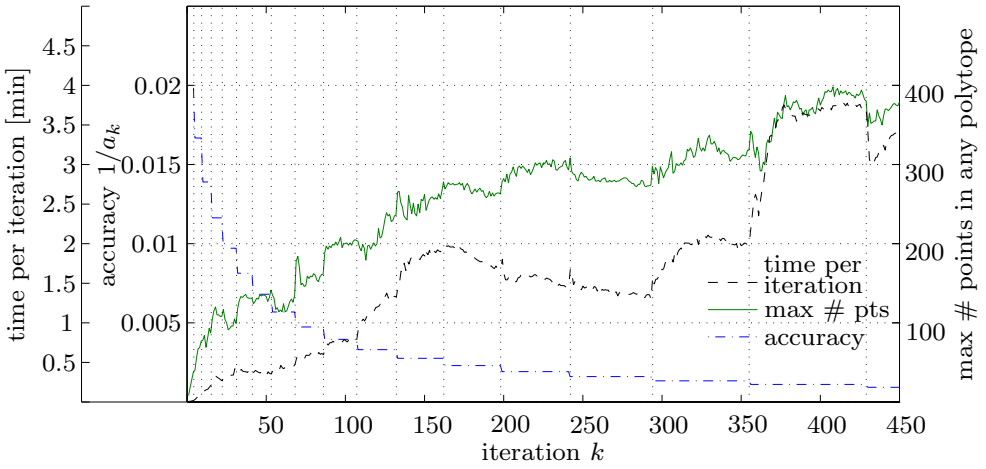
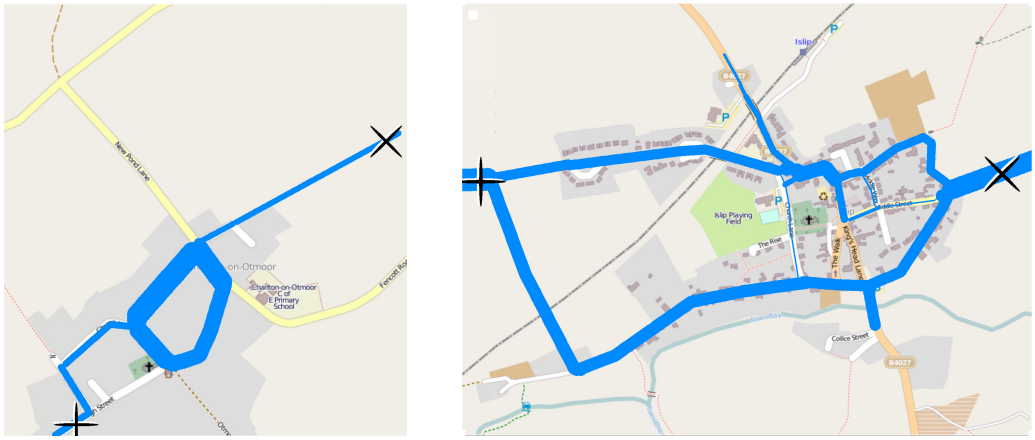


Figure 8.4: Performance indicators for the fixpoint computation using $F_{\mathsf{rew},\vec{r}}$, on the example of Charlton-on-Otmoor. Note the changes in the number of points and time to complete an iteration as the accuracy changes (figure adapted from [43]).

Strategy Evaluation. We perform strategy synthesis for an example choice of the target vector, $\vec{v} = (0.7, 0.7, 6.0)$, for our objective φ, applied to two small villages



(a) Charlton-on-Otmoor: 43 edges in G, 501 states in $\mathcal{G}$.

(b) Islip: 125 edges in G, 1527 states in $\mathcal{G}$.

Figure 8.5: Simulation results under synthesised Player 1 strategy and uniform Player 2. The start and the goal are shown by a plus (+) and a cross (×) respectively. The thickness of the lines represents the expected proportion of trip time spent on the respective road.

in the UK. We evaluate the constructed strategies π by simulating it in the respective games together with an environment strategy σ that picks hazards uniformly at random. We illustrate the simulation results in Figure 8.5. The figure is generated by simulating 10000 paths in the induced DTMC, and for each path extracting the sequence of edges in the graph G the car goes through. Each such sequence $\rho = e_0 e_1 e_2 \ldots$ has an attached probability p_ρ from the simulation, and we then evaluate the proportion of the time the car spends in each edge e via $t_e = \sum\{p_\rho \mid e \in \rho\}$. For each edge e, the thickness of the corresponding line in Figure 8.5 is proportional to t_e. Note that, in Figure 8.5 (b), roads are picked whose distance to the goal is not the shortest, which is because the strategy achieves a trade-off between the objectives: since maximising road quality is traded off against other goals, it may be beneficial to take roads with higher quality to improve the expectation, while at the same time possibly incurring accidents and traffic rule violations with higher probability.

8.1.4 Discussion

To implement a strategy as a controller on an autonomous vehicle, we have to ensure that the strategy state is consistent with the system state. The strategy's view of the system is determined by the game model, which can be thought of as simulating the game inside the strategy. To update the state, the strategy needs to measure the system state and trigger the corresponding transitions, both in Player 2 and stochastic

states. In Player 1 states, the transitions selected by the strategy are interpreted as changing the actuator settings of the car to control the system. The strategies that we synthesise have deterministic choice functions π_{c}, while the randomisation is part of the memory update function π_{u}. Therefore, a controller based on such a strategy can choose specific actions, rather than distributions, and yet realise a randomised strategy achieving the desired objectives.

The total time to compute the strategy for Charlton-on-Otmooor (CoO) is 2627s, while the time for Islip is only 1934s, even though the state space for the Islip model is more than three times the size of that for CoO. However, the total time is influenced by the concrete topology of the respective road graphs. Thus, the synthesis for CoO took 111 iterations with 133 extreme points at the latest iteration, while for Islip synthesis took only 87 iterations with 114 extreme points. Larger villages in experiments resulted in significantly higher computation times.

Possible Extensions. Scalability could be increased by employing domain-specific techniques: for example, we could consider a receding-horizon control approach, where only a finite horizon is considered in strategy synthesis, which is extended step by step as the game advances, see for example [137]. We could thus synthesise a strategy that only considers the roads up to a given distance. However, with receding-horizon control global guarantees have to be ensured even if making local planning steps.

Further, with human drivers, there exists a "principle of trust" with other traffic participants (such as other cars and pedestrians), whom driver can assume to behave reasonably. For example, other cars obey the traffic rules most of the time, and pedestrians usually use side- and crosswalks. This trust relationship could be cast as queries of the form $\varphi^A \to \varphi$, where φ^A expresses the assumptions on the uncontrollable environment. This is in contrast to our compositional framework, where φ^A is satisfied by another component.

Finally, one could augment the user interface to allow more interactive use: in addition to selecting the desired target location for the reachability objective O_{trl}, and the desired trade-off from the Pareto set as in step (S3) of our synthesis procedure of Section 4.4, the visualisation of the resulting strategy could be supplemented with more details, while maintaining clarity and intuitive feedback to a non-technical user.

Availability. The related files are in the `games/car` subfolder of the examples directory in the PRISM-games 2.0 release. To run the computations, the model and properties file can be generated using `generate.py` (reproduced in Appendix B.1), which also produces a bash script to run the synthesis.

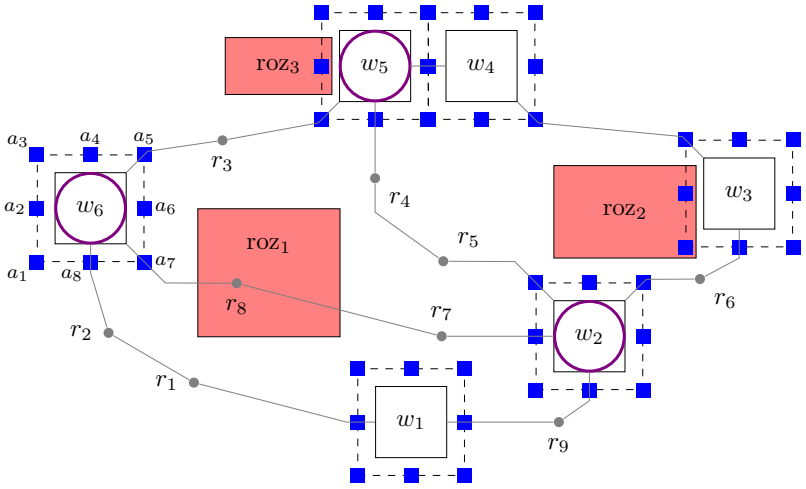


Figure 8.6: A road network for UAV ISR missions (adapted from [77]). Anglepoints are shown as small (blue) rectangles, roadpoints are shown as grey circles between waypoints, ROZs are shown as (red) rectangles, and checkpoints are indicated by circles.

8.2 UAV Path Planning

Control of unmanned vehicles has received increased attention in recent years, with versatile applications in both the civilian and military domains. We study the control of an unmanned aerial vehicle (UAV) in a road network surveillance scenario, where some decisions are made by the UAV in full autonomy, while other decisions are dictated by a human operator, for example, via a remote wireless link. This case study is based on [63], where analysis based on MDPs and stochastic games was performed, and we focus in this section on the game model.

8.2.1 Problem Setting

We consider the scenario of a UAV performing intelligence, surveillance, and reconnaissance (ISR) missions over a road network. Figure 8.6 shows a map of the road network, which has six surveillance *waypoints* labelled $w_1, w_2, \ldots, w_6$. To take pictures from different angles at a waypoint, we discretise angles of approach in increments of 45°, obtaining eight *anglepoints* $a_1, a_2, \ldots, a_8$ around each waypoint. Roads connecting waypoints are discretised into *roadpoints* $r_1, r_2, \ldots, r_9$. Red polygons in Figure 8.6 represent *restricted operating zones* (ROZs), in which flying the UAV may be dangerous or compromise stealth requirements.

We assume the UAV is equipped with the necessary low-level controls to transition between waypoints and roadpoints, and perform surveillance manoeuvres, such as loitering above waypoints. The UAV performs flying tasks autonomously, and interacts with a human operator that may give additional input to influence the planning decisions. The human operator is primarily in charge of the sensor tasks, for example selecting the anglepoints of approach. Moreover, the operator can decide to influence the UAVs steering decisions at designated *checkpoints* w_2, w_5 and w_6, marked by circles in Figure 8.6. The piloting plan for the UAV has to satisfy a high-level mission specification [77]. We focus on surveillance of the road network with minimum fuel consumption, while trying to avoid ROZs. We synthesise strategies which can be implemented on the UAV's onboard automation interface.

8.2.2 Model

We model the UAV controller synthesis scenario as a stochastic game. Since we want to find a flying plan for the UAV, we model the UAV as Player 1, and the operator as Player 2. Figure 8.7 shows a fragment of our game model, in which states controlled by the UAV and operator are drawn in circles and boxes, respectively. The probability distributions in the model correspond to choices where statistics about the operator behaviour are assumed to be known. Player 2 nondeterminism for the operator either models that we cannot quantify distributions, or that we want to maintain the ability of the operator to alter the mission dynamically to address previously unforeseen circumstances. In addition to using Player 2 to model the operator, we could model Player 2 nondeterminism of truly adversarial nature, for example, if the UAV has to avoid being detected by an intelligent adversary.

UAV Model. The dotted boxes in Figure 8.7 correspond to fragments of the UAV model. The UAV moves on roadpoints between waypoints, where the operator performs sensor tasks, potentially requiring the UAV to loiter until exiting the waypoint. At roadpoints and waypoints that are not checkpoints, the UAV selects where to fly next, for example, referring to Figure 8.6, at w_3, the UAV can go to r_6 or w_4 (note that the available choices do not depend on the anglepoint). At checkpoints, the operator has some choice over the flying path of the UAV. If the operator has complete power at checkpoints (modelled as a nondeterministic choice of Player 2), the game semantics allows the operator to behave completely adversarially in the worst case. For example, if the mission requires to visit w_3, the operator could ask the UAV to fly in the loop $w_2, w_6, w_5, w_2, w_6, \ldots$ forever, so that the UAV never completes its mis-

sion. To avoid such behaviours, we define the *delegation probability* p_{del}, with which the operator delegates to the UAV the task of picking the next road. The delegation probability is not specific to any particular map or mission, and thus can be quantified by obtaining statistics on how often an operator delegates to the UAV automation during training or past missions.

Operator Model. Our model of the human operator takes into account the behaviour and performance of the operator, based, for example, on high-level trends induced from human factors literature, see [63] for a discussion. The following explanation refers to the dashed boxes in Figure 8.7, corresponding to the operator model. The operator performs image analysis tasks, which we count using k. Incrementing k is indicated by $k{+}{+}$, and the operator reaches *fatigue* if k reaches a threshold T. The *workload* of the operator is either high or low, with uniform distribution. Analysis of the images depends on the *proficiency* of the operator, which depends on k and the workload: under low workload, images are *good* with probability $p_l(k) = p_l(0)$ if $k < T$ and $p_l(k) = f \cdot p_l(0)$ if $k \geq T$, and we define $p_h(k)$ symmetrically for high workload, where $p_l(0)$ and $p_h(0)$ are the initial accuracies, and $f < 1$ is a *fatigue discount factor*. If the image is *bad* then the operator asks the UAV to *loiter* at the waypoint to collect more data. If the image is *good* the operator exits the waypoint, and enters a flying phase. In waypoints that are not checkpoints, the UAV then takes over, and at checkpoints the operator can delegate or prescribe the next flying target.

8.2.3 Analysis

We now analyse our model under an example UAV mission of covering the waypoints w_1, w_2, and w_6 (starting at w_1) in minimum time. We are particularly interested in the trade-offs between the likelihood of visiting ROZs and the mission completion time. Assume each loiter takes 10 time units and flying each road segment takes 60 time units, that is, we define a reward structure time(loiter) = 10 and time(fly) = 60. We also define the reward structure ROZ(s) = 1 for all fly-transitions that are outgoing from states corresponding to positions in ROZs (see Figure 8.6). To encode the mission objective, we introduce self-loops with zero rewards at the states w_1, w_2 and w_6 of the game model to make them terminal states. This does not make the game stopping, as the UAV may still cycle between, for example, w_4 and w_5, but, since we are interested in minimising the mission completion time, the UAV ($\mathsf{Player\ 1}$) has no incentive to choose this strategy. Formally, we consider the objectives $\mathsf{Erew}^{\leq}$(time) and $\mathsf{Erew}^{\leq}$(ROZ), and we analyse the qualitative trends for the objectives.

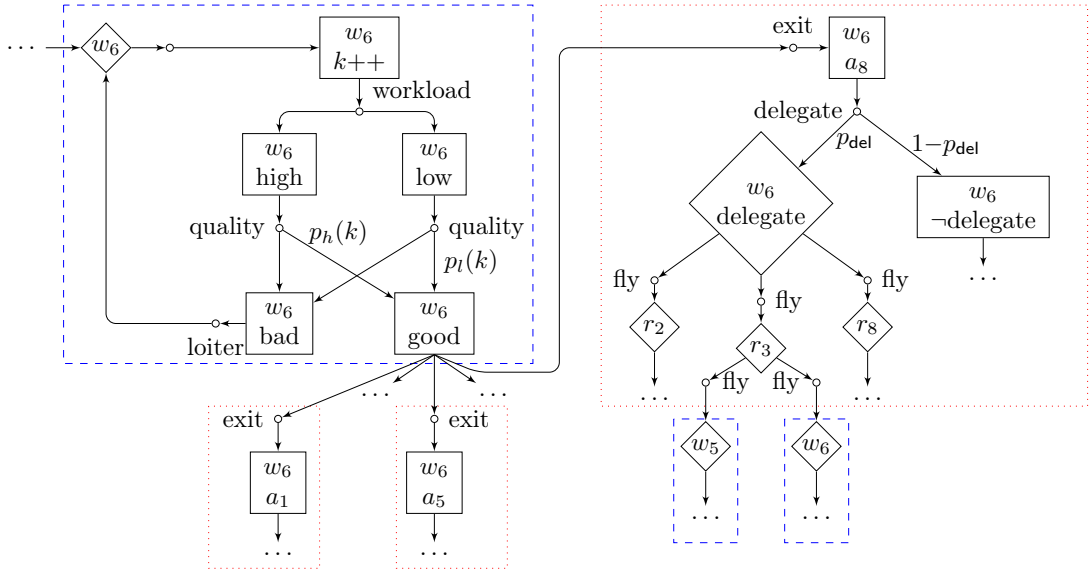


Figure 8.7: A fragment of the game between the operator (Player 2) and UAV (Player 1) at waypoint w_6. Ellipses indicate where we cut off the game for the sake of this presentation. Since w_6 is a checkpoint, the operator can delegate. Dashed and dotted rectangles, respectively, indicate the operator and UAV dynamics.

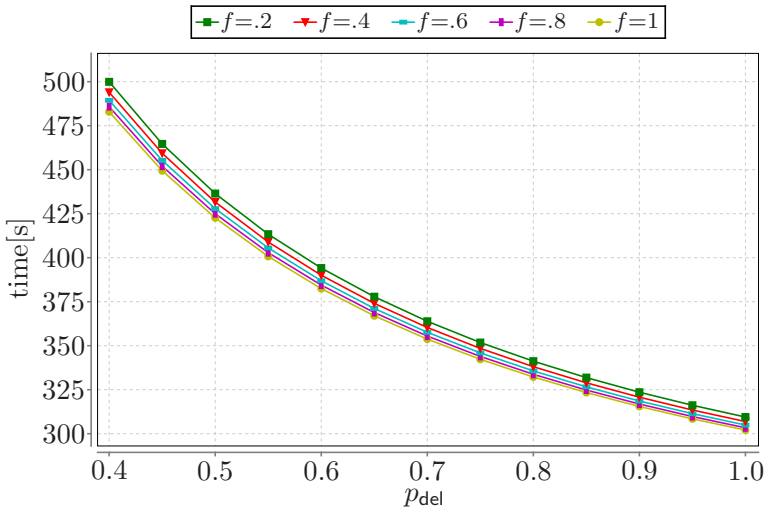


Figure 8.8: Expected mission completion time as a function of delegation probability p_{del} and for various fatigue discount factors f. We fix $T = 0$, $p_l(0) = 0.9$ and $p_h(0) = 0.8$.

Delegation Probability. We first analyse the effect of the delegation probability p_{del} on the achievable expected mission completion time, v_{time}. Figure 8.8 shows a plot of v_{time} as a function of p_{del}. The higher the delegation probability (that is, the less operator intervention), the faster the mission can be completed. If we, moreover, vary

the operator fatigue discount factor f, we observe that v_{time} increases with fatigue, that is, with decreasing f. In our current model the operator fatigue only affects the UAV loitering behaviour, and is independent of p_{del}, which is why the curves for different values of f are parallel to each other.

Trade-offs. We first consider the query of minimising the expected time of mission completion, and simultaneously minimising the expected number of ROZ visits, v_{ROZ}, that is, we consider the CQ $\mathsf{Erew}^{\leq}(\text{time})(v_{\text{time}}) \wedge \mathsf{Erew}^{\leq}(\text{ROZ})(v_{\text{ROZ}})$. We investigate the achievable trade-offs by computing approximations of the Pareto sets, using the $F_{\mathsf{rew},\vec{r}}$ operator; we let $a_0 = 100$, and $\eta = 1$, and terminate after $k_{\max} = 100$ iterations (when increasing $k_{\max}$ beyond 100 we observe no significant change in the resulting sets). Firstly, we show the effect of the operator accuracy parameters $p_l(0)$ and $p_h(0)$ in Figure 8.9. While we observe that higher operator accuracy leads to less time needed to complete the mission (because of less loitering), the UAV performance is robust to within at most 10% deviation in mission completion time (for the fixed parameters $p_{\mathsf{del}} = 0.5$, $T = 10$ and $f = 0.7$). Secondly, we investigate the effect of the delegation probability p_{del} in Figure 8.10, where we fix, for the sake of illustration, $p_l(0) = 0.9$, $p_h(0) = 0.8$, $T = 2$ and $f = 0.7$. The trends show that, as the delegation probability decreases, longer missions result from mitigating the chances of entering ROZs. The sensitivity of the UAV performance to the delegation probability is higher than to the accuracy parameters $p_l(0)$ and $p_h(0)$. This suggests that, in the design of UAVs interacting with human operators, a key aspect is deciding under which circumstances an operator can outright delegate the flying plan, while latent mission objectives exist for the UAV.

8.2.4 Discussion

We studied the interaction of a UAV with a human operator, and the latter's influence on the mission performance. The operator has some freedom to steer the UAV, and the UAV has to achieve the mission under all possible operator behaviours. We model human traits of the operator such as fatigue and proficiency, and the optimal flying plans of the UAV can depend on these characteristics. If models for individual operators are available, we may even synthesise *individualised* optimal UAV piloting plans. Further, by separating the operator and the UAV model, the points of interaction become apparent. We can, for example, model that the operator tends to delegate to the UAV automation more often under higher workload or fatigue levels.

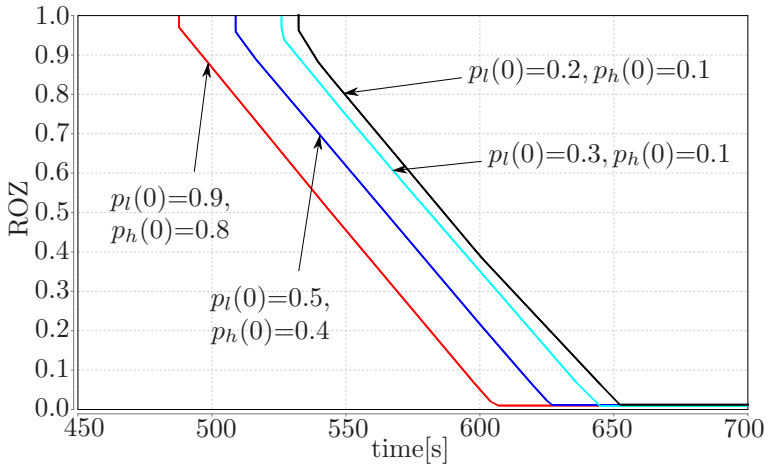


Figure 8.9: Trade-offs under varying operator accuracy function parameters. We set $p_{\mathsf{del}} = 0.5$, $T = 2$ and $f = 0.7$.

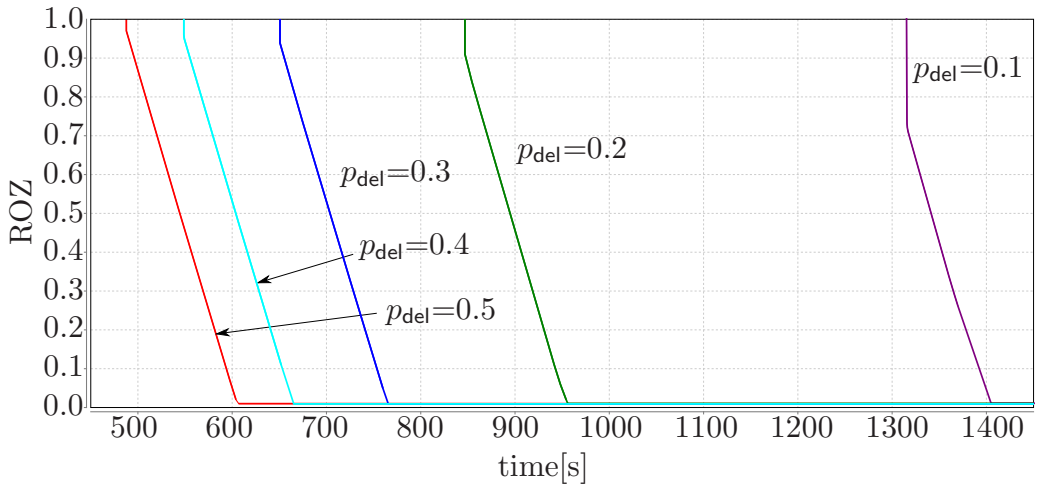


Figure 8.10: Trade-offs under varying delegation probability p_{del}. We set $p_l(0) = 0.9$, $p_h(0) = 0.8$, $T = 2$ and $f = 0.7$.

Possible Extensions. In order to achieve better UAV performance, we can strengthen the assumptions on the operator. Consider a (hypothetical) checkpoint $\tilde{c}$ where the operator can choose waypoints w_x, w_y, and w_z. We call a convex set of distributions over probabilities to go from $\tilde{c}$ to one of $\{w_x, w_y, w_z\}$ an *admissible operating region* (AOR). A larger AOR represents a weaker assumption on the operator behaviour, resulting in lower performance of UAV flying plans, but increased robustness. The fully nondeterministic model, shown in Figure 8.11 (left), allows the operator to pick any distribution with a randomised strategy, and the corresponding AOR is represented by the green triangle in Figure 8.11 (right). If we know about the possible distribu-

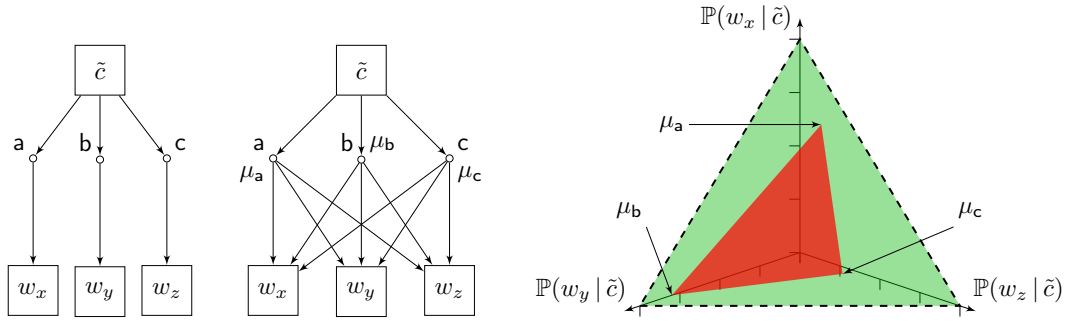


Figure 8.11: The AOR of the operator modelled fully nondeterministic (left) is endowed with learned distributions p_i (middle), each of which corresponds to a corner of the refined AOR (right).

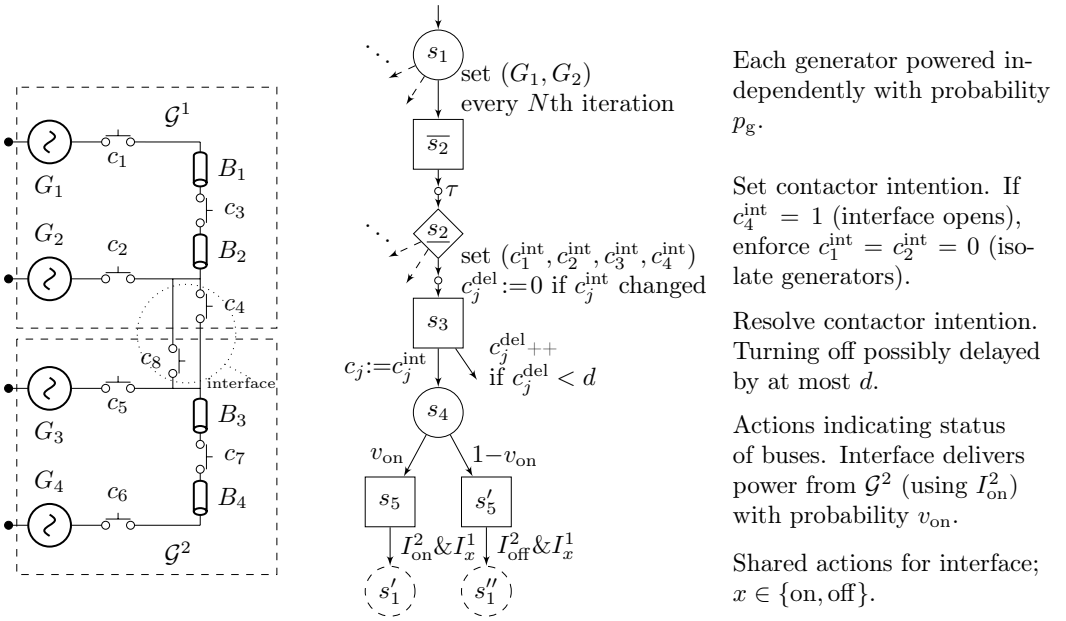
tions of the operator's choices, we may refine the model and obtain, for example, the one shown in Figure 8.11 (middle), where μ_a for $a \in \{\mathsf{a}, \mathsf{b}, \mathsf{c}\}$ represent probability distributions (for instance choosing w_x with probability $\mu_a(w_x)$); the corresponding AOR is drawn as the red triangle in Figure 8.11 (right). Note that, if we synthesise a UAV (Player 1) plan in a PA model (with no Player 2), all distributions for the operator are fixed, and hence represent an AOR consisting of just a single point.

As in the autonomous driving case study, we can synthesise specifications of the form $\varphi^A \to \varphi^G$, where φ^A represents assumptions on the operator and φ^G represents guarantees on UAV mission performance. Hence, instead of explicitly encoding assumptions on the operator's behaviour via concrete probability distributions in the model, we can use implicit assumptions expressed in the specifications.

Availability. The related files are in the `games/uav` subfolder of the examples directory in the PRISM-games 2.0 release. We reproduce the model and properties files (`uav.prism` and `uav.props`, respectively) in Appendix B.2.

8.3 Aircraft Power Control

In avionics, advances in electronics technology have initiated a recent transition to more-electric aircraft in order to, among other benefits, reduce take-off weight and optimise power consumption in-flight. We investigate the assume-guarantee strategy synthesis to control the electric power distribution on an aircraft. Previous studies of the problem appeared, for example in [140], which is concerned with synthesis from (non-quantitative) LTL specifications, and in [102], where the controller is synthesised



(a) Single-line diagram of two AC components.

(b) Game model $\mathcal{G}^1$. The stages are explained on the right hand side.

Figure 8.12: Aircraft electric power system (adapted from [97]).

under a fixed topology of routing power through the aircraft. This section is mainly based on our work in [10].

8.3.1 Problem Setting

We model the electrical power system of a more-electric aircraft according to the Honeywell, Inc. patent [97]. Power is to be routed from generators to buses (and loads attached to them, not shown in the figure) by controlling the contactors (that is, controllable switches) connecting the network nodes. The single-line diagram of the full power system in Figure 8.12 shows how power from the generators (G_i) can be routed to the buses (B_i) through the contactors (c_i). The controllers have to take into account that the contactors may have a delayed response, and that the generators available in the system may be reconfigured, or even exhibit failures. The primary control objectives are to ensure that the buses are powered, while avoiding unsafe configurations due to short-circuits between power sources.

We extend the approaches of [102, 140] to the compositional stochastic games setting with multiple objectives, where the behaviour of generators is described via probability distributions. We demonstrate how our approach yields controllers that

ensure given reliability levels and higher uptimes than those reported in [140], while maintaining the flexibility to decide how power is routed between components, albeit for a smaller model than in [102].

8.3.2 Model

The system comprises several components, each consisting of buses and generators, and we consider the high-voltage AC subsystem, shown in Figure 8.12 (a), where the dashed boxes represent the components set out in [97]. Since the aircraft is to be controlled continually, we use long-run properties to specify correctness. The game models and control objectives in [140] are encoded using linear temporal logic (LTL) properties. We extend their models to stochastic games with quantitative specifications, where the contactors are controlled by Player 1 and the contactor dynamics and the interfaces are controlled by Player 2. The advantage of stochasticity is that the reliability specifications desired in [140] can be faithfully encoded. Further, games allow us to model truly adversarial behaviour (for example, uncontrollable contactor dynamics), as well as nondeterministic interleaving in the composition. The individual components are physically separated for reliability, and hence allow limited interaction and communication. We model an interface to communicate the status between components, and compose them by means of our game composition of Definition 4.3.

Repeated Multi-Stage Game Model. We model each component as an infinite loop of Player 2 and Player 1 actions. The model of one AC subsystem, $\mathcal{G}^1$, is shown in Figure 8.12 (b); $\mathcal{G}^2$ is symmetric. One iteration of the loop represents one unit of time Δ_{d}, and the system steps through several stages, corresponding to the states s_1 to s_5 in $\mathcal{G}^1$, and symmetrically in $\mathcal{G}^2$. In s_1 the status of the generators is set; in $\underline{s_2}$ the controller sets the contactors; in s_3 the delay is chosen nondeterministically; in s_4 actions specify whether both buses are powered, and whether a failure occurs; and in s_5 information is transmitted over the interface. The τ-labelled Dirac transitions precede all Player 1 states to enable composition [11]. The loop then starts again at s_1, potentially with some variables changed, indicated as s_1' or s_1'' in Figure 8.12 (b).

Contactors, Buses and Generators. We derive the models based on the LTL description of [140]: the status of the buses and generators are kept in Boolean variables $B_1, \ldots, B_4$ and $G_1, \ldots, G_4$, respectively, and their truth value represents whether the bus or generator is powered; the contactor status is kept in Boolean variables $c_1, \ldots, c_8$, and their truth value represents if the corresponding contactor lets the current flow. For instance, if in $\mathcal{G}^1$ the generator G_1 is on but G_2 is off, the

controller needs to switch the contactors c_1 and c_3 on, in order to power both buses B_1 and B_2. At the same time, short circuits from connecting generators to each other must be avoided, for example, contactors c_1, c_2 and c_3 cannot be on at the same time, as this configuration connects G_1 and G_2. The contactors are, for example, solid state power controllers [125], which typically have non-negligible reaction times with respect to the times the buses should be powered. Hence, as in [140], we model that Player 1 can only set the *intent* c_i^{int} of contactor i, and only after some delay is the contactor status c_i set to this intent. For the purposes of this demonstration, we only model a delayed turn-off time, d, as it is typically larger than the turn-on time (for example 40 ms, the turn-off time reported in [57]). Whether or not a contactor is delayed is controlled by Player 2.

Interface. The components can deliver power to each other via an interface, shown Figure 8.12 (a), which is bidirectional, that is, power can flow both ways. The original design in [97] does not include connector c_8, and so c_4 has to ensure that no short circuits occur over the interface: if B_3 is powered, c_4 may only connect if B_2 is unpowered, and vice versa; hence, c_4 can only be on if both B_2 and B_3 are unpowered. By adding c_8, we break this cyclic dependence. Actions shared between components model transmission of power. The actions I_x^1 and I_y^2 for $x, y \in \{\text{on}, \text{off}\}$ model whether power is delivered via the interface from $\mathcal{G}^1$ or $\mathcal{G}^2$, respectively. Hence, power flows from $\mathcal{G}^1$ to $\mathcal{G}^2$ via c_8, and the other direction via c_4; the contactors can also ensure that power cannot flow in the other direction, preventing short circuits.

Generator Assumptions. We assume that the generator status remains the same for N time steps, that is, after 0, N, $2N$, ... steps (of duration Δ_{d} each) the status may change, with the generators each powered with probability p_{g}, independently from each other. N and p_{g} are understood to be able to be obtained from the mean-time-to-failure of the generators. These quantitative aspects of the model are in contrast to [140], where, due to non-probabilistic modelling, the strongest assumption is that generators do not fail at the same time.

8.3.3 Analysis

The main objective is to maximise uptime of the buses, while avoiding failures due to short circuits, as in [140]. Hence, the controller has to react to the generator status, and cannot just leave all contactors connected. The properties are specified as ratio rewards, since we are interested in the proportion of time the buses are powered. To use the (Conj) rule from Section 4.2.3, we attach all rewards to the status actions or

Table 8.2: Strategy synthesis for aircraft case study. A minus (−) for v_{on} means the interface is not used, and for k means that no value is applicable. The timeout (TO) is 4 h. The forward slash (/) separates values for the two components.

Target			Model Properties				Parameters		Performance	
v_{b}	v_{f}	v_{on}	N	d	p_g	$\lvert S\rvert$	ε	a_0	k	Runtime [s]
0.9	0.01	−	0	0	0.8	1152	0.001	1000	16/16	16/15
0.9	0.01	−	1	0	0.8	2304	0.001	1000	26/26	47/46
0.9	0.01	−	1	1	0.8	7600	0.001	1000	−/−	TO/TO
0.9	0.01	−	2	0	0.8	3456	0.001	1000	46/46	173/177
0.9	0.01	−	2	1	0.8	11400	0.001	1000	156/156	2269/2291
0.9	0.01	−	2	2	0.8	87240	0.001	1000	−/−	TO/TO
0.9	0.01	0.6	0	0	0.8	2432	0.01	100	11/21	41/79
0.9	0.01	0.6	1	0	0.8	4864	0.01	100	21/21	156/160
0.9	0.01	0.6	1	1	0.8	16496	0.01	100	33/40	1420/1511
0.9	0.01	0.6	2	0	0.8	7296	0.01	100	32/22	573/477
0.9	0.01	0.6	2	1	0.8	24744	0.01	100	34/23	3323/2386
0.9	0.01	0.6	2	2	0.8	187560	0.01	100	−/−	TO/TO

the synchronised actions I_x^1 and I_y^2 in order to obtain specifications defined on actions and obey the restrictions on the action alphabets. Moreover, every time step, the reward structure time attaches Δ_{d} to these actions to measure the progress of time.

For $i \in \{1, 2\}$, the reward structure buses^i assigns Δ_{d} for each time unit both buses of $\mathcal{G}^i$ are powered; and the reward structure fail^i assigns 1 for every time unit a short circuit occurs in $\mathcal{G}^i$. Since the synchronised actions I_{on}^1 and I_{on}^2 are taken whenever power is delivered over the interface, we attach reward structures, with the same name, assigning Δ_{d} whenever the corresponding action is taken. For each component $i \in \{1, 2\}$, the objectives are the following:

- $O_{\text{bus}}^i \equiv \mathsf{Pratio}(\text{buses}^i/\text{time})(v_{\text{b}})$: to keep the uptime of the buses above v_{b};
- $O_{\text{safe}}^i \equiv \mathsf{Pratio}^{\leq}(\text{fail}^i/\text{time})(v_{\text{f}})$: to keep the failure rate below v_{f}; and
- $O_{\text{int}}^i \equiv \mathsf{Pratio}(I_{\text{on}}^i/\text{time})(v_{\text{on}})$: if used, to keep the interface uptime above v_{on}.

We consider the local specification $\varphi^i = O_{\text{bus}}^i \wedge O_{\text{safe}}^i \wedge O_{\text{int}}^i$ for game $\mathcal{G}^i$, $i \in \{1, 2\}$. Using the (Conj) rule from Section 4.2.3, we can obtain a strategy composed of the local strategies to control the full system, satisfying the global specification $\varphi = O_{\text{bus}}^1 \wedge O_{\text{safe}}^2 \wedge O_{\text{bus}}^1 \wedge O_{\text{safe}}^2$, that is, both components are safe and the buses are powered.

Strategy Synthesis. Table 8.2 shows, for several parameter choices, the experimental results. When the interface is not used, we consider the simpler specification

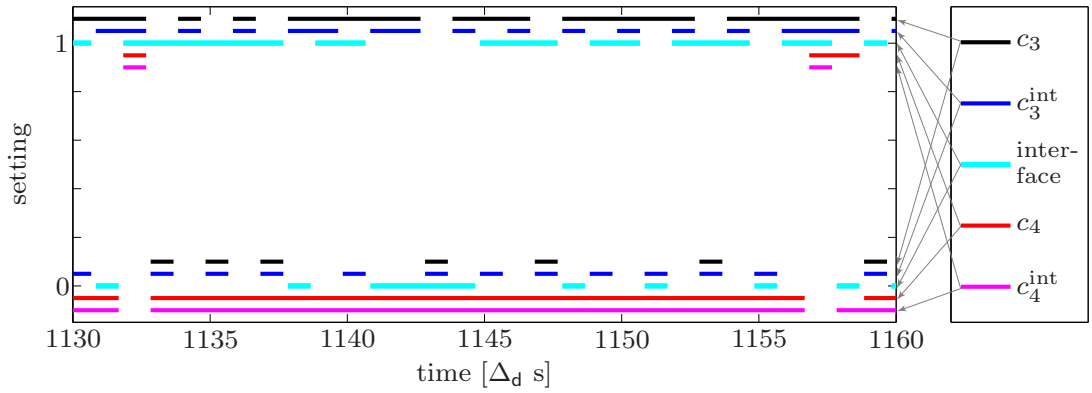


Figure 8.13: Snapshot of simulating the strategy for $N = 2$ and $d = 1$, showing the interface operation against a uniformly randomising environment.

$\varphi^i = O^i_{\text{bus}} \wedge O^i_{\text{safe}}$ for game $\mathcal{G}^i$, $i \in \{1, 2\}$. The table shows that the synthesised strategies are making effective use of the redundancy introduced by the two generators with reliability $p_g = 0.8$, and achieve a bus uptime of at least 0.9. To achieve this, the strategy cannot just leave all contactors connected, since short circuits have to be prevented. As the delay d increases, the specification becomes harder to satisfy, since the generator status may change faster than the contactors are able to compensate. For further illustration, Figure 8.13 shows a simulation trace of the strategy for $N = 2$ and $d = 1$ for the model with interface, against a uniformly randomising adversary. While the strategy activates the interface only rarely (due to the redundancy of two generators), we can see how the intention to turn c_4 on via c_4^{int} is set only if the interface is powered, and how the turn-off delay affects c_4. The contactor c_3 changes status much more frequently, and we can observe that the delay causes not all intention changes to be realised. Towards minimising wear on the contactors, a further objective could be added to bound the switching rate.

In [140], the uptime objective was encoded in LTL by requiring that buses are powered at least every Kth time step, yielding an uptime for the buses of $1/K$, which translates to an uptime of 20% (by letting $K = 5$). In contrast, using stochastic games we can utilise the statistics of the generator reliability, and obtain bus uptimes of up to 90% for generator health $p_{\text{g}} = 0.8$. For the models without delay, the synthesised strategies approximate memoryless deterministic strategies, but, when adding delay, randomisation is introduced in the memory updates.

8.3.4 Discussion

We analysed the control of electric power distribution in components of a more-electric aircraft. The design of the power system is componentised, which we reflect in our model by using multiple components that interact via an interface. Our specifications are formulated using ratio reward objectives with almost sure satisfaction semantics, and we use rewards to quantify the quality of the synthesised strategies.

Since our model is inspired from the LTL model of [140], all state variables are Booleans, for example, a bus can only either be powered or unpowered. Hence, if a bus is powered, all connected loads are assumed to be powered. To obtain a more accurate model, one could encode electrical current flows through the AC systems in numerical variables, and use Kirchoff's current law to ensure that the current flows at branches are balanced to zero, while interpreting the generators as idealised current sources with infinite impedance.

A shortcoming of our model is that the interface is not modelled using asymmetric contracts of the form $A \to B$. Instead, we encode an assumption using a stochastic state so that I^i_{on} is chosen with probability v_{on}. However, this yields to deadlocks in the composition: for example, if in $\mathcal{G}^1$ the interface distribution samples that the interface delivers power, that is, going to s_5 and enabling I^2_{on}, then it may be the case that $\mathcal{G}^2$ does not deliver power, and does not have I^2_{on} enabled, even if it is ensured that with probability v_{on} I^2_{on} is taken. The global property is still satisfied due to the fairness condition in the assume-guarantee rules, which ensures that the deadlocked states in the composition are never accessed (see Section 4.2.3). Yet, a more natural way of modelling the interface is to use nondeterminism on the "receiver" side, and apply asymmetric contracts. Synthesis of such queries, including disjunctions for Pratio objectives, is beyond the scope of this thesis, as is resolving the circularity arising from specifications of the form $A \to B$ for $\mathcal{G}^1$ and $B \to A$ for $\mathcal{G}^2$.

Further, for a safety-critical system such as an aircraft, the target values for the quoted objectives do not offer satisfactory performance of the strategy synthesis implementation. Theoretically, our algorithms can achieve arbitrary accuracy, but would exceed realistic computational performance expectations, as Table 8.2 indicates. Studying model checking for rare events may yield insights towards synthesis of strategies for such critical systems [9].

Possible Extensions. The electrical power system in [97] consists of further subsystems dedicated to low-voltage AC, and both high- and low-voltage DC, that can all be modelled similarly in our framework. Between each pair of interacting compo-

nents, an interface is required, and to prevent deadlocks the multi-stage game model needs to prescribe a fixed order for the interface actions between components. This is merely a modelling requirement and has no effect on the achievability of the specification, since each component views all other components as a single environment (albeit in our current model represented as a distribution over interface actions).

Availability. The model file `power_comp.prism` and properties file `power_comp.props` are available in the `games/power` directory of the PRISM-games 2.0 examples, and are reproduced in Appendix B.3.

8.4 Room Temperature Control

We consider a benchmark for temperature control in several rooms of a house, where we are interested in regulating the temperature in each room as close as possible around a given setpoint. This case study is inspired by the room temperature control benchmark of [61]. We consider the continuous dynamics of the temperature in several rooms under probabilistic perturbation, and derive a finite-state stochastic game representation using a discretisation procedure.

8.4.1 Problem Setting

Consider a building with several rooms. The temperature in each room depends on the adjacent rooms, the outside ambient temperature, and whether or not windows are opened. Factors such as people in the room radiating heat, or uneven wall isolation, affect the temperature, which we model as probabilistic uncertainty. Standard models for temperature regulation are given as stochastic differential equations (SDEs), and to obtain finite-state turn-based stochastic games we need to discretise the SDEs, for which we use methods similarly to those in [126, 128]. We thus model the problem as discrete-time SDEs, where we use familiar physical parameters, discretise the state space formally to obtain finite games, and then perform strategy synthesis. The obtained strategies can then be mapped back to the continuous-space model. Furthermore, the error incurred during discretisation can be formally bounded. Our focus is to demonstrate strategy synthesis on the discretised models.

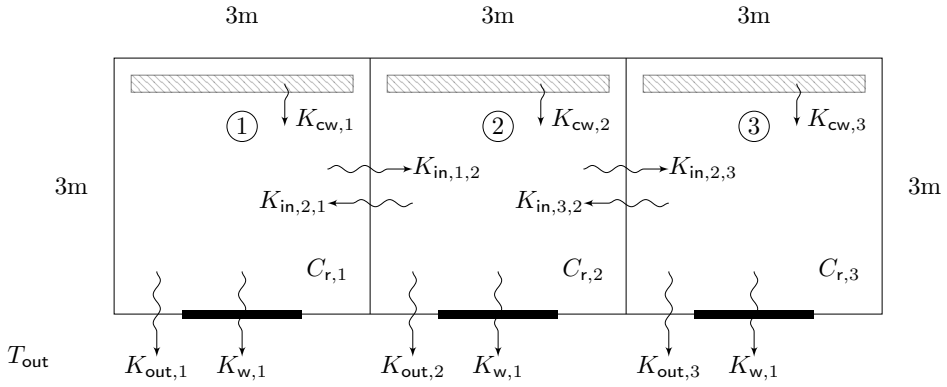


Figure 8.14: Room layout. Room indices are shown in circles, coolers are shown as hatched rectangles, and heat transfer is indicated by arrows.

8.4.2 Model

We first develop the discrete-time, continuous space SDEs, and then show how they are discretised. Let I be the index set of the rooms, and, for each room $i \in I$, let $J_i \subseteq I \backslash \{i\}$ be the index set of its neighbours. Each room i has a valve, controlled by $u_{\mathsf{v},i}$ that can take one of five settings in $\{0, \frac{1}{4}, \frac{2}{4}, \frac{3}{4}, 1\}$, corresponding to the extent it is opened. Each room also has some outward-facing window of size $1.4 \times 1.4\text{m}^2$, controlled by $u_{\mathsf{w},i}$ that can be either 0 or 1, corresponding to closed and open, respectively. Note that these controls are discrete. We consider rooms of uniform size $3 \times 3 \times 2\text{m}^3$, with the layout of the rooms in Figure 8.14, but this we can straightforwardly generalise. We assume that the inputs $u_{\mathsf{v},i}$ and $u_{\mathsf{w},i}$ are kept constant between time steps, which are of length Δ_{d}. We also assume that walls have negligible dynamics, and that the air is well mixed, that is, the temperature in each room is uniform. For a room i, the dynamics of its temperature x_i as a function of time t is given by

$$x_i(t+1) = x_i(t) + \frac{\Delta_{\mathsf{d}}}{C_{\mathsf{r},i}} \cdot \textstyle\sum_{j \in J_i} K_{\mathsf{in},i,j} \cdot (x_j(t) - x_i(t)) \tag{8.1}$$

$$+ \frac{\Delta_{\mathsf{d}}}{C_{\mathsf{r},i}} \cdot K_{\mathsf{cw},i} \cdot u_{\mathsf{v},i}(t) \cdot (T_{\mathsf{cw}} - x_i(t)) \tag{8.2}$$

$$+ \frac{\Delta_{\mathsf{d}}}{C_{\mathsf{r},i}} \cdot (K_{\mathsf{out},i} + K_{\mathsf{w},i} \cdot u_{\mathsf{w},i}(t)) \cdot (T_{\mathsf{out}} - x_i(t)) \tag{8.3}$$

$$+ \eta_{\mathsf{r},i}(t), \tag{8.4}$$

where we define the symbols in Table 8.3. The components of the temperature dynamics are: the head transfer between neighbouring rooms in (8.1); the effect of cooling in (8.2); the effect of the windows in (8.3); and a disturbance $\eta_{\mathsf{r},i}(t)$ in (8.4), which is

Table 8.3: Symbols used in definition of temperature dynamics of room i.

Symbol	Value	Units (SI)	Description
$x_i(t)$		$[°C]$	room temperature at time t
$u_{\mathsf{v},i}$		$[–]$	control for the valve
$u_{\mathsf{w},i}$		$[–]$	control for the window
$C_{\mathsf{r},i}$	2200	$[\mathrm{J\,K^{-1}}]$	thermal capacity of room
Δ_{d}	$20 \cdot 60$	$[s]$	temporal discretisation period
$K_{\mathsf{cw},i}$	2.5	$[\mathrm{W\,m^{-2}\,K^{-1}}]$	convection coefficient of chilled water
$K_{\mathsf{in},i,j}$	1.2	$[\mathrm{W\,m^{-2}\,K^{-1}}]$	convection coefficient of inside wall facing neighbour $j \in J_i$
$K_{\mathsf{out},i}$	$(1.8, 1.2, 1.8)$	$[\mathrm{W\,m^{-2}\,K^{-1}}]$	convection coefficient of outside wall
$K_{\mathsf{w},i}$	0.78	$[\mathrm{W\,m^{-2}\,K^{-1}}]$	convection coefficient of window
T_{out}	30	$[°C]$	nominal outside ambient temperature
T_{cw}	10	$[°C]$	chilled water temperature
T_{set}	20	$[°C]$	temperature setpoint
$\eta_{\mathsf{r},i}$		$[°C]$	room temperature disturbance
$\sigma^2_{\mathsf{r},i}$	$65^{-2} \cdot \Delta_{\mathsf{d}}$	$[°C^2\,s]$	variance of room temperature disturbance

a zero-mean stationary random process with variance $\sigma^2_{\mathsf{r},i}$. The thermal capacity $C_{\mathsf{r},i}$ of room i is proportional to the volume of the room, and the thermal convection coefficients K_{cw}, K_{in}, K_{out} and K_{w} are proportional to the respective area of the contact surfaces. We abbreviate the dynamics by writing $x_i(t{+}1) = f_i(x_i, \vec{x}^i, u_{\mathsf{v},i}, u_{\mathsf{w},i}){+}\eta_{\mathsf{r},i}(t)$, where we denote by $\vec{x}^i = (x_{j_1}, x_{j_2}, \ldots)$ the vector of temperatures of the neighbours $J_i = \{j_1, j_2, \ldots\}$ of room i.

Discretisation. We discretise the dynamics by gridding the continuous temperatures x_i, for each room i, into m_i uniformly sized bins, and we consider temperatures at $\pm 2°C$ around the temperature setpoint T_{set}. Hence, the bins have *diameter* $\delta_i = 4/m_i$, and we define the ℓth bin for x_i by $B_{i,\ell} \stackrel{\text{def}}{=} \{x_i \,|\, (\ell-1)\cdot\delta_i \leq x_i - T_{\mathsf{set}} + 2 \leq \ell\cdot\delta_i\}$, and its *representative point* by $b_{i,\ell} \stackrel{\text{def}}{=} T_{set} - 2 + \frac{2\ell-1}{2}\delta_i$, which is at the centre of the bin. Given x_i, we denote by $\tilde{x}_i$ the representative point $b_{i,\ell}$ of the bin such that $x_i \in B_{i,\ell}$. Note that the temperature of each room depends on the temperatures of its neighbours (see (8.1)), and so we discretise all temperature dimensions per room. We want to obtain a component game for each room, which we then compose by synchronising on actions labelling transitions that indicate that the room temperature changes between bins. For each temperature dimension i, we thus need for each bin in room i a corresponding bin in all its neighbours J_i, whose boundaries precisely align. This is why, for each given temperature dimension i, we use the exact same

grid for each room (but different temperature dimensions can have different grids). For the presented evaluation, we fix $m_i = 5$ for all i.

The SDE model specifies a probability to go from $x_i(t)$ to $x_i(t+1)$ for each value pair. In the discretised model, we need to calculate the probability to transition between bins, which is calculated by marginalising the distributions. We can evaluate the probability of transitioning from $\tilde{x}_i$ to $\tilde{x}'_i$ given that the neighbours of room i are at temperatures $\vec{\tilde{x}}^i$, and for fixed controls $u_{\mathsf{v},i}$ and $u_{\mathsf{w},i}$, by

$$\tilde{P}_i(\tilde{x}'_i, \tilde{x}_i, \vec{\tilde{x}}^i, u_{\mathsf{v},i}, u_{\mathsf{w},i}) = \frac{1}{\eta_{\mathsf{norm}}} \cdot \int_{\tilde{x}'_i - \delta_i/2}^{\tilde{x}'_i + \delta_i/2} \phi(x'_i \mid f_i(\tilde{x}_i, \vec{\tilde{x}}^i, u_{\mathsf{v},i}, u_{\mathsf{w},i}), \sigma^2_{\mathsf{r},i})\, dx'_i,$$

where $\phi(x \mid \mu, \sigma^2_{\mathsf{r},i})$ is the Gaussian probability density function with mean μ and variance $\sigma^2_{\mathsf{r},i}$, and $\eta_{\mathsf{norm}} \stackrel{\text{def}}{=} \sum_{\tilde{x}'_i} \tilde{P}_i(\tilde{x}'_i, \tilde{x}_i, \vec{\tilde{x}}^i, u_{\mathsf{v},i}, u_{\mathsf{w},i})$ is used to normalise the distribution, since we truncate temperatures outside the range ± 2°C in the binning process.

Stochastic Game Formulation. From the discretised transition probabilities $\tilde{P}_i$ for the individual rooms i, we now construct stochastic games. Recall that the temperature $\tilde{x}_i$ is controlled by setting the valve control $u_{\mathsf{v},i}$, which has to counteract the effects of the window setting $u_{\mathsf{w},i}$, as well as the temperature in the neighbouring rooms $\vec{\tilde{x}}^i$. Since we model turn-based games, we have to express the temporal sequence of setting these variables in several stages, which we model similarly to the aircraft model in Section 8.3. We thus let Player 1 (the controller) set $u_{\mathsf{v},i}$, and Player 2 (the environment) set $u_{\mathsf{w},i}$ and $\vec{\tilde{x}}^i$. The temperature dynamics $\tilde{x}_i$ are then set without nondeterminism, but stochastically according to $\tilde{P}_i$. Furthermore, in our compositional model, we need to introduce action names to communicate the temperature of the neighbouring rooms between component games. We introduce communication stages similarly to the aircraft model, where each component i sends its temperature $\tilde{x}_i$ using the temp$i$_$\tilde{x}_i$ action, and receives the temperature $\tilde{x}_j$ from neighbouring room $j \in J_i$ by synchronising on the temp$j$_$\tilde{x}_j$ action. We show the multi-stage game $\mathcal{G}^i$ for room i in Figure 8.15. Note that the games $\mathcal{G}^i$ are controllable multichain, since the disturbance $\eta_{\mathsf{r},i}$ ensures a high degree of connectivity of the discretised game. Furthermore, the games are compatible and do not deadlock.

Since we are interested in keeping the temperature around a setpoint, we define reward structures tempdev_i for the respective rooms i by $\text{tempdev}_i(x_i) = (T_{\mathsf{set}} - x_i)^2$. The reward structures are discretised by considering the representative points, and so we get $\text{tempdev}_i(\tilde{x}_i) = (T_{\mathsf{set}} - \tilde{x}_i)^2$. To measure time, we note that the temp2_$\tilde{x}_2$ action is synchronised across all rooms, and so we define the reward structure time by

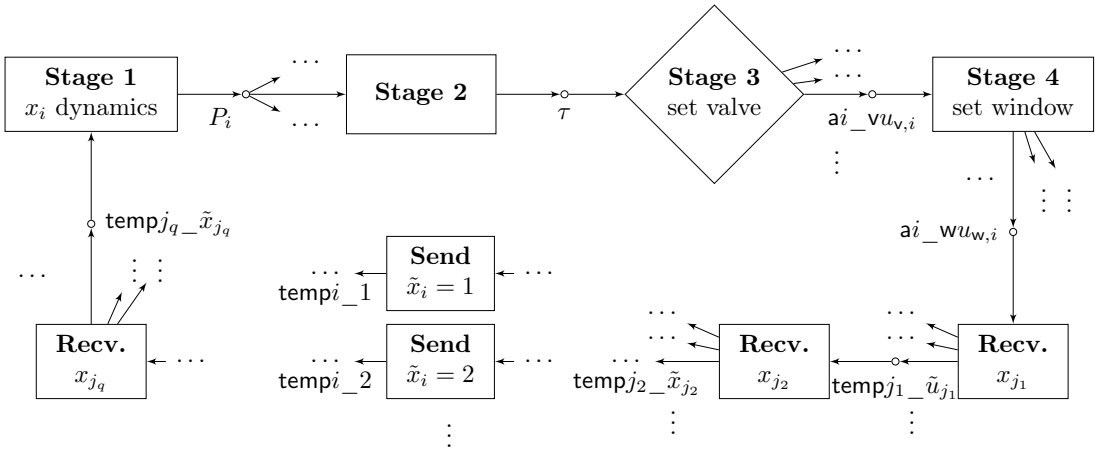


Figure 8.15: Discretised multi-stage game. Stages 1–4 correspond to controlling the window, valve and temperature dynamics of the room, while the receive and send stages correspond to modelling the interaction with the neighbouring rooms. We abbreviate $|J_i| = q$.

$\text{time}(\mathsf{temp2\_}\tilde{x}_2) = 1$ for all $\tilde{x}_2$ (alternatively, we could introduce an additional stage in the games to synchronise time.) We can now consider ratios of rewards that are independent of the number of stages in the discretised games.

Discretisation Error Computation. We estimate the discretisation errors, which depend on the reward structures used in a similar way to [129]. We consider the N-step error of total rewards, where one step is one loop in the multi-stage game, and then divide by $N \cdot \text{time}(\mathsf{temp2\_}\tilde{x}_2) = N$ for the ratios of rewards. We can express the temperature dynamics of room i in affine form as

$$x_i(t+1) = \alpha_i \cdot x_i(t) + \sum_{j \in J_i} \alpha_{i,j} \cdot x_j(t) + \beta_{\mathsf{v},i} \cdot u_{\mathsf{v},i}(t) + \beta_{\mathsf{w},i} \cdot u_{\mathsf{w},i}(t) + \eta_{\mathsf{r},i}(t),$$

with constants α_i, $\alpha_{i,j}$, $\beta_{\mathsf{v},i}$ and $\beta_{\mathsf{w},i}$ expressing the influence of the respective variables. For each room i, Player 1 is interested in optimising a reward function $J^*_{i,0}$, depending on the (expectation) objective under consideration. Consider a reward structure yielding a value of $R_{i,t}(x_i)$ at time t for a temperature x_i. At time t, $J^*_{i,t}$ consists of the reward $R_{i,t}$ at time t, up to (including) time N (sometimes called the *cost-to-go*), and the reward H_i from time $N+1$ to infinity. Hence, we define

$$J^*_{i,t}(x_i) \stackrel{\text{def}}{=} \min_{u_{\mathsf{v},i}} \max_{u_{\mathsf{w},i}} \max_{\vec{x}^i} \left(R_{i,t}(x_i) + \int J^*_{i,t+1}(x'_i) \cdot \phi(x'_i \mid f_i(x_i, \vec{x}^i, u_{\mathsf{v},i}, u_{\mathsf{w},i}), \sigma^2_{\mathsf{r},i})\, dx'_i \right)$$

$$J^*_{i,N+1}(x_i) \stackrel{\text{def}}{=} H_i(x_i).$$

To obtain a bounded error, we require Lipschitz continuity of the rewards, that is,

$$|H_i(x_i) - H_i(x_i')| \leq h_{H,i} \cdot |x_i - x_i'|$$
$$|R_{i,t}(x_i) - R_{i,t}(x_i')| \leq \sum_{j \in J_i \cup \{i\}} h_{R,j} \cdot |x_j - x_j'|,$$

for some $h_{H,i}$ and $h_{R,j}$. We recursively define $h_{i,t}$ by $h_{i,N+1} \stackrel{\text{def}}{=} h_{H,i}$, and $h_{i,t} \stackrel{\text{def}}{=} h_{R,i} + h_{i,t+1} \cdot |\alpha_i|$, and obtain that the reward $J^*_{i,t}$ is also Lipschitz continuous, that is,

$$\left|J^*_{i,t}(x_i) - J^*_{i,t}(x_i')\right| \leq h_{i,t} \cdot |x_i - x_i'|.$$

The error in discretisation for each individual quantity is at most the diameter of the bin, that is, δ_i for x_i, multiplied by the Lipschitz constant of that quantity, $h_{R,i}$. The error in $J^*_{i,t}$ at time t, defined as $E_{i,t} \stackrel{\text{def}}{=} \left|J^*_{i,t}(x_i(t)) - J^*_{i,j}(\tilde{x}_i(t))\right|$, consists of three parts: (i) the error related to the reward $R_{i,t}$, which is $\sum_{j \in J_i} h_{R,j} \cdot \delta_j$; (ii) the error related to the approximate J_{t+1}, which is $E_{i,t+1}$; and (iii) the error related to the approximation of the stochastic kernel $\eta_{\mathsf{r},i}$ inside the integral, which is $h_{i,t+1} \cdot \sum_{j \in J_i} |\alpha_{i,j}| \cdot \delta_j$. Note that the error is independent of the already discretised inputs. Thus,

$$E_{i,t} = \sum_{j \in J_i} h_{R,j} \cdot \delta_j + E_{i,t+1} + h_{i,t+1} \cdot \sum_{j \in J_i} |\alpha_{i,j}| \cdot \delta_j, \qquad E_{i,N+1} = h_{H,i} \cdot \delta_i.$$

We obtain that

$$E_{i,0} = (N+1) \cdot \sum_{j \in J_i} h_{R,j} \cdot \delta_j + h_{H,i} \cdot \delta_i + (\sum_{t=0}^{N} h_{i,t+1}) \cdot \sum_{j \in J_i} |\alpha_{i,j}| \cdot \delta_j,$$

and, as we consider the long-run ratio with time, that is, the number of steps, in the denominator, we get the error

$$\lim_{N \to \infty} \frac{E_{i,0}}{N+1} = \sum_{j \in J_i} h_{R,j} \cdot \delta_j + \frac{h_{R,i}}{1 - |\alpha_i|} \cdot \sum_{j \in J_i} |\alpha_{i,j}| \cdot \delta_j.$$

To ensure reasonable efficiency of our synthesis implementation, we discretise only very crudely, obtaining large errors. We emphasise, however, that the errors are over-estimates, and can be made arbitrarily small by increasing the number of bins m_i. Thus, with the current configuration, we obtain an approximate discretisation error of 10°C for each temperature dimension, which is magnitudes above the target values

we are interested in, but we note that these are overestimates and can be tightened by increasing the number of bins.

8.4.3 Analysis

We now analyse the composed stochastic game $\mathcal{G} = \mathcal{G}^1 \parallel \mathcal{G}^2 \parallel \mathcal{G}^3$. We are interested in the ratio of the temperature deviation over time, which are defined on traces. We also want to use the cooling circuits as little as possible to save energy, and hence introduce the reward structures valve_i for room i, defined by $\text{valve}_i(\mathsf{a}i\_\mathsf{v}u_{\mathsf{v},i}) = \frac{u_{\mathsf{v},i}}{4}$, corresponding to the valve setting $u_{\mathsf{v},i}$. Lastly, we can impose an assumption on the time the windows are opened using the reward structure window_i for room i, defined by $\text{window}_i(\mathsf{a}i\_\mathsf{w}u_{\mathsf{w},i}) = u_{\mathsf{w},i}$.

For each room i, we consider the objectives $O^i_{\text{td}} \equiv \mathsf{Eratio}^{\leq}(\text{tempdev}_i/\text{time})(v^i_{\text{td}})$ to minimise the temperature deviation, $O^i_{\text{v}} \equiv \mathsf{Eratio}^{\leq}(\text{valve}_i/\text{time})(v^i_{\text{v}})$ to minimise the energy usage from opening the valve, and $O^i_{\text{w}} \equiv \mathsf{Eratio}^{\leq}(\text{window}_i/\text{time})(v^i_{\text{w}})$ to minimise the time the windows are open. We consider two scenarii. Firstly, $\psi^1 = O^1_{\text{v}} \wedge (O^2_{\text{td}} \to O^1_{\text{td}})$ for $\mathcal{G}^1$, $\psi^2 = O^2_{\text{v}} \wedge O^2_{\text{td}}$ for $\mathcal{G}^2$, and $\psi^3 = O^3_{\text{v}} \wedge (O^2_{\text{td}} \to O^3_{\text{td}})$ for $\mathcal{G}^3$ bound the temperature deviation and the amount the valve is opened, for each room independently, assuming the respective neighbours have bounded temperature deviation. Applying our composition rules from Section 4.2.3, we derive the global specification $\psi = \bigwedge_{i \in I} O^i_{\text{v}} \wedge O^i_{\text{td}}$, which requires that all rooms have regulated temperature without using excessive cooling. Secondly, $\varphi^1 = (O^1_{\text{w}} \wedge O^2_{\text{td}}) \to O^1_{\text{td}}$ for $\mathcal{G}^1$, $\varphi^2 = O^2_{\text{w}} \to O^2_{\text{td}}$ for $\mathcal{G}^2$, and $\varphi^3 = (O^3_{\text{w}} \wedge O^2_{\text{td}}) \to O^3_{\text{td}}$ for $\mathcal{G}^3$ bound the temperature deviation under the assumption that the windows are not excessively opened, and the respective neighbours have bounded temperature deviation. The global specification we derive is $\varphi = \bigwedge_{i \in I} O^i_{\text{w}} \to \bigwedge_{i \in I} O^i_{\text{td}}$, requiring that all rooms have regulated temperature, given that the windows in all rooms are not opened for too long. Note that room 2 does not impose an assumption on its neighbours, since it only has two outward facing walls, and hence its temperature is more easily regulated than that of rooms 1 and 3, which have three outward facing walls. The absence of assumptions on neighbours for room 2 allows us to break the circularity when applying our composition rules (see the discussion in Section 4.5) when deriving the global specifications.

Strategy Synthesis. We synthesise strategies for the discretised game using our assume-guarantee framework. The runtimes are reported in Table 8.4. We see that imposing requirements on the valve means that the achievable temperature deviation is high. However, if we assume that the windows are closed, the temperature de-

Table 8.4: Performance for room temperature case study. $M_{\min} = 2$, $M_{\max} = 10000$ (never hit). The $\mathcal{G}$ column is the total time for assume-guarantee synthesis. Increase factor $\eta = 1$. The discretised models $\mathcal{G}^1$, $\mathcal{G}^2$ and $\mathcal{G}^3$ have 1478, 1740 and 1478 states, respectively.

Spec.	Target			Algorithm Parameters				Synthesis Time [s]		
	$\vec{v}_{\mathrm{t}}$	$\vec{v}_{\mathrm{v}}$	$\vec{v}_{\mathrm{w}}$	ε	a_0	$k_{\max}$	$d_{\max}$	$\mathcal{G}^1$	$\mathcal{G}^2$	$\mathcal{G}^3$
ψ	0.8	0.8	–	0.05	100	250	10	678	27	621
ψ	0.8	0.8	–	0.02	500	250	10	1115	29	843
ψ	0.8	0.8	–	0.01	1000	250	10	3370	34	8605
φ	0.3	–	$(0.3, 0.2, 0.3)$	0.05	100	100	20	829	69	734
φ	0.3	–	$(0.3, 0.2, 0.3)$	0.02	500	100	20	852	72	749
φ	0.3	–	$(0.3, 0.2, 0.3)$	0.01	1000	100	20	860	92	2480

viation can be considerably lowered (the middle room has a slightly more stringent requirement here, but also no assumptions on its neighbours.) We do not consider specifications containing both O^i_{v} and O^i_{w}, since the increased dimensionality penalises performance of the algorithms. The current results are computed in three different accuracies ε, where we adjust the baseline accuracy a_0 accordingly, since the expected truncated energy may diverge with a rate $1/a_0$ if rounding, even if the specification is achievable, and so we require $1/a_0 < \varepsilon$. However, when not using rounding, the number of extreme points in the fixpoint computation slows the computation. Even here we can see that with increased accuracy the time required for synthesis shows a clear upward tendency. Note also that the execution time depends on the algorithm parameters, see Table 8.1. Considering the full composed model monolithically, which has of the order of 10^9 states, is outright infeasible for the global multi-objective specifications ψ and φ, and challenging even if only a single objective is considered.

Strategy Exploration. We utilise the simulator of PRISM-games 2.0 to analyse the synthesised strategies. Figures 8.16 and 8.17 show the temperature in room 1 for ψ^1 and ϕ^1 respectively, as well as the relevant reward structures, for the first 35 time steps of length Δ_{d} seconds of simulating against a uniformly randomising environment. In Figure 8.16, since the strategy satisfies ψ^1, the valve is not used more than at a rate 0.8. Furthermore, even though the temperature deviation in room 2 is above 0.8, the strategy still regulates the temperature deviation in room 1 below 0.8. In Figure 8.17, the window is not open more than 0.3 of the time, but the temperature deviation in room 2 is above 0.3. Still, the strategy is able to regulate the temperature deviation in room 1 below 0.3. In both figures we can observe that the strategy opens the valve if the temperature goes above the setpoint of 20°C, and closes if the temperature falls.

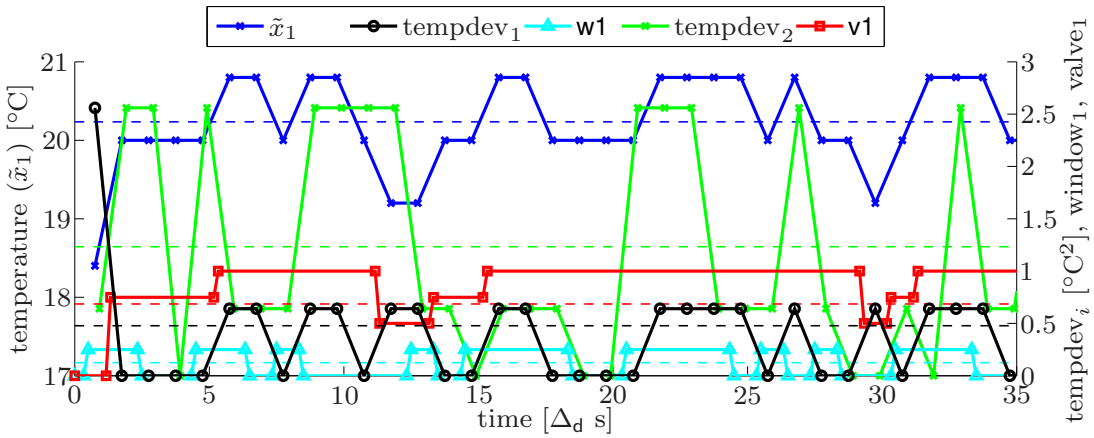


Figure 8.16: Simulating the strategy for ψ^1. Dashed lines are averages over 150 steps.

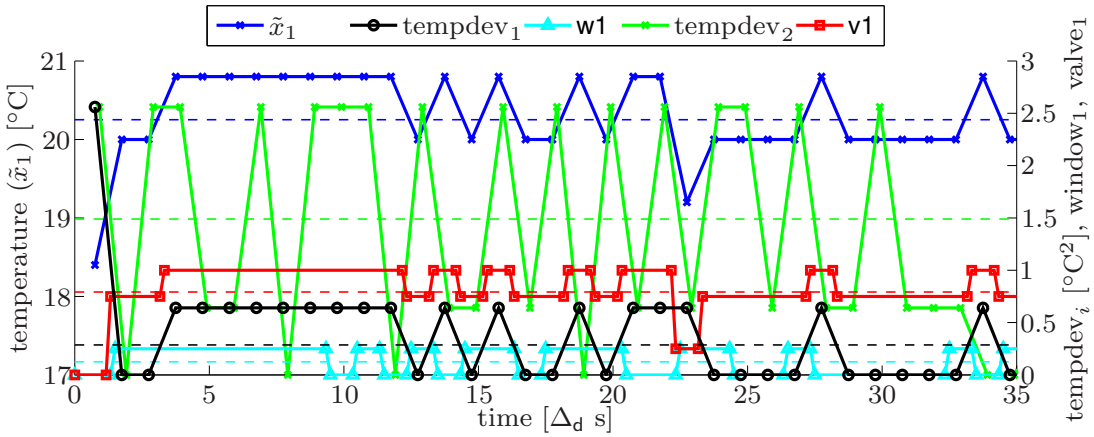


Figure 8.17: Simulating the strategy for ϕ^1. Dashed lines are averages over 150 steps.

8.4.4 Discussion

We have derived a compositional stochastic game model for controlling the temperature in several rooms to maintain an overall condition in a building. Having a strategy per room means also that we assume a cooling circuit in each room that can be independently controlled, but this can be straightforwardly relaxed since the room dynamics can be heterogeneous. The model parameters have the expected effect on the dynamics: lower thermal convection coefficients mean better isolation, and so we assume that the outside walls are better isolated than the inner walls. Further, if increasing the room size, the neighbouring rooms have decreased influence, since the room temperature is controlled primarily by the local cooling circuit.

Comparing this case study with our aircraft electric power system case study of Section 8.3, we note that the power system is modelled directly as a stochastic game,

and so the discrete dynamics are overt and can be analysed and augmented if necessary. In contrast, after discretising the room temperature dynamics, the finite-state model no longer has a straightforward correspondence to the original equations. The multi-step model also needs to reflect the control dynamics; for example, swapping the order in which the window and valve status are set may yield different achievability results. A key advantage of considering the SDE formulation, however, is that the dynamics can be expressed in a formalism familiar to engineers, and hence the precise encoding as a stochastic game does not have to be examined.

Possible Extensions. A straightforward extension is to consider different room topologies, rather than the one we analyse in the concrete instance studied in this section. When a room has several neighbours, their temperature deviations can be lumped together as a single assumption, that is, instead of requiring temp_{j_1} and temp_{j_2} to be below $v_{\text{t}}^{j_1}$ and $v_{\text{t}}^{j_1}$ we could require that $\text{temp}_{j_1} + \text{temp}_{j_2}$ is below $v_{\text{t}}^{j_1} + v_{\text{t}}^{j_1}$, ensuring that the dimensionality of the specifications does not increase with the number of neighbours. Our formulation already provides for such extensions, but care has to be taken to avoid circularity in deriving the global specification. Further, the control of temperatures in rooms can be generalised to other thermo- or fluid-dynamic systems that require control against an unknown environment.

Availability. The related files are in the `games/temperature` subfolder of the examples directory in the PRISM-games 2.0 release. To run the computations, the model and properties file can be generated using `temperature.m` (reproduced in Appendix B.4), which also produces a bash script to run the synthesis.

8.5 Summary

We have presented four case studies of controller synthesis for systems modelled as stochastic games. In the study of autonomous driving in an urban setting (Section 8.1) and path planning by an UAV (Section 8.2), we investigated strategy synthesis for multi-objective specifications with expected total rewards in monolithic models. We then employed our assume-guarantee framework to synthesise strategies controlling the electrical power distribution in an aircraft (Section 8.3), and regulating the temperature in several rooms of a building (Section 8.4), both using multi-objective specifications with ratios.

Our synthesis methods are highly dependent on the size of the state space and the desired accuracy parameters of our algorithm implementation. A key factor de-

termining the efficiency of our implementation is that the number of extreme points in the computed polytopes may increase exponentially with the iteration count (see also Section 7.2.1). The number of extreme points also affects the time for the strategy construction. However, we also observe that the memory that is used in the constructed strategies is generally small compared to the full memory size resulting from using all extreme points at each state, suggested in the strategy construction of Definition 6.2. When restricting the memory a-priori to a certain size, an approach that does not use the geometric interpretation may be possible. Such an approach could improve efficiency, but also sacrifice completeness in the most general case.

In particular, in the room temperature control case study, we demonstrated that we can obtain stochastic game models from standard physical models used in control system design. This modelling capability allows us to shift the design and debugging effort to the overt model, and the effects are then transferred to the game via a "correct-by-construction" discretisation. This alleviates the effort of working with hard-coded discrete dynamics in the game that do not have a straightforward correspondence with the original physical processes that are being modelled. The current discretisation method uses uniform bin sizes, leading to high (albeit conservative) discretisation error estimates. Finding tighter error bounds and adapting bin sizes according to the error, as in the monolithic setting (see, for example, [126]), is an attractive topic of further research.

The case studies demonstrated that our assume-guarantee framework has the potential to greatly improve scalability of strategy synthesis. The most significant advantage of assume-guarantee strategy synthesis is that the complexity grows linearly with the number of components, compared to an exponential growth in the monolithic setting (the number of players stays constant during composition, and therefore does not influence complexity). For the mulit-component systems considered in the case studies of Sections 8.3 and 8.4, the fully compositional models have in the order of 10^6 to 10^{10} states, and we see in Table 8.2 that the synthesis times out at 4 h already for models of state space size in the order of 10^3 to 10^5 states, depending on the selected accuracy and number of objectives. An experimental analysis demonstrating that our assume-guarantee approach can greatly reduce the runtime of synthesis compared to directly synthesising for the full monolithic system appeared in [11], but with expected total reward objectives only. Note, however, in the monolithic setting only the immediately required objectives have to be included in the specification, while in the assume-guarantee setting objectives are required from the designer to specify the interface contracts between components, which additionally may lead to

higher synthesis times per component. Also note that finding the weight vectors (see Section 5.4) for specifications involving disjunctions adds further complexity, and is required mostly in the assume-guarantee setting. Yet, since the local specifications do not have to scale with the number of components, the scalability benefits in our assume-guarantee framework dominate as the number of components increases.

Chapter 9

Conclusions

9.1 Summary

The central question examined in this thesis is how to automatically find controllers for systems that operate in an unpredictable, potentially adverse environment. We assume that some of the uncertainty in the environment is quantifiable, so that we can assume that it behaves according to a probability distribution, while some of the uncertainty may be entirely unknown, which we model as nondeterminism, obtaining models that are more general than MDPs. We therefore model a system as a stochastic game, and consider the problem of finding a strategy that wins the game against the environment, which can then be interpreted as a controller for the modelled system.

Often, a system is built from several components in order to manage the complexity in its design and implementation. We have seen in our case studies two examples of such a compositional design approach (an aircraft electric power system and room temperature control), illustrating the benefits and challenges in the analysis of multi-component systems. The most appealing benefit of a compositional approach is that large systems can be more efficiently analysed component by component, with clearly defined interfaces, rather than having to take the whole system into consideration at the same time, and hence scalability of the analyses is improved. In order to utilise the compositionality of the system in strategy synthesis, we introduce, in Chapter 4, a formal framework for composing subsystems, and for specifying the interactions between them. To synthesise a strategy for the full, global system, the assume-guarantee synthesis approach amounts to synthesising a local strategy for each component, and then composing these strategies to obtain a strategy for the global system. We can then employ *assume-guarantee synthesis rules* to ensure that the global strategy sat-

isfies the required specification: a synthesis rule guarantees a global specification for the full system under the assumption that the components satisfy their local specifications. Using assume-guarantee rules we can formally define the interaction between components, and, in particular, define interfaces, or contracts, between the components.

Applying assume-guarantee synthesis rules requires finding strategies for the individual components. We develop strategy synthesis methods for specifications that consist of multiple quantitative objectives. Our motivations for considering Boolean combinations of objectives are, on the one hand, to specify contracts between subsystems, which make use of logical implications such as $A \to B$, and, on the other hand, to achieve trade-offs between conflicting goals, such as optimising the output of a plant, while keeping the costs at a minimum. Synthesising strategies for multi-objective queries is challenging because all objectives have to be considered at the same time. We develop a set of transformations allowing one to analyse Boolean combinations of objectives (such as expected mean-payoff and ratio of expectations) by considering conjunctions of almost sure mean-payoff objectives, for which we give a strategy synthesis algorithm. The synthesis methods we develop focus on long-run objectives, but our tool implementation also supports expected total rewards.

Contributions. We can summarise our contributions in two domains:

- We develop the theoretical foundations for an assume-guarantee strategy synthesis framework for stochastic games. Our work supports modelling systems as stochastic games, strategy synthesis for Boolean combinations of quantitative objectives, assume-guarantee rules for analysing strategies of composed games, and computing compositional Pareto sets to instantiate targets for the local specifications. While previous work has focused on subsets of these aspects, to the best of our knowledge, this thesis provides the first comprehensive framework. In Chapter 4, we address the composition of stochastic games for synthesis, where the key challenge is to ensure that winning strategies for the individual components can be composed to a winning strategy for the full game. Further, previous work on strategy synthesis largely considers combinations of only two aspects among (i) multiple quantitative objectives, (ii) the presence of an adversary (Player 2), and (iii) probabilities. Combining all three aspects is a non-trivial extension over either set of two, and is the focus of Chapters 5–6.

- We demonstrate the effectiveness of our approach by implementing our algorithms in the PRISM-games 2.0 tool, which we describe in Chapter 7. Using

our tool, the user can model a system as multiple components, assign local specifications, and synthesise strategies for the individual components and the composed game. Pareto set visualisation and simulation provide insight about the system model and the synthesised strategies to the designer. We demonstrate the viability of our synthesis methods on a set of case studies in Chapter 8. The autonomous driving and UAV path planning case studies explore the modelling of autonomous systems as stochastic games and the strategy synthesis for expected total rewards. The aircraft power control and room temperature control case studies employ our assume-guarantee synthesis techniques for ratio rewards, demonstrating improved scalability.

9.2 Future Work

Several directions for future research arise from the work in this thesis. We highlight in this section directions that are of particular relevance.

Relaxing Compatibility. To facilitate componentised model development, we would like to impose minimal restrictions on the components, and ensure that these restrictions can be checked efficiently. Ideally such compatibility conditions can be checked in linear time in the size of the individual components, in particular, without requiring to build the full composition. An improvement to both the versatility of our framework and the efficiency of modelling components would be to relax our compatibility condition in Section 4.1.3, which is needed to ensure that Player 1 strategies of different components cannot suggest contradicting moves at a Player 1 state in the composition. Using *permissive* strategies that can suggest several choices in a state [58] may be a way of weakening out compatibility condition, if we can find a local condition on the components that ensures that there is a choice in the composition that all local strategies can agree on, and hence remain winning for the composed game.

Rule Inference. The ultimate goal of compositional methods is to be able to directly synthesise local strategies for the subsystems from the global system specification, or, in a similar vein, to automatically decompose the global specification into appropriate local specifications that are easier and faster to analyse. In our assume-guarantee framework, we provide the necessary formalism for composing local strategies, but it remains an open question how to find the local specifications automatically. We expect the general problem of automatic synthesis from a global specification to be

undecidable for our quantitative objectives, as this is also the case for LTL specifications [108]. Manually decomposing specifications could be facilitated by considering a set of composition rules such as the ones presented in this thesis, and by automatically inferring which rules are applicable to derive the global specification.

Strategy Classification. Development of synthesis methods greatly benefits from understanding the classes of strategies that are necessary and sufficient for the respective players to win a game. For example, showing that Player 2 only requires MD strategies allowed us to prove decidability for Pmp CQs in Section 6.1. Another consideration is that different representations of the same strategy can affect the difficulty and elegance of proofs, for example we use DU strategies in most of our proofs, but we recourse to SU strategies in the strategy construction we propose, which is also motivated by the practical consideration of needing a succinct representation. A key open question is understanding for which models and specifications finite SU strategies are strictly more powerful than finite DU strategies. A preliminary discussion is given in [50], but does not address quantitative objectives or multi-objective queries.

Counterexample-Guided Abstraction Refinement. In order to address scalability limitations in terms of the state space size when analysing formal models, a popular approach is to group similar states together and perform analysis only on these groups, obtaining a smaller model. The suitability of the smaller model can be justified by developing appropriate (probabilistic bi-)simulations for stochastic games (see [120] for PAs). Since the smaller model should yield the same answer as the original full-sized model, selecting an appropriate grouping (or *abstraction*) is a key problem, and in our case of synthesis for two-player stochastic games it is exacerbated by the presence of different kinds of states. One solution is to start with a very coarse abstraction, test if there is a *counterexample* to the feasibility of synthesis on the abstract model, and if so, gradually *refining* it, that is, splitting up groups, until the synthesis problem is feasible. The three key requirements for this approach are an abstraction for stochastic games; a method to produce finite counter-examples; and a refinement procedure based on the counter-examples. For games where both players have MD strategies, [30] addresses these three requirements, but as we already noted in Section 4.5, in multi-objective stochastic games, counterexamples may not exist due to non-determinacy, or might not be representable as MD strategies. A fruitful direction for further work would be to explore this approach of counterexample-guided

abstraction refinement (CEGAR), in particular how to find, represent, and utilise counterexamples.

General Specifications. To give a system designer more flexibility, we suggest extending the class of specifications that can be synthesised, with particular emphasis on specifications defined on traces, for use in the assume-guarantee framework. We note at this point that the types of objectives that we consider, Pmp, Pratio, Emp, Eratio, ratioE and EE, can be combined arbitrarily in conjunction, and Emp, Eratio and ratioE can be combined in MQs. Allowing both expected total rewards and long-run objectives would help to control both the transient and the recurrent behaviour of systems. Further, by using Boolean combinations of ω-regular objectives with arbitrary probability thresholds, one could specify communication protocols similarly to using GR1 specifications [107], but with the ability to tune the relative importance or criticality of assumptions and guarantees: for example, if a GR1 specification $\square\lozenge A \to \square\lozenge G$ is not satisfiable, but strengthening the assumption $\square\lozenge A$ to $\square\lozenge (A \wedge B)$ means that it can no longer be satisfied by the environment (for instance, by another component in the system), then one solution could be to consider a specification $\mathbb{P}(\square\lozenge (A \wedge B)) \geq 0.9 \to \mathbb{P}(\square\lozenge G) \geq 0.9$. Also, properties useful for the development of autonomous systems could be conditional expectations and quantiles [4], as well as minimising the variance of the mean-payoff [20]. Of similar importance to providing synthesis for more general specifications is establishing the complexity classes of the related achievability, synthesis, and Pareto set computation problems. For multi-objective stochastic games, few results are known, and the developed algorithms mostly compute approximate solutions. The complexity of synthesis in stochastic games for expected mean-payoff objectives remains to be investigated, and subsumes reachability of multiple targets, of which the complexity classification is a long standing open problem.

Bidirectional Interfaces. In order to develop bidirectional interfaces between components, it might be necessary that each component guarantees an assumption for the respective other component, that is $\mathcal{G}^1 \models \varphi_1 \to \varphi_2$ and $\mathcal{G}^2 \models \varphi_2 \to \varphi_1$. We noted in Section 4.5 that the circular rule (Circ) does not allow to resolve this case satisfactorily. A more natural view could be to operate with non-zero-sum games, where, in $\mathcal{G}^1$, Player 1 only cares about satisfying φ_1, assuming that Player 2 only cares about satisfying φ_2, and vice versa for $\mathcal{G}^2$. The desired behaviour would be that Player 1 in $\mathcal{G}^1$ tries to satisfy φ^1 but not at the price of violating φ^2, which in turn is left to be satisfied by Player 1 in $\mathcal{G}^1$. In [28] the authors explore such an approach for

non-stochastic systems, but it remains an open problem how to encode the interaction of players between components, so that they can be considered in isolation.

Concurrent Games. Finally, towards a more natural framework for componentised systems, and systems interacting with an uncontrollable environment, synthesis for concurrent games is a further important direction. Composing turn-based games requires preservation of the turn-based nature in the composition. This requirement restricts the versatility of our assume-guarantee framework, since strong conditions need to be imposed to ensure compatibility and non-blocking of the components. To address this, compositional frameworks typically assume *input-enabledness* of the components, that is, all environment actions are enabled in all states. However, when interpreting subsystems as games, input-enabledness means that the move chosen at a state may depend on the strategies of both players. Concurrent stochastic games with single quantitative ω-regular objectives have been analysed by defining a Bellman-style operator as a solution to a matrix game [56], but we are not aware of research in the multi-objective setting with quantitative properties. Furthermore, a synthesis framework for concurrent games could allow more flexibility in how continuous-space models are discretised and encoded in finite state games.

9.3 Outlook

This thesis has developed a comprehensive framework to synthesise strategies for the control of autonomous systems. Compared to traditional engineering disciplines, strategy synthesis from quantitative temporal logic objectives is currently in relative infancy. However, with vital infrastructure depending increasingly on the reliable operation of computer systems, we believe that formal strategy synthesis has the potential to fundamentally transform the design, certification, and operational processes, and will constitute a key element of the digital future.

Bibliography

[1] R. Alur and T.A. Henzinger. Reactive modules. *FMSD*, 15(1):7–48, 1999.

[2] R. Alur, T.A. Henzinger, F.Y.C. Mang, S. Qadeer, S.K. Rajamani, and S. Tasiran. MOCHA: Modularity in model checking. In *CAV*, pages 521–525. Springer, 1998.

[3] R. Bagnara, P.M. Hill, and E. Zaffanella. The Parma Polyhedra Library: Toward a complete set of numerical abstractions for the analysis and verification of hardware and software systems. *Science of Compututer Programming*, 72(1–2):3–21, 2008.

[4] C. Baier, C. Dubslaff, and S. Klüppelholz. Trade-off analysis meets probabilistic model checking. In *LICS*, volume 1, pages 1–10. ACM/IEEE, 2014.

[5] C. Baier, M. Größer, and F. Ciesinski. Quantitative analysis under fairness constraints. In *ATVA*, volume 5799 of *LNCS*, pages 135–150. Springer, 2009.

[6] C. Baier, M. Größer, M. Leucker, B. Bollig, and F. Ciesinski. Controller synthesis for probabilistic systems (extended abstract). In *IFIP TCS*, volume 155, pages 493–506. Springer, 2004.

[7] C. Baier and J.P. Katoen. *Principles of Model Checking*. MIT press, 2008.

[8] C. Baier, J. Klein, and S. Klüppelholz. A compositional framework for controller synthesis. In *CONCUR*, volume 6901 of *LNCS*, pages 512–527. Springer, 2011.

[9] B. Barbot. *Acceleration for Statistical Model Checking*. Thèse de doctorat, Laboratoire Spécification et Vérification, ENS Cachan, France, 2014.

[10] N. Basset, M. Kwiatkowska, U. Topcu, and C. Wiltsche. Strategy synthesis for stochastic games with multiple long-run objectives. In *TACAS*, volume 9035 of *LNCS*, pages 256–271. Springer, 2015.

[11] N. Basset, M. Kwiatkowska, and C. Wiltsche. Compositional controller synthesis for stochastic games. In *CONCUR*, volume 8704 of *LNCS*, pages 173–187. Springer, 2014.

[12] N. Basset, M. Kwiatkowska, and C. Wiltsche. Compositional strategy synthesis for stochastic games, 2015. (in preparation).

[13] C. Belta, A. Bicchi, M. Egerstedt, E. Frazzoli, E. Klavins, and G.J. Pappas. Symbolic planning and control of robot motion. *Robotics & Automation Magazine*, 14(1):61–70, 2007.

[14] P.D. Bertsekas. *Dynamic programming and optimal control*, volume 2. Athena Scientific, 1995.

[15] D. Blackwell. An analog of the minimax theorem for vector payoffs. *Pacific Journal of Mathematics*, 6(1):1–8, 1956.

[16] R. Bloem, K. Chatterjee, T.A. Henzinger, and B. Jobstmann. Better quality in synthesis through quantitative objectives. In *CAV*, volume 5643 of *LNCS*, pages 140–156. Springer, 2009.

[17] R. Bloem, K. Greimel, T. Henzinger, and B. Jobstmann. Synthesizing robust systems. In *FMCAD*, pages 85–92. IEEE, 2009.

[18] T. Brázdil, V. Brožek, K. Chatterjee, V. Forejt, and A. Kučera. Two views on multiple mean-payoff objectives in Markov decision processes. *LMCS*, 10(4), 2014.

[19] T. Brázdil, V. Brožek, V. Forejt, and A. Kučera. Stochastic games with branching-time winning objectives. In *LICS*, pages 349–358. ACM/IEEE, 2006.

[20] T. Brázdil, K. Chatterjee, V. Forejt, and A. Kucera. Trading performance for stability in Markov decision processes. In *LICS*, pages 331–340. ACM/IEEE, 2013.

[21] T. Brázdil, K. Chatterjee, V. Forejt, and A. Kučera. MultiGain: A controller synthesis tool for MDPs with multiple mean-payoff objectives. In *TACAS*, volume 9035 of *LNCS*, pages 181–187. Springer, 2015.

[22] G. Brown, M. Carlyle, J. Salmerón, and K. Wood. Defending critical infrastructure. *Interfaces*, 36(6):530–544, 2006.

[23] J.R. Büchi and L.H. Landweber. Solving sequential conditions by finite-state strategies. *Transactions of the American Mathematical Society*, 138(295–311):5, 1969.

[24] M. Campbell, M. Egerstedt, J. P. How, and R. M. Murray. Autonomous driving in urban environments: approaches, lessons and challenges. *Philosophical Transactions A*, 368(1928):4649–4672, 2010.

[25] K. Chatterjee and L. Doyen. Energy parity games. In *ICALP*, volume 6199 of *LNCS*, pages 599–610. Springer, 2010.

[26] K. Chatterjee, L. Doyen, T.A. Henzinger, and J.F. Raskin. Generalized mean-payoff and energy games. In *FSTTCS*, volume 8 of *LIPIcs*, pages 505–516. Schloss Dagstuhl, 2010.

[27] K. Chatterjee and M. Henzinger. Faster and dynamic algorithms for maximal end-component decomposition and related graph problems in probabilistic verification. In *SODA*, pages 1318–1336. ACM-SIAM, 2011.

[28] K. Chatterjee and T.A. Henzinger. Assume-guarantee synthesis. In *TACAS*, volume 4424 of *LNCS*, pages 261–275. Springer, 2007.

[29] K. Chatterjee and T.A. Henzinger. Reduction of stochastic parity to stochastic mean-payoff games. *Information Processing Letters*, 106(1):1–7, 2008.

[30] K. Chatterjee, T.A. Henzinger, R. Jhala, and R. Majumdar. Counterexample-guided planning. In *UAI*, pages 104–111. AUAI Press, 2005.

[31] K. Chatterjee, T.A. Henzinger, B. Jobstmann, and R. Singh. QUASY: Quantitative synthesis tool. In *TACAS*, volume 6605 of *LNCS*, pages 267–271. Springer, 2011.

[32] K. Chatterjee, M. Jurdziński, and T. Henzinger. Simple stochastic parity games. In *CSL*, volume 2803 of *LNCS*, pages 100–113. Springer, 2003.

[33] K. Chatterjee, M. Jurdziński, and T.A. Henzinger. Quantitative stochastic parity games. In *Symposium on Discrete Algorithms*, pages 121–130. SIAM, 2004.

[34] K. Chatterjee, Z. Komárková, and J. Křetínský. Unifying two views on multiple mean-payoff objectives in Markov decision processes. In *LICS*, pages 244–256. ACM/IEEE, 2015.

[35] K. Chatterjee, R. Majumdar, and T.A. Henzinger. Markov decision processes with multiple objectives. In *STACS*, volume 3884 of *LNCS*, pages 325–336. Springer, 2006.

[36] K. Chatterjee, M. Randour, and J.F. Raskin. Strategy synthesis for multi-dimensional quantitative objectives. *Acta Informatica*, 51(3–4):129–163, 2014.

[37] K. Chatterjee and Y. Velner. Hyperplane separation technique for multidimensional mean-payoff games. In *CONCUR*, volume 8052 of *LNCS*, pages 500–515. Springer, 2013.

[38] T. Chen, V. Forejt, M. Kwiatkowska, D. Parker, and A. Simaitis. Automatic verification of competitive stochastic systems. *FMSD*, 43(1):61–92, 2013.

[39] T. Chen, V. Forejt, M. Kwiatkowska, D. Parker, and A. Simaitis. PRISM-games: A model checker for stochastic multi-player games. In *TACAS*, volume 7795 of *LNCS*, pages 185–191. Springer, 2013.

[40] T. Chen, V. Forejt, M. Kwiatkowska, A. Simaitis, A. Trivedi, and M. Ummels. Playing stochastic games precisely. In *CONCUR*, volume 7454 of *LNCS*, pages 348–363, 2012.

[41] T. Chen, V. Forejt, M. Kwiatkowska, A. Simaitis, and C. Wiltsche. On stochastic games with multiple objectives. In *MFCS*, volume 8087 of *LNCS*, pages 266–277. Springer, 2013.

[42] T. Chen, V. Forejt, M. Kwiatkowska, A. Simaitis, and C. Wiltsche. On stochastic games with multiple objectives. Technical report, University of Oxford, 2013. Technical Report CS-RR-13-06.

[43] T. Chen, M. Kwiatkowska, A. Simaitis, and C. Wiltsche. Synthesis for multi-objective stochastic games: An application to autonomous urban driving. In *QEST*, volume 8054 of *LNCS*, pages 322–337. Springer, 2013.

[44] Y.F. Chen, E.M. Clarke, A. Farzan, M.H. Tsai, Y.K. Tsay, and B.Y. Wang. Automated assume-guarantee reasoning through implicit learning. In *CAV*, volume 6174 of *LNCS*, pages 511–526. Springer, 2010.

[45] C.H. Cheng, A. Knoll, M. Luttenberger, and C. Buckl. GAVS+: An open platform for the research of algorithmic game solving. In *TACAS*, volume 6605 of *LNCS*, pages 258–261. Springer, 2011.

[46] L. Cheung, N. Lynch, R. Segala, and F. Vaandrager. Switched PIOA: Parallel composition via distributed scheduling. *TCS*, 365(1–2):83–108, 2006.

[47] A. Church. Applications of recursive arithmetic to the problem of circuit synthesis. *Summaries of the Summer Institute of Symbolic Logic*, 1:3–50, 1957.

[48] E.M. Clarke, D.E. Long, and K. L. McMillan. Compositional model checking. In *LICS*, pages 353–362, 1989.

[49] J.L. Cohon. *Multiobjective programming and planning*. Courier Corporation, 2013.

[50] J. Cristau, C. David, and F. Horn. How do we remember the past in randomised strategies? *EPTCS*, 25:30–39, 2010.

[51] DARPA. Urban Challenge. http://archive.darpa.mil/grandchallenge/, [Online; accessed 19-August-2015].

[52] B.A. Davey and H.A. Priestley. *Introduction to lattices and order*. Cambridge University Press, 1990.

[53] L. De Alfaro. *Formal verification of probabilistic systems*. PhD thesis, Stanford University, 1997.

[54] L. de Alfaro and T.A. Henzinger. Interface automata. *SIGSOFT Software Engineering Notes*, 26(5):109–120, 2001.

[55] L. de Alfaro, T.A. Henzinger, and R. Jhala. Compositional methods for probabilistic systems. In *CONCUR*, volume 2154 of *LNCS*, pages 351–365. Springer, 2001.

[56] L. De Alfaro and R. Majumdar. Quantitative solution of omega-regular games. In *Symposium on Theory of Computing*, pages 675–683. ACM, 2001.

[57] Automation Direct. Part number AD-SSR610-AC-280A, Relays and Timers, Book 2 (14.1), eRL-45, 2014.

[58] K. Dräger, V. Forejt, M.Z. Kwiatkowska, D. Parker, and M. Ujma. Permissive controller synthesis for probabilistic systems. In *TACAS*, volume 8413 of *LNCS*, pages 531–546. Springer, 2014.

[59] A. Ehrenfeucht and J. Mycielski. Positional strategies for mean payoff games. *International Journal of Game Theory*, 8(2):109–113, 1979.

[60] K. Etessami, M. Kwiatkowska, M.Y. Vardi, and M. Yannakakis. Multi-objective model checking of Markov decision processes. *LMCS*, 4(8):1–21, 2008.

[61] A. Fehnker and F. Ivančić. Benchmarks for hybrid systems verification. In *HSCC*, volume 2993 of *LNCS*, pages 326–341. Springer, 2004.

[62] L. Feng, M. Kwiatkowska, and D. Parker. Automated learning of probabilistic assumptions for compositional reasoning. In *FASE*, volume 6603 of *LNCS*, pages 2–17. Springer, 2011.

[63] L. Feng, C. Wiltsche, L. Humphrey, and U. Topcu. Controller synthesis for autonomous systems interacting with human operators. In *ICCPS*, pages 70–79. ACM, 2015.

[64] J. Filar and K. Vrieze. *Competitive Markov decision processes*. Springer, 1996.

[65] E. Filiot, N. Jin, and J.F. Raskin. Antichains and compositional algorithms for LTL synthesis. *Formal Methods in System Design*, 39(3):261–296, 2011.

[66] A. Fisher, C.A. Jacobson, E.A. Lee, R.M. Murray, A. Sangiovanni-Vincentelli, and E. Scholte. Industrial cyber-physical systems–iCyPhy. In *Complex Systems Design & Management*, pages 21–37. Springer, 2014.

[67] V. Forejt, M. Kwiatkowska, G. Norman, D. Parker, and H. Qu. Quantitative multi-objective verification for probabilistic systems. In *TACAS*, volume 6605 of *LNCS*, pages 112–127. Springer, 2011.

[68] V. Forejt, M. Kwiatkowska, and D. Parker. Pareto curves for probabilistic model checking. In *ATVA*, volume 7561 of *LNCS*, pages 317–332. Springer, 2012.

[69] M. Gelderie. Strategy composition in compositional games. In *ICALP*, volume 7966 of *LNCS*, pages 263–274. Springer, 2013.

[70] S. Ghosh, R. Ramanujam, and S. Simon. Playing extensive form games in parallel. In *CLIMA*, volume 6245 of *LNCS*, pages 153–170. Springer, 2010.

[71] H. Gimbert and F. Horn. Solving simple stochastic tail games. In *SODA*, pages 847–862. ACM-SIAM, 2010.

[72] H. Gimbert and E. Kelmendi. Two-player perfect-information shift-invariant submixing stochastic games are half-positional. *arXiv preprint arXiv:1401.6575*, 2014.

[73] T.A. Henzinger. *The theory of hybrid automata*. Springer, 2000.

[74] H. Hermanns. *Interactive Markov chains: and the quest for quantified quality*. Springer, 2002.

[75] F. Horn. *Random Games*. PhD thesis, Université Denis Diderot - Paris 7 & Rheinisch-Westfälische Technische Hochschule Aachen, 2008.

[76] B. Huber, J. Rambau, and F. Santos. The Cayley trick, lifting subdivisions and the Bohne-Dress theorem on zonotopal tilings. *JEMS*, 2:179–198, 1999.

[77] L.R. Humphrey, E. Wolff, and U. Topcu. Formal specification and synthesis of mission plans for unmanned aerial vehicles. In *AAAI Spring Symposium*, 2014.

[78] G. Katz, D. Peled, and S. Schewe. Synthesis of distributed control through knowledge accumulation. In *CAV*, volume 6806 of *LNCS*, pages 510–525. Springer, 2011.

[79] A. Klenke. *Probability theory: a comprehensive course*. Springer, 2013.

[80] A. Komuravelli, C.S. Păsăreanu, and E.M. Clarke. Assume-guarantee abstraction refinement for probabilistic systems. In *CAV*, volume 7358 of *LNCS*, pages 310–326. Springer, 2012.

[81] H. Kress-Gazit, G.E. Fainekos, and G.J. Pappas. Where's Waldo? sensor-based temporal logic motion planning. In *ICRA*, pages 3116–3121. IEEE, 2007.

[82] T. Krieger. On Pareto equilibria in vector-valued extensive form games. *Mathematical Methods of Operations Research*, 58(3):449–458, 2003.

[83] O. Kupferman and M.Y. Vardi. Church's problem revisited. *Bulletin of Symbolic Logic*, 5(2):245–263, 1999.

[84] M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *CAV*, volume 6806 of *LNCS*, pages 585–591. Springer, 2011.

[85] M. Kwiatkowska, G. Norman, D. Parker, and H. Qu. Compositional probabilistic verification through multi-objective model checking. *Information and Computation*, 232:38–65, 2013.

[86] M. Kwiatkowska, D. Parker, and C. Wiltsche. PRISM-games 2.0: A tool for multi-objective strategy synthesis for stochastic games. In *TACAS*, 2016.

[87] D.A. Levin, Y. Peres, and E.L. Wilmer. *Markov chains and mixing times*. AMS, 2009.

[88] A.V. Lotov and K. Miettinen. Visualizing the Pareto frontier. In *Multiobjective optimization*, pages 213–243. Springer, 2008.

[89] Y. Lustig and M. Vardi. Synthesis from component libraries. *FoSSaCS*, 5504:395–409, 2009.

[90] N.A. Lynch and M.R. Tuttle. Hierarchical correctness proofs for distributed algorithms. In *Principles of Distributed Computing*, pages 137–151. ACM, 1987.

[91] L. MacDermed and C.L. Isbell. Solving stochastic games. In *NIPS*, pages 1186–1194. Curran Associates, Inc., 2009.

[92] R.T. Marler and J.S. Arora. Survey of multi-objective optimization methods for engineering. *Structural and Multidisciplinary Optimization*, 26(6):369–395, 2004.

[93] R.T. Marler and J.S. Arora. The weighted sum method for multi-objective optimization: new insights. *Structural and Multidisplinary Optimization*, 41(6):853–862, 2010.

[94] D.A. Martin. The determinacy of Blackwell games. *Journal of Symbolic Logic*, 63(4):1565–1581, 1998.

[95] J.F. Mertens and A. Neyman. Stochastic games. *International Journal of Game Theory*, 10(2):53–66, 1981.

[96] N. Meuleau, K.E. Kim, L.P. Kaelbling, and A.R. Cassandra. Solving POMDPs by searching the space of finite policies. In *UAI*, pages 417–426. Morgan Kaufmann, 1999.

[97] R.G. Michalko. Electrical starting, generation, conversion and distribution system architecture for a more electric vehicle, 2008. US Patent 7,439,634.

[98] S. Nain and M. Vardi. Synthesizing probabilistic composers. In *FoSSaCS*, volume 7213 of *LNCS*, pages 421–436. Springer, 2012.

[99] J. Nash. Non-cooperative games. *Annals of Mathematics*, pages 286–295, 1951.

[100] J. von Neumann. Zur Theorie der Gesellschaftsspiele. *Mathematische Annalen*, 100(1):295–320, 1928.

[101] A. Nilim and L. El Ghaoui. Robust control of Markov decision processes with uncertain transition matrices. *Operations Research*, 53(5):780–798, 2005.

[102] P. Nuzzo, H. Xu, N. Ozay, J.B. Finn, A.L. Sangiovanni-Vincentelli, R.M. Murray, A. Donzé, and S.A. Seshia. A contract-based methodology for aircraft electric power system design. *Access*, 2:1–25, 2014.

[103] OpenStreetMap. http://openstreetmap.org/, [Online; accessed 19-August-2015].

[104] N. Ozay, U. Topcu, R.M. Murray, and T. Wongpiromsarn. Distributed synthesis of control protocols for smart camera networks. In *ICCPS*, pages 45–54. IEEE, 2011.

[105] C.H. Papadimitriou and M. Yannakakis. On the approximability of trade-offs and optimal access of web sources. In *FOCS*, pages 86–92. IEEE, 2000.

[106] C.S. Păsăreanu, D. Giannakopoulou, M.G. Bobaru, J.M. Cobleigh, and H. Barringer. Learning to divide and conquer: applying the L* algorithm to automate assume-guarantee reasoning. *FMSD*, 32(3):175–205, 2008.

[107] N. Piterman, A. Pnueli, and Y. Sa'ar. Synthesis of reactive (1) designs. In *VMCAI*, volume 3855 of *LNCS*, pages 364–380. Springer, 2006.

[108] A. Pneuli and R. Rosner. Distributed reactive systems are hard to synthesize. In *FOCS*, pages 746–757. IEEE, 1990.

[109] A. Pnueli. In transition from global to modular temporal reasoning about programs. In *LMCS*, volume 13 of *NATO ASI Series*, pages 123–144. Springer, 1985.

[110] M.L. Puterman. *Markov decision processes: discrete stochastic dynamic programming*. Wiley-Interscience, 2009.

[111] C.S. Păsăreanu and D. Giannakopoulou. Towards a compositional SPIN. In *Model Checking Software*, volume 3925 of *LNCS*, pages 234–251. Springer, 2006.

[112] M. Randour, J.F. Raskin, and O. Sankur. Percentile queries in multi-dimensional Markov decision processes. In *CAV*, volume 9206 of *LNCS*, pages 123–139. Springer, 2015.

[113] G. Rennen, E.R. van Dam., and D. den Hertog. Enhancement of sandwich algorithms for approximating higher-dimensional convex Pareto sets. *Journal on Computing*, 23(4):493–517, 2011.

[114] R.T. Rockafellar. *Convex Analysis*. Princeton University Press, 1997.

[115] S.M. Ross. *Stochastic processes*, volume 2. John Wiley & Sons New York, 1996.

[116] O. Sankur. *Robustness in Timed Automata: Analysis, Synthesis, Implementation*. Thèse de doctorat, LSV, ENS Cachan, France, 2013.

[117] S. Schewe. Synthesis for probabilistic environments. *ATVA*, 4218:245–259, 2006.

[118] R. Segala. *Modelling and Verification of Randomized Distributed Real Time Systems*. PhD thesis, Massachusetts Institute of Technology, 1995.

[119] R. Segala and N. Lynch. Probabilistic simulations for probabilistic processes. *Nordic Journal of Computing*, 2(2):250–273, 1995.

[120] R. Segala and A. Turrini. Comparative analysis of bisimulation relations on alternating and non-alternating probabilistic models. In *QEST*, pages 44–53. IEEE, 2005.

[121] Lloyd S Shapley. Stochastic games. *PNAS*, 39(10):1095, 1953.

[122] E. Shieh, B. An, R. Yang, M. Tambe, C. Baldwin, J. DiRenzo, B. Maule, and G. Meyer. PROTECT: A deployed game theoretic system to protect the ports of the United States. In *AAMAS*, pages 13–20. IFAAMAS, 2012.

[123] N. Shimkin and A. Shwartz. Guaranteed performance regions in Markovian systems with competing decision makers. *Automatic Control*, 38(1):84–95, 1993.

[124] A. Simaitis. *Automatic Verification of Competitive Stochastic Systems*. PhD thesis, University of Oxford, 2013.

[125] M. Sinnett. 787 no-bleed systems: saving fuel and enhancing operational efficiencies. *Aero Quarterly*, pages 6–11, 2007.

[126] S.E.Z. Soudjani, C. Gevaerts, and A. Abate. FAUST^2: Formal Abstractions of Uncountable-STate STochastic processes. In *TACAS*, volume 9035 of *LNCS*, pages 272–286. Springer, 2015.

[127] M. Svorenova, I. Cerna, and C. Belta. Optimal control of MDPs with temporal logic constraints. In *CDC*, pages 3938–3943. IEEE, 2013.

[128] I. Tkachev and A. Abate. Formula-free finite abstractions for linear temporal verification of stochastic hybrid systems. In *HSCC*, pages 283–292. ACM, 2013.

[129] I. Tkachev, A. Mereacre, J.P. Katoen, and A. Abate. Quantitative automata-based controller synthesis for non-autonomous stochastic hybrid systems. In *HSCC*, pages 293–302. ACM, 2013.

[130] S. Topaloglu. A multi-objective programming model for scheduling emergency medicine residents. *Computers & Industrial Engineering*, 51(3):375–388, 2006.

[131] C. Urmson, J. Anhalt, D. Bagnell, C. Baker, R. Bittner, M. N. Clark, J. Dolan, D. Duggins, T. Galatali, C. Geyer, et al. Autonomous driving in urban environments: Boss and the urban challenge. *Journal of Field Robot.*, 25(8):425–466, 2008.

[132] Y. Velner. Robust multidimensional mean-payoff games are undecidable. In *FoSSaCS*, volume 9034 of *LNCS*, pages 312–327. Springer, 2015.

[133] Y. Velner and A. Rabinovich. Church synthesis problem for noisy input. In *FoSSaCS*, volume 6604 of *LNCS*, pages 275–289. Springer, 2011.

[134] C. von Essen and D. Giannakopoulou. Analyzing the next generation airborne collision avoidance system. In *TACAS*, volume 8413 of *LNCS*, pages 620–635. Springer, 2014.

[135] C. von Essen and B. Jobstmann. Synthesizing efficient controllers. In *VMCAI*, volume 7148 of *LNCS*, pages 428–444. Springer, 2012.

[136] C. Weibel. *Minkowski sums of polytopes: combinatorics and computation*. PhD thesis, École Polytechnique Fédérale de Lausanne, 2007.

[137] T. Wongpiromsarn, U. Topcu, and R. M. Murray. Receding horizon temporal logic planning. *Automatic Control*, 57(11):2817–2830, 2012.

[138] T. Wongpiromsarn, U. Topcu, N. Ozay, H. Xu, and R.M. Murray. TuLiP: a software toolbox for receding horizon temporal logic planning. In *HSCC*, pages 313–314. ACM, 2011.

[139] S.H. Wu, S.A. Smolka, and E.W. Stark. Composition and behaviors of probabilistic I/O automata. In *Theoretical Computer Science*, pages 513–528. Elsevier, 1994.

[140] H. Xu, U. Topcu, and R.M. Murray. A case study on reactive protocols for aircraft electric power distribution. In *CDC*, pages 1124–1129. IEEE, 2012.

[141] Z. Yin, A.X. Jiang, M. Tambe, C. Kiekintveld, K. Leyton-Brown, T. Sandholm, and J.P. Sullivan. TRUSTS: Scheduling randomized patrols for fare inspection in transit systems using game theory. *AI Magazine*, 33(4):59, 2012.

[142] L. Zadeh. Optimality and non-scalar-valued performance criteria. *Automatic Control*, 8(1):59–60, 1963.

[143] J. Zhao. The equilibria of a multiple objective game. *International Journal of Game Theory*, 20(2):171–182, 1991.

[144] U. Zwick and M. Paterson. The complexity of mean payoff games on graphs. *Theoretical Computer Science*, 158(1):343–359, 1996.

Appendix A

Proofs

A.1 Proof of Lemma 3.4

Proof. Fix a game $\mathcal{G}$ and strategies π, σ. We use induction on the length i of paths $\lambda \in \Omega_{\mathcal{G}}^{\mathsf{fin}}$. In the base case, for $i = 1$, we have that $\mathbb{P}_{\mathcal{G}}^{(\pi,\sigma)}(s_{\mathsf{init}}) = 1 = \mathbb{P}_{\mathcal{G}}^{\pi,\sigma}(s_{\mathsf{init}})$. Now assume $\mathbb{P}_{\mathcal{G}}^{(\pi,\sigma)}(\underline{\lambda}s) = \mathbb{P}_{\mathcal{G}}^{\pi,\sigma}(\underline{\lambda}s)$ for all paths $\underline{\lambda}s \in \Omega_{\mathcal{G}}^{\mathsf{fin}}$ of odd length $i \geq 1$, and consider a path $\underline{\lambda}s(a,\mu)t$ of length $i+2$. Let $\mathcal{D} = \mathcal{G}^{(\pi,\sigma)}$. We have that

$$\mathbb{P}_{\mathcal{G}}^{(\pi,\sigma)}(\underline{\lambda}s(a,\mu)) = \sum_{\kappa \text{ s.t. } \mathsf{path}_{\mathcal{G}}(\kappa)=\underline{\lambda}s(a,\mu)} \mathbb{P}_{\mathcal{D}}(\kappa) \tag{A.1}$$

from the definition of the measure $\mathbb{P}_{\mathcal{G}}^{(\pi,\sigma)}$. The measure $\mathbb{P}_{\mathcal{D}}$ is defined such that $\mathbb{P}_{\mathcal{D}}(\kappa) = \prod_{l=0}^{i} \Delta_{\mathcal{D}}(\kappa_l, \kappa_{l+1})$, where κ_l is the lth element on the path κ in $\mathcal{D}$. From the definition of the path mapping $\mathsf{path}_{\mathcal{G}}$, we have $\kappa_{i-1} = (s, \mathfrak{m}_{i-1}, \mathfrak{n}_{i-1})$, $\kappa_i = ((a,\mu), \mathfrak{m}_i, \mathfrak{n}_i)$, and $\kappa_{i+1} = (t, \mathfrak{m}_{i+1}, \mathfrak{n}_{i+1})$. Hence, the right hand side of (A.1) equals

$$\sum_{\substack{\underline{\kappa} \text{ s.t. } \mathsf{path}_{\mathcal{G}}(\underline{\kappa}) = \underline{\lambda}s \\ \mathfrak{m}_i, \mathfrak{m}_{i+1}, \mathfrak{n}_i, \mathfrak{n}_{i+1}}} \mathbb{P}_{\mathcal{D}}(\underline{\kappa}) \cdot \Delta_{\mathcal{D}}(\kappa_{i-1}, \kappa_i) \cdot \Delta_{\mathcal{D}}(\kappa_i, \kappa_{i+1}).$$

The memory of π and σ after seeing history $\underline{\lambda}s$ is distributed according to $\mathfrak{d}^{\pi}_{\underline{\lambda}s}$ and $\mathfrak{d}^{\sigma}_{\underline{\lambda}s}$, respectively. Since $\underline{\lambda}s$ is fixed, the right hand side of (A.1) further becomes

$$\sum_{\underline{\kappa} \text{ s.t. } \mathsf{path}_{\mathcal{G}}(\underline{\kappa})=\underline{\lambda}s} \mathbb{P}_{\mathcal{D}}(\underline{\kappa}) \cdot \sum_{\substack{\mathfrak{m}_{i-1}, \mathfrak{m}_i, \mathfrak{m}_{i+1} \\ \mathfrak{n}_{i-1}, \mathfrak{n}_i, \mathfrak{n}_{i+1}}} \mathfrak{d}^{\pi}_{\underline{\lambda}s}(\mathfrak{m}_{i-1}) \cdot \mathfrak{d}^{\sigma}_{\underline{\lambda}s}(\mathfrak{n}_{i-1}) \cdot \Delta_{\mathcal{D}}(\kappa_{i-1}, \kappa_i) \cdot \Delta_{\mathcal{D}}(\kappa_i, \kappa_{i+1}).$$

From the induction hypothesis we substitute $\sum_{\underline{\kappa} \text{ s.t. } \mathsf{path}_{\mathcal{G}}(\underline{\kappa})=\underline{\lambda}s} \mathbb{P}_{\mathcal{D}}(\underline{\kappa})$ by $\mathbb{P}^{\pi,\sigma}_{\mathcal{G}}(\underline{\lambda}s)$. Let $\mathcal{D}' = \mathcal{G}^{\pi,\sigma}$. We can then further substitute $\mathbb{P}^{\pi,\sigma}_{\mathcal{G}}(\underline{\lambda}s) = \sum_{\underline{\kappa'} \text{ s.t. } \mathsf{path}_{\mathcal{G}}(\underline{\kappa'})=\underline{\lambda}s} \mathbb{P}_{\mathcal{D}'}(\underline{\kappa'})$, from the definition of the measure $\mathbb{P}_{\mathcal{D}'}$.

Consider the case $s \in S_\Diamond$. From the definition of $\mathcal{D}$ (Definition 3.5), we have $\sum_{\mathfrak{m}_i,\mathfrak{n}_i} \Delta_{\mathcal{D}}(\kappa_{i-1},\kappa_i) = \pi_{\mathsf{c}}(t,\mathfrak{m}_{n-1})(a,\mu)$ and $\sum_{\mathfrak{m}_{i+1},\mathfrak{n}_{i+1}} \Delta_{\mathcal{D}}(\kappa_i,\kappa_{i+1}) = \mu(t)$. Moreover, the $\mathfrak{d}^{\sigma}_{\underline{\lambda}s}(\mathfrak{n}_{i-1})$ terms sum to one. The right hand side of (A.1) becomes

$$\sum_{\underline{\kappa'} \text{ s.t. } \mathsf{path}_{\mathcal{G}}(\underline{\kappa'})=\underline{\lambda}s} \mathbb{P}_{\mathcal{D}'}(\underline{\kappa'}) \cdot \sum_{\mathfrak{m}_{i-1} \text{ in } \kappa_{i-1}} \mathfrak{d}^{\pi}_{\underline{\lambda}s}(\mathfrak{m}_{i-1}) \cdot \pi_{\mathsf{c}}(s,\mathfrak{m}_{i-1})(a,\mu) \cdot \mu(t).$$

We now consider the states along paths κ' of $\mathcal{D}'$, where we denote κ'_l the lth element along κ' (starting at $l = 0$). To induce $\mathcal{D}'$, Definition 3.5 is applied twice, and for notational convenience in the following, we omit the path mappings.

(K1) If $|\underline{\lambda}| > 0$, we have $\kappa'_{i-2} = (\nu^{\pi}_{\underline{\lambda}})^{\sigma}_{\underline{\lambda}}$, for the distribution ν in $\mathsf{last}(\underline{\lambda})$, and if $|\underline{\lambda}| = 0$, we have $\kappa'_{i-2} = (\nu^{\pi})^{\sigma}$ with $\nu = s_{\mathsf{init}}$. We have $\Delta_{\mathcal{D}'}(\kappa'_{i-2},\kappa'_{i-1}) = \nu(s) \cdot \sum_{\mathfrak{m}_{i-1}} \mathfrak{d}^{\pi}_{\underline{\lambda}s}(\mathfrak{m}_{i-1}) \cdot \pi_{\mathsf{c}}(s,\mathfrak{m}_{i-1})(a,\mu)$. When summing over $\underline{\kappa'}$ such that $\mathsf{path}_{\mathcal{G}}(\underline{\kappa'}) = \underline{\lambda}s$, the summation over the outgoing moves of s is included. To be able to fix the move (a,μ) we let

$$\mathbb{P}_{\mathcal{D}'}((a,\mu)|\underline{\lambda}s) \stackrel{\text{def}}{=} \mathbb{P}_{\mathcal{D}'}(s(a,\mu)|\underline{\lambda})/\mathbb{P}_{\mathcal{D}'}(s|\underline{\lambda}) = \Delta_{\mathcal{D}'}(\kappa'_{i-2},\kappa'_{i-1})/\nu(s).$$

(K2) From the states κ'_l of $\mathcal{D}'$ with even indices l, there is only a single outgoing move, as they originate from player states in $\mathcal{G}$. Thus, $\Delta_{\mathcal{D}'}(\kappa'_{i-1},\kappa'_i) = 1$.

(K3) We have $\kappa'_i = (a, (\mu^{\pi}_{\mathfrak{d}_{\underline{\lambda}s(a,\mu)}})^{\sigma}_{\mathfrak{d}_{\underline{\lambda}s(a,\mu)}})$, and hence $\sum_{(b,\nu)\in\Delta(t)} \Delta_{\mathcal{D}'}(\kappa'_i,\kappa'_{i+1}) = \mu(t)$.

Applying (K1)–(K3), the right hand side of (A.1) becomes

$$\sum_{\underline{\kappa'} \text{ s.t. } \mathsf{path}_{\mathcal{G}}(\underline{\kappa'})=\underline{\lambda}s} \mathbb{P}_{\mathcal{D}'}(\underline{\kappa'}) \cdot \mathbb{P}_{\mathcal{D}'}((a,\mu)|\underline{\lambda}s) \cdot \Delta_{\mathcal{D}'}(\kappa'_{i-1},\kappa'_i) \cdot \left(\sum_{(b,\nu)\in\Delta(t)} \Delta_{\mathcal{D}'}(\kappa'_i,\kappa'_{i+1}) \right).$$

Redistributing the summations, and noting that $\underline{\lambda}s(a,\mu)t$ is fixed, this equals

$$\sum_{\kappa' \text{ s.t. } \mathsf{path}_{\mathcal{G}}(\kappa')=\underline{\lambda}s(a,\mu)} \prod_{l=1}^{i} \Delta_{\mathcal{D}'}(\kappa'_l,\kappa'_{l+1}) = \sum_{\kappa' \text{ s.t. } \mathsf{path}_{\mathcal{G}}(\kappa')=\underline{\lambda}s(a,\mu)} \mathbb{P}_{\mathcal{D}'}(\kappa') = \mathbb{P}^{\pi,\sigma}_{\mathcal{G}}(\underline{\lambda}s(a,\mu)),$$

concluding the induction step for $s \in S_\Diamond$. The case for $s \in S_\Box$ is symmetric. The equivalence between the path distributions holds for infinite paths due to Carathéodory's extension theorem (Theorem 1.53 of [79]). □

A.2 Proof of Proposition 5.1

Proof. The proof method is based on similar results in [36, 124]. Consider the game $\mathcal{G}$ in Figure 5.4 with objective $\mathsf{Pmp}(\vec{r})(\vec{0})$. From s_0, when Player 2 chooses a sequence w of actions with $|w| \leq n+1$, the total rewards are shifted by the vector $-(\alpha_w, 2^{|w|} - 1 - \alpha_w)$, where $\alpha_w \stackrel{\text{def}}{=} \sum_{j=1}^{|w|} \delta_{w_j=\mathsf{a}} 2^{j-1}$ is the number corresponding to the binary word w represented with the least significant bit first, with a coding for 1 and b for 0.

Exponential memory DU strategy. We show that there is a winning DU strategy π for Player 1 with exponential memory $\mathfrak{M} \stackrel{\text{def}}{=} \{\mathsf{a}, \mathsf{b}\}^{n+1}$, which at state s_{n+1} plays the distribution ν_w defined by $\nu_w(\mathsf{a}) \stackrel{\text{def}}{=} \frac{\alpha_w}{2^{n+1}-1}$ and $\nu_w(\mathsf{b}) \stackrel{\text{def}}{=} 1 - \nu_w(\mathsf{a})$, where $w \in \mathfrak{M}$ is the current memory, determining α_w. This strategy compensates the shift incurred while going through the Player 2 states, and hence, for every loop, the expected total reward is $(0, 0)$. Thus also the overall average reward is $(0, 0)$. As the strategy π has finite memory, the induced PA $\mathcal{G}^\pi$ is finite, and it suffices to consider MD strategies for Player 2 in $\mathcal{G}^\pi$, see Lemma 3.7. Let R_k be the random variable on the total reward of the kth loop. The random variables $(R_k)_{k\geq 0}$ are independent identically distributed and of expectation zero, and we apply the strong law of large numbers to obtain that $1/N \sum_{k=0}^{K} R_k$ converges almost surely towards the common mean 0. Hence, π is winning for almost sure convergence.

No sub-exponential DU strategy. We show that every finite DU strategy achieving $\mathsf{Pmp}(\vec{r})$ requires at least exponential memory. Consider a finite DU strategy π with less than $2^{n+1}-1$ memory elements. We show that it loses against some finite strategy σ. For every memory element $\mathfrak{m} \in \mathfrak{M}$, there exist at least two distinct sequences $w^1_\mathfrak{m}$ and $w^2_\mathfrak{m}$ such that the memory updated from $\mathfrak{m}$ is the same after seeing either $w^1_\mathfrak{m}$ or $w^2_\mathfrak{m}$, denoted $f(\mathfrak{m})$, and such that $\mathsf{rew}(w^1_\mathfrak{m}) \geq \mathsf{rew}(w^2_\mathfrak{m}) + 1$ for r_1. Consider the finite memory strategy σ^1 (respectively σ^2) that simulates the deterministic memory of π and plays the actions in $w^1_\mathfrak{m}$ (respectively $w^2_\mathfrak{m}$) from s_0 and memory $\mathfrak{m}$. The strategy π reacts to $f(\mathfrak{m})$ at state s_{n+1}, and so compensates either for $w^1_\mathfrak{m}$ or $w^2_\mathfrak{m}$. Let $\mathcal{D}_i \stackrel{\text{def}}{=} \mathcal{G}^{\pi,\sigma^i}$. We extend the reasoning to K loops as follows. For pairwise associated sequences $w^i = w^i_{\mathfrak{m}_1} l_1 w^i_{\mathfrak{m}_2} l_2 \cdots w^i_{\mathfrak{m}_K} l_K$ with $i \in \{1, 2\}$, where π plays l_k after the kth loop, it holds that $\mathsf{rew}(r_1)(w^1) \geq \mathsf{rew}(r_1)(w^2) + K$ and $\mathbb{P}_{\mathcal{D}_1}(w^1) = \mathbb{P}_{\mathcal{D}_2}(w^2)$. Hence the average reward in the two DTMCs are separated by $1/L$ where L is the length of a loop. Hence, if π wins against σ^1, then $\mathbb{P}_{\mathcal{D}_1}(\mathsf{mp}(r_1) = 0) = \mathbb{P}_{\mathcal{D}_1}(\mathsf{mp}(r_2) = 0) = 1$, and hence, $\mathbb{P}_{\mathcal{D}_2}(\mathsf{mp}(r_1) \leq -1/L) = 1$. The strategy π loses against σ^1 or σ^2.

Linear memory SU strategy. We now show how the distribution ν_w can be simulated by an SU strategy π that contains only $2(n+1)$ memory elements. Let $\mathfrak{M} \stackrel{\text{def}}{=} \bigcup_{i=0}^{n+1} \{\mathsf{a}_i, \mathsf{b}_i\}$, and let $\pi_{\mathsf{c}}(s_{n+1}, l_{n+1}) \stackrel{\text{def}}{=} l$ for $l \in \{\mathsf{a}, \mathsf{b}\}$, that is, l_i is the memory at state s_i corresponding intuitively to the intention of Player 1 to play the action l. We denote by $\mathbb{P}(l_i|w)$ the probability of Player 1 being in memory l_i after having read the sequence w of length i, starting from s_0.

We now inductively define a memory update function such that, for $i \leq n+1$ and $w \in \{\mathsf{a}, \mathsf{b}\}^i$, $\mathbb{P}(\mathsf{a}_i|w) = \frac{\alpha_w}{2^i - 1}$ (and $\mathbb{P}(\mathsf{b}_i|w) = 1 - \mathbb{P}(\mathsf{a}_i|w)$), so that, in particular, when $i = n+1$, Player 1 chooses the next move according to the distribution ν_w. In the base case (when $i = 0$), $\mathbb{P}(\mathsf{a}_0|\varepsilon) = 1$ necessitates that the initial memory as well as the memory when returning after each loop to s_0 is $\pi_{\mathsf{d}}(s_0) \stackrel{\text{def}}{=} \pi_{\mathsf{u}}(l_{n+1}, l') \stackrel{\text{def}}{=} \mathsf{a}_0$. When going from s_i to s_{i+1} via an action q, the memory $l_i \in \{\mathsf{a}_i, \mathsf{b}_i\}$ in s_i is updated to l'_{i+1} in s_{i+1}, under the condition

$$\mathbb{P}(l'_{i+1}|wq) = \mathbb{P}(\mathsf{a}_i|w) \cdot \pi_{\mathsf{u}}(\mathsf{a}_i, q)(l'_{i+1}) + \mathbb{P}(\mathsf{a}_i|w) \cdot \pi_{\mathsf{u}}(\mathsf{b}_i, q)(l'_{i+1}). \tag{A.2}$$

Taking $l' = \mathsf{a}$ and taking q to be a or b in (A.2) gives as necessary conditions

$$\mathbb{P}(\mathsf{a}_{i+1}|w\mathsf{a}) = \frac{\alpha_w + 2^i}{2^{i+1} - 1} = \frac{\alpha_w}{2^i - 1} \cdot \pi_{\mathsf{u}}(\mathsf{a}_i, \mathsf{a})(\mathsf{a}_{i+1}) + \left(1 - \frac{\alpha_w}{2^i - 1}\right) \cdot \pi_{\mathsf{u}}(\mathsf{b}_i, \mathsf{a})(\mathsf{a}_{i+1}) \tag{A.3}$$

$$\mathbb{P}(\mathsf{a}_{i+1}|w\mathsf{b}) = \frac{\alpha_w}{2^{i+1} - 1} = \frac{\alpha_w}{2^i - 1} \cdot \pi_{\mathsf{u}}(\mathsf{a}_i, \mathsf{a})(\mathsf{a}_{i+1}) + \left(1 - \frac{\alpha_w}{2^i - 1}\right) \cdot \pi_{\mathsf{u}}(\mathsf{b}_i, \mathsf{b})(\mathsf{a}_{i+1}); \tag{A.4}$$

taking $l' = \mathsf{b}$ in (A.2) gives symmetric conditions.

We now define the memory update function according to these conditions. Define $\pi_{\mathsf{u}}(\mathsf{a}_i, \mathsf{a}) \stackrel{\text{def}}{=} \mathsf{a}_{i+1}$ and $\pi_{\mathsf{u}}(\mathsf{b}_i, \mathsf{b}) \stackrel{\text{def}}{=} \mathsf{b}_{i+1}$, following the intuition that there is no need to change the intention to play a or b, corresponding to the current memory a_i and b_i, respectively, when the intention is followed. Further, using the conditions in (A.3), we obtain, for $l, \bar{l} \in \{\mathsf{a}, \mathsf{b}\}$ with $\bar{l} \neq l$ that $\pi_{\mathsf{u}}(l_i, \bar{l})(l_{i+1}) \stackrel{\text{def}}{=} \frac{2^i - 1}{2^{i+1} - 1}$ and $\pi_{\mathsf{u}}(l_i, \bar{l})(\bar{l}_{i+1}) \stackrel{\text{def}}{=} \frac{2^i}{2^{i+1} - 1}$. We thus have defined π so that at s_{n+1} the choices it made according to ν_w. Then, as shown above, this strategy is winning. Moreover, π contains $2(n+1)$ memory elements, and is therefore exponentially smaller than the DU strategy described above, concluding the proof. □

Appendix B

Case Study Files

B.1 Autonomous Urban Driving

```
#!/usr/bin/python -O
from __future__ import division
import datetime, math, sys
import copy
import osm2graph

# default map:
filename = "islip"
goal = 46 # Middle Street (right) ... islip
init = 124 # Kidlington Road (top left) ... islip

if(len(sys.argv)>=2):
    if(int(sys.argv[1])==1):
        filename = "islip"
        goal = 46 # Middle Street (right) ... islip
        init = 124 # Kidlington Road (top left) ... islip
    elif(int(sys.argv[1])==2):
        filename = "charlton"
        goal = 33 # road right top ... charlton
        init = 43 # road left bottom ... charlton
    else:
        print "Unknown map ID specified ... usage:"
        print "Call generate.py n, where n stands for one of the following:"
        print "\t1 ... islip"
        print "\t2 ... charlton"
        print "If n is not specified, islip is taken as default."
        sys.exit(1);

smg = open("%s.prism" % (filename), "w");
prop = open("%s.props" % (filename), "w");
bash = open("%s.sh" % (filename), "w");

mapfile = "%s.osm" % (filename)

##########################################################################
# Prepare Graph                                                          #
##########################################################################

G = osm2graph.read_osm(mapfile)
usedvalues = {}
# calculate distances and assign to links
for e in G.edges(data=True):
    db = osm2graph.calculateDistanceAndBearing(G.node[e[0]]["data"], G.node[e[1]]["data"])
    # fill in relevant tags
    name = ''
    if "name" in e[2]["data"].tags:
        name = e[2]["data"].tags["name"]
    oneway = False
    if "oneway" in e[2]["data"].tags and e[2]["data"].tags["oneway"]=="yes":
        oneway = True
    lanes = 1
    if "lanes" in e[2]["data"].tags:
        lanes = e[2]["data"].tags["lanes"]
    value = 0; # give the road a value depending on its type
    values = {'motorway': 20,
              'motorway_link': 19,
              'trunk': 15,
              'trunk_link': 14,
              'primary': 10,
              'primary_link': 9,
              'secondary': 8,
              'secondary_link': 7,
              'tertiary': 6,
              'tertiary_link': 5,
              'living_street': 0.5,
```

```
                'pedestrian': 0, # never use
                'residential': 3,
                'unclassified': 1,
                'service': 1,
                'track': 0.2,
                'bus_guideway': 0, # never use
                'raceway': 0, # never use
                'road': 0.5}
    if "highway" in e[2]["data"].tags:
        if e[2]["data"].tags["highway"] in values:
            value = values[e[2]["data"].tags["highway"]]
            usedvalues[e[2]["data"].tags["highway"]] = True

    e[2]["data"] = {"dist": db[0], "init_bearing": db[1], "final_bearing": db[1], "oneway": oneway, "value":
         value, "name": name}

# first, remove zero-value edges (they are never used, e.g. train tracks)
to_remove = []
for e in G.edges(data=True):
    if e[2]["data"]["value"] == 0:
        to_remove.append(e)
for e in to_remove:
    G.remove_edge(e[0],e[1])

# collapse graph to not contain several edges where no intersection is
changed = True
while changed:
    changed = False
    for e in G.edges(data=True):
        edges = G.edges(e[1],data=True)
        if len(edges)==1 and not edges[0]==e:
            # need to prevent roads coming in to be ignored
            collapse = True
            for f in G.edges(data=True):
                if (f[0] != e[0]) and f[1]==e[1]:
                    collapse = False
            if(collapse):
                name = ''
                if e[2]["data"]["name"] is not "":
                    name = e[2]["data"]["name"]
                elif edges[0][2]["data"]["name"] is not "":
                    name = edges[0][2]["data"]["name"]
                oneway = e[2]["data"]["oneway"] or edges[0][2]["data"]["oneway"]
                # connect e to edges[0]
                init_bearing = e[2]["data"]["init_bearing"]
                final_bearing = edges[0][2]["data"]["final_bearing"]
                value = min(e[2]["data"]["value"], edges[0][2]["data"]["value"])
                G.add_edge(e[0],edges[0][1],attr_dict={'data': {'dist': e[2]["data"]["dist"]+edges[0][2]["data"][
                     "dist"], 'name': name, 'oneway': oneway, 'init_bearing': init_bearing, 'final_bearing':
                     final_bearing, 'value': value}})
                G.remove_edge(e[0],e[1])
                G.remove_edge(edges[0][0],edges[0][1])
                changed = True
                break

# for each edge also introduce the reverse edge
# moreover, assign which edge to go to for a u-turn
# note that lanes do not get assigned separate edges
for e in G.edges(data=True):
    if not e[2]["data"]["oneway"] and not G.has_edge(e[1],e[0]):
        new_data = copy.deepcopy(e[2])
        new_data["data"]["init_bearing"] = ((180 - e[2]["data"]["final_bearing"]) % 360)
        new_data["data"]["final_bearing"] = ((180 - e[2]["data"]["init_bearing"]) % 360)
        G.add_edge(e[1],e[0],attr_dict=e[2])

# identify which edge has how many successors
successors = {}
for e in G.edges(data=True):
    succ = 0;
    for f in G.edges([e[1]]):
        if (f[1],f[0]) == (e[0],e[1]): # no uturn in intersection
            continue
        succ = succ + 1
    if succ not in successors:
        successors[succ] = [e]
    else:
        new_succs = successors[succ]
        new_succs.append(e)
        successors[succ] = new_succs

# here identify which transitions are left turns, which ones right turns, and which ones go straight.
for e in G.edges(data=True):
    for f in G.edges([e[1]], data=True):
        # going from e to f
        angle = (e[2]["data"]["final_bearing"] - f[2]["data"]["init_bearing"] + 180) % 360
        if angle < 135: # right turn
            pass
        elif angle >= 135 and angle < 225: # straight
            pass
        else: # left
            pass

########################################################################
# Hazards and Reactions                                                #
########################################################################

hazards = ['pedestrian', 'obstacle', 'jam']
# control ocurrence probability, one for each hazard, the larger, the more likely
alphas = {'roadblock': 0.002,
          'pedestrian': 0.05,
          'obstacle': 0.02,
```

```
          'jam': 0.1}
reactions = {'roadblock': ['uturn'],
             'pedestrian' : ['brake', 'honk', 'changelane'],
             'obstacle': ['changelane', 'uturn'],
             'jam': ['honk', 'uturn']}
accident_prob = {('pedestrian', 'brake')      : 0.01,
                 ('pedestrian', 'honk')       : 0.04,
                 ('pedestrian', 'changelane') : 0.03,
                 ('obstacle', 'changelane')   : 0.02,
                 ('obstacle', 'uturn')        : 0.02,
                 ('jam', 'honk')              : 0.01,
                 ('jam', 'uturn')             : 0.02 }

# all possible reactions
reacts = set([])
for r in reactions.itervalues():
    reacts = reacts | set(r)
reacts = list(reacts)

def powerset(S):
    if len(S) <= 1:
        yield S
        yield []
    else:
        for s in powerset(S[1:]):
            yield [S[0]]+s
            yield s
# all possible combinations of hazards
haz = []
for i1 in xrange(0,len(hazards)):
    haz.append([hazards[i1]])
    for i2 in xrange(i1+1, len(hazards)):
        haz.append([hazards[i1],hazards[i2]])
haz.append([])

########################################################################
# Output PRISM Model                                                   #
########################################################################

# one field per edge
N = len(G.edges())
M = 3*(len(haz)-1) - 2*len(hazards)

# build a dictionary of edges and the associated ids, and do the same in reverse as well
edgeid = {}
idedge = {}
# also record lanes of each road
lanes = {}
i = 0
for e in G.edges(data=True):
    edgeid[(e[0],e[1])] = i
    idedge[i] = (e[0],e[1])
    if "lanes" in e[2]["data"]:
        lanes[i] = e[2]["data"]["lanes"]
    else:
        lanes[i] = 1
    i = i + 1

smg.write("// PRISM model for autonomous car case study\n")
smg.write("// Two player stochastic game with three objectives.\n")
smg.write("//\n")
smg.write("// This is a machine-generated file\n")
smg.write("// Author: Clemens Wiltsche\n")
today = datetime.date.today()
smg.write("// Generated: %s\n" % (today.strftime('%d/%m/%Y')))
smg.write("\n\n")
smg.write("smg\n\n")

# initialize player actions
smg.write("player p1\n")
first = True
for e in G.edges():
    i = edgeid[e]
    if(not first):
        smg.write(",\n")
    first = False
    smg.write("\t")
    # reactions
    for r in reacts:
        smg.write("[%s_%i], " % (r, i))
    # steering
    j = 0
    first_ = True
    onlyuturn = True
    for f in G.edges([e[1]]):
        if (f[1],f[0]) == (e[0],e[1]): # no uturn in intersection
            continue
        onlyuturn = False
        if(not first_):
            smg.write(", ")
        first_ = False
        smg.write("[move_%i_%i]" % (j, i))
        j = j + 1
    # alternatively, termination
    if onlyuturn or len(G.edges([e[1]]))==0:
        smg.write("[term_%i]" % (i))
smg.write("\nendplayer\n\n")

smg.write("player p2\n")
```

```
first = True
for e in G.edges():
    i = edgeid[e]
    if(not first):
        smg.write(",\n")
    first = False
    smg.write("\t")
    # hazard
    for h in hazards:
        smg.write("[%s_%i], " % (h, i))
    smg.write("[hazard_%i]" % (i))
smg.write(",\n\t[term]")
smg.write("\nendplayer\n\n")

# positions in topology
smg.write("const int POS_init = %i;\n" % (init))
smg.write("const int POS_goal = %i;\n" % (N))
smg.write("const int POS_term = %i;\n" % (N+1))
for e in G.edges(data=True):
    i = edgeid[(e[0],e[1])]
    name = ''
    if "name" in e[2]["data"]:
        name = e[2]["data"]["name"]
    if not "name" == '':
        smg.write("const int POS_%i = %i;\t// name: %s\n" % (i, i, name))
    else:
        smg.write("const int POS_%i = %i;\n" % (i, i))

smg.write("\n")

# the topology
for e in G.edges(data=True):
    ie = edgeid[(e[0],e[1])]
    if len(G.edges([e[1]]))==0:
        if ie==goal:
            smg.write("const int DEST_%i = POS_goal;\n" % (ie))
        else:
            smg.write("const int DEST_%i = POS_term;\n" % (ie))
    else:
        j = 0
        onlyone = False
        if len(G.edges([e[1]]))==1:
            onlyone = True
        for f in G.edges([e[1]], data=True):
            if ie != goal and (f[1],f[0]) == (e[0],e[1]): # no uturn in intersection
                if onlyone:
                    smg.write("const int DEST_%i = POS_term;\n" % (ie))
                continue
            i_f = edgeid[(f[0],f[1])]
            if ie==goal and onlyone:
                smg.write("const int DEST_%i = POS_goal;\n" % (ie))
            elif ie==goal:
                smg.write("const int DEST_%i_%i = POS_goal;\n" % (j, ie))
            else:
                name_to = ''
                if "name" in f[2]["data"]:
                    name_to = f[2]["data"]["name"]
                name_from = ''
                if "name" in e[2]["data"]:
                    name_from = e[2]["data"]["name"]
                if not name_to == '' and not name_from == '':
                    smg.write("const int DEST_%i_%i = POS_%i;\t// from %s to %s\n" % (j, ie, i_f, name_from,
                            name_to))
                elif not name_to == '':
                    smg.write("const int DEST_%i_%i = POS_%i;\t// to %s\n" % (j, ie, i_f, name_to))
                elif not name_from == '':
                    smg.write("const int DEST_%i_%i = POS_%i;\t// from %s\n" % (j, ie, i_f, name_from))
                else:
                    smg.write("const int DEST_%i_%i = POS_%i;\n" % (j, ie, i_f))
                j = j + 1
smg.write("\n")

# returns a distribution of hazard probabilities given the length of a road
def distr(length):
    # how many hazard combinations are there?
    dist = {} # nothing in distribution yet
    cumulative = 0.0
    empty_i = 0
    # fill distribution - one entry for each hazard combination
    for i in xrange(0,len(haz)):
        hs = haz[i]
        if hs==[]:
            empty_i = i
        else:
            Alpha = 1
            for h in hs:
                Alpha = Alpha*alphas[h]
            dist[i] = math.tanh(Alpha*length)/len(haz)
            cumulative = cumulative + dist[i]
    # probability of nothing happening
    dist[empty_i] = 1 - cumulative
    # return distribution
    return dist

# write hazard probabilities for each edge
for e in G.edges(data=True):
    ie = edgeid[(e[0],e[1])]
    d = distr(e[2]["data"]["dist"])
    for j in xrange(0, len(haz)):
        smg.write("const double DISTR_%i_%i = %f;\n" % (j, ie, d[j]))
```

```
smg.write("\n")

for e in G.edges(data=True):
    ie = edgeid[(e[0],e[1])]
    if (e[1],e[0]) in edgeid or lanes[ie]>1: # changelane possible if not single-lane oneway
        smg.write("const int CHANGELANE_0_%i = -1;\n" % (ie)) # accident first option
        smg.write("const int CHANGELANE_1_%i = -2;\n" % (ie)) # changing is second option
    else: # changelane illegal - go to violation state in any case
        smg.write("const int CHANGELANE_0_%i = -3;\n" % (ie))
        smg.write("const int CHANGELANE_1_%i = -3;\n" % (ie))

# write uturn reactions for each edge
for e in G.edges(data=True):
    ie = edgeid[(e[0],e[1])]
    if (e[1],e[0]) in edgeid: # uturn possible if not one-way (this also means highways)
        revedgeid = edgeid[(e[1],e[0])]
        smg.write("const int UTURN_0_%i = -1;\n" % (ie)) # accident first option
        smg.write("const int UTURN_1_%i = -2;\n" % (ie)) # turning is second option
        smg.write("const int UTURNPLAYER_%i = 1;\n" % (ie)) # if in -2, need player 1
        smg.write("const int REVEDGE_%i = %i;\n" % (ie, revedgeid)) # reverse edge exists
    else: # uturn illegal - go to violation state in any case
        smg.write("const int UTURN_0_%i = -3;\n" % (ie))
        smg.write("const int UTURN_1_%i = -3;\n" % (ie))
        smg.write("const int UTURNPLAYER_%i = 2;\n" % (ie)) # if in -3, need player 2
        smg.write("const int REVEDGE_%i = %i;\n" % (ie, ie)) # reverse edge doesn't exist but is irrelevant

smg.write("\nglobal p : [1..2] init 2;\n") # players
smg.write("global car_position : [0..%i] init POS_init;\n" % (N+2))
# special states:
# -1: accident (terminal)
# -2: sink - goto next edge
# -3: traffic violation (terminal)
# -4: braking
# -5: uturning
# -11: game terminated
smg.write("global s : [-11..%i] init 0;\n" % (M))

def subhazard(from_s, to_s, subhaz, k):
    if len(subhaz)==1:
        carreact(from_s, subhaz[0], k)
    else:
        for i in xrange(0, len(subhaz)):
            smg.write("\t[%s_%i] p=2 & s=%i & car_position=POS_%i-> (p'=1) & (s'=%i);\n" % (subhaz[i], k, from_s,
                 k, to_s[i]))

def carreact(from_s, hazz, k):
    for i in xrange(0, len(reactions[hazz])):
        if (reactions[hazz][i]=='brake'): # brake action is special (need to insert state to let time pass)
            smg.write("\t[brake_%i] p=1 & s=%i & car_position=POS_%i -> %f : (p'=2) & (s'=-1) + %f : (p'=1) & (s
                 '=-2);\n" % (k, from_s, k, accident_prob[(hazz,'brake')], 1.0-accident_prob[(hazz,'brake')]))
        elif(reactions[hazz][i]=='uturn'): # uturn action is special
            smg.write("\t[uturn_%i] p=1 & s=%i & car_position=POS_%i -> %f : (p'=2) & (s'=UTURN_0_%i) + %f : (p'=
                 UTURNPLAYER_%i) & (s'=UTURN_1_%i) & (car_position'=REVEDGE_%i);\n" % (k, from_s, k,
                 accident_prob[(hazz,'uturn')], k, 1.0-accident_prob[(hazz,'uturn')], k, k, k))
        elif(reactions[hazz][i]=='changelane'): # changelane action is special
            smg.write("\t[changelane_%i] p=1 & s=%i & car_position=POS_%i -> %f : (p'=2) & (s'=CHANGELANE_0_%i) +
                  %f : (p'=1) & (s'=CHANGELANE_1_%i);\n" % (k, from_s, k, accident_prob[(hazz,'changelane')], k,
                 1.0-accident_prob[(hazz,'changelane')], k))
        else: # standard actions
            smg.write("\t[%s_%i] p=1 & s=%i & car_position=POS_%i -> %f : (p'=2) & (s'=-1) + %f : (p'=1) & (s
                 '=-2);\n" % (reactions[hazz][i], k, from_s, k, accident_prob[(hazz,reactions[hazz][i])], 1.0-
                 accident_prob[(hazz,reactions[hazz][i])]))

# the base cases - one for each number of successors (taking no uturn at intersection into account)
for i, succs in successors.iteritems():
    # k is the first edge with i successors
    k = edgeid[(succs[0][0],succs[0][1])]
    # the corresponding distribution
    smg.write("\n\nmodule field_%i\n\n" % (k))
    smg.write("\t[hazard_%i] p=2 & s=0 & car_position=POS_%i -> DISTR_0_%i+DISTR_1_%i+DISTR_2_%i+DISTR_5_%i : (s
         '=-6) + DISTR_3_%i+DISTR_4_%i+DISTR_6_%i : (s'=-7);\n" % (k, k, k, k, k, k, k, k, k))

    # jam or pedestrian
    smg.write("\t[hazard_%i] p=2 & s=-6 & car_position=POS_%i -> (DISTR_2_%i+DISTR_5_%i)/(DISTR_0_%i+DISTR_1_%i+
         DISTR_2_%i+DISTR_5_%i) : (s'=-8) + (DISTR_0_%i+DISTR_1_%i)/(DISTR_0_%i+DISTR_1_%i+DISTR_2_%i+DISTR_5_%i)
          : (s'=-9);\n" % (k, k, k, k, k, k, k, k, k, k, k, k, k, k))
    # obstacle or nothing
    smg.write("\t[hazard_%i] p=2 & s=-7 & car_position=POS_%i -> (DISTR_3_%i+DISTR_4_%i)/(DISTR_3_%i+DISTR_4_%i+
         DISTR_6_%i) : (s'=-10) + (DISTR_6_%i)/(DISTR_3_%i+DISTR_4_%i+DISTR_6_%i) : (s'=-2) & (p'=1);\n" % (k, k,
          k, k, k, k, k, k, k, k))

    # jam&pedestrian or jam only
    pj = 0
    if ['pedestrian','jam'] in haz:
        pj = haz.index(['pedestrian','jam'])
    else:
        pj = haz.index(['jam','pedestrian'])
    j = haz.index(['jam'])
    smg.write("\t[hazard_%i] p=2 & s=-8 & car_position=POS_%i -> DISTR_2_%i/(DISTR_2_%i+DISTR_5_%i) : (s'=%i) +
         DISTR_5_%i/(DISTR_2_%i+DISTR_5_%i) : (s'=%i) & (p'=1);\n" % (k, k, k, k, k, pj+1, k, k, k, j+1))

    # pedestrian&obstacle or pedestrian only
    po = 0
    if ['pedestrian','obstacle'] in haz:
        po = haz.index(['pedestrian','obstacle'])
    else:
        po = haz.index(['obstacle','pedestrian'])
    p = haz.index(['pedestrian'])
```

```
    smg.write("\t[hazard_%i] p=2 & s=-9 & car_position=POS_%i -> DISTR_1_%i/(DISTR_0_%i+DISTR_1_%i) : (s'=%i) +
         DISTR_0_%i/(DISTR_0_%i+DISTR_1_%i) : (s'=%i) & (p'=1);\n" % (k, k, k, k, k, po+1, k, k, k, p+1))

    # obstacle&jam or obstacle only
    oj = 0
    if ['jam','obstacle'] in haz:
        oj = haz.index(['jam','obstacle'])
    else:
        oj = haz.index(['obstacle','jam'])
    o = haz.index(['obstacle'])
    smg.write("\t[hazard_%i] p=2 & s=-10 & car_position=POS_%i -> DISTR_4_%i/(DISTR_3_%i+DISTR_4_%i) : (s'=%i) +
         DISTR_3_%i/(DISTR_3_%i+DISTR_4_%i) : (s'=%i) & (p'=1);\n" % (k, k, k, k, k, oj+1, k, k, k, o+1))

    # instantiate hazard and pick reaction - bad guy followed by good guy
    for j in xrange(0, len(haz)-1):
        if len(haz[j]) == 1:
            subhazard(j+1, [], haz[j], k)
        if len(haz[j]) == 2:
            subhazard(j+1, [haz.index([haz[j][0]])+1, haz.index([haz[j][1]])+1], haz[j], k)

    # pick destination - good guy
    for j in xrange(0,i):
        smg.write("\t[move_%i_%i] p=1 & s=%i & car_position=POS_%i-> (car_position'=DEST_%i_%i) & (s'=0) & (p'=2)
             ;\n" % (j, k, -2, k, j, k))

    if i == 0:
        smg.write("\t[term_%i] p=1 & s=%i & car_position=POS_%i -> (car_position'=DEST_%i) & (s'=0) & (p'=2);\n"
             % (k, -2, k, k))

    smg.write("\nendmodule\n\n\n")

# iterate through topology
for i, succs in successors.iteritems():
    # if just one edge with i successors, then have it covered already
    if len(succs) > 1:
        l = edgeid[(succs[0][0],succs[0][1])]
        for e in succs[1:]: # start iterating at seccond edge
            k = edgeid[(e[0],e[1])]
            # module and position
            smg.write("module field_%i = field_%i [POS_%i=POS_%i" % (k, l, l, k))
            # hazard
            for h in hazards:
                smg.write(", %s_%i=%s_%i" % (h, l, h, k))
            smg.write(", hazard_%i=hazard_%i" % (l, k))
            # reactions
            for r in reacts:
                smg.write(", %s_%i=%s_%i" % (r, l, r, k))
            # steering
            for j in xrange(0,i):
                smg.write(", move_%i_%i=move_%i_%i, DEST_%i_%i = DEST_%i_%i" % (j, l, j, k, j, l, j, k))
            # hazard-set distribution
            for j in xrange(0,len(haz)):
                smg.write(", DISTR_%i_%i=DISTR_%i_%i" % (j, l, j, k))
            # uturn reactions
            smg.write(", UTURN_0_%i=UTURN_0_%i, UTURN_1_%i=UTURN_1_%i, REVEDGE_%i=REVEDGE_%i, UTURNPLAYER_%i=
                 UTURNPLAYER_%i" % (l, k, l, k, l, k, l, k))
            # changelane reactions
            smg.write(", CHANGELANE_0_%i=CHANGELANE_0_%i, CHANGELANE_1_%i=CHANGELANE_1_%i" % (l, k, l, k))
            # alternatively, termination
            if i == 0:
                smg.write(", term_%i=term_%i, DEST_%i=DEST_%i" % (l, k, l, k))
            smg.write("] endmodule\n")

# terminal states
smg.write("\nmodule terminals\n")
smg.write("\t[term] s=-1 -> (car_position'=POS_term) & (p'=2) & (s'=-11);\n") # accident
smg.write("\t[term] s=-3 -> (car_position'=POS_term) & (p'=2) & (s'=-11);\n") # illegal action
smg.write("\t[term] car_position=POS_goal -> (car_position'=POS_term) & (p'=2) & (s'=-11);\n")
smg.write("\t[term] car_position=POS_term -> (car_position'=POS_term) & (p'=2) & (s'=-11);\n")
smg.write("endmodule\n\n")

# REWARDS
# reaching goal
smg.write("\nrewards \"reach_goal\"\n")
smg.write("\tcar_position=POS_goal : 1;\n")
smg.write("endrewards\n\n")
# avoiding accidents
smg.write("\nrewards \"accidents\"\n")
smg.write("\ts=-1 : 1;\n")
smg.write("endrewards\n\n")
# road value
smg.write("\nrewards \"road_quality\"\n")
for e in G.edges(data=True):
    ie = edgeid[(e[0],e[1])]
    smg.write("\tcar_position=%i & s=-2: %f;\n" % (ie, e[2]["data"]["value"]*e[2]["data"]["dist"]/1000))
smg.write("endrewards\n\n")

##########################################################################
# Properties File                                                        #
##########################################################################

# write the property - a three-objective conjunctive goal
if(len(sys.argv)>=2):
    if(int(sys.argv[1])==1): # islip
        prop.write("<<1>> (R{\"reach_goal\"}>=0.7 [ C ] &  R{\"accidents\"}<=0.3 [ C ] &  R{\"road_quality
             \"}>=6.0 [ C ])");
    elif(int(sys.argv[1])==2): # islip
        prop.write("<<1>> (R{\"reach_goal\"}>=0.7 [ C ] & R{\"accidents\"}<=0.3 [ C ] & R{\"road_quality\"}>=6.0
             [ C ])");
```

```
else: # islip is default
    prop.write("<<1>> (R{\"reach_goal\"}>=0.7 [ C ] & R{\"accidents\"}<=0.3 [ C ] & R{\"road_quality\"}>=6.0 [ C
        ])");

########################################################################
# Bash File                                                            #
########################################################################

# write the command to execute
bash.write("#!/bin/bash\n\n")
bash.write("../../bin/prism %s{.prism,.props} -prop 1 -multirounding -multimaxciter 500 -baselineaccuracy 200 -
     increasefactor 1.01 -paretoepsilon 0.001 -logcpareto -gs -exportstrat %s.strat 2>&1 1> %s.log &\n" % (
     filename, filename, filename))

########################################################################
# Wrap up                                                              #
########################################################################

# close down files - flush
smg.close()
prop.close()
bash.close()
```

B.2 UAV Path Planning

uav.prism

```
smg

// operator parameters
const double p=0.5; // probability of increasing workload due to other uncertain tasks
const double accu_load1; // accuracy at workload level 1 (low)
const double accu_load2; // accuracy at workload level 2 (high)
const double fd; // accuracy discount due to fatigue
const int COUNTER; // fatigue threshold
const double del; // probability of operator to delegate

// uav variables
global stop : bool init false; // done visiting all waypoints
global c : [0..3] init 0; // choices at the checkpoint
formula roz = (r=8) | (w=3&a=1) | (w=3&a=2) | (w=5&a=2); // restricted operating zones

//players
global p1 : [1..2] init 1; // 1 ... UAV, 2 ... operator
player p1
    UAV, [camera], [fly], [delegated], [uav_stop]
endplayer

player p2
    operator, [image], [process], [wait], [delegate], [prescribe], [operator_stop]
endplayer

// OPERATOR MODEL
module operator
    k : [0..100] init 0; // fatigue level measured by completed tasks
    t : [0..2] init 0; // workload level
    s : [0..2] init 0; // status of image processing, 0: init, 1: good, 2: bad
    q : [0..2] init 2; // delegation status: don't care if 2, delegation if 0, no delegation if 1

    // image processing, the workload may increase due to other unknown tasks
    [image] !stop & t=0 & s=0 -> (1 - p) : (t'=1) & (s'=0) + p : (t'=2) & (s'=0);
    // not fatigue, workload level 1
    [process] !stop & p1=2 & t=1 & s=0 & k<=COUNTER -> accu_load1 : (s'=1)&(k'=k + 1) + (1 - accu_load1) : (s'=2)&(k'=k + 1);
    // fatigue, workload level 1
    [process] !stop & p1=2 & t=1 & s=0 & k>COUNTER -> accu_load1 * fd : (s'=1) + (1 - accu_load1 * fd) : (s'=2);
    // not fatigue, workload level 2
    [process] !stop & p1=2 & t=2 & s=0 & k=>COUNTER -> accu_load2 : (s'=1)&(k'=k + 1) + (1 - accu_load2) : (s'=2)&(k'=k + 1);
    // fatigue, workload level 2
    [process] !stop & p1=2 & t=2 & s=0 & k>COUNTER -> accu_load2 * fd : (s'=1) + (1 - accu_load2 * fd) : (s'=1);

    // image analysis is bad, UAV needs to wait at the waypoint and take another image
    [wait] !stop & p1=2 & s=2 -> (p1'=1) & (t'=0) & (s'=0);

    // if image analysis is good, UAV can continue flying
    // at checkpoints, operator may suggest route for the UAV

    // with probability del, the operator delegates the decision of the next cornerpoint to the UAV,
    [] !stop & p1=2 & s=1 & (w=2 | w=5 | w=6) & (q=2) -> del : (q'=0) + (1 - del) : (q'=1);

    [delegate] !stop & p1=2 & s=1 & (w=2 | w=5 | w=6) & (q=0) -> (p1'=1) & (t'=0) & (s'=0) & (q'=2); // allow UAV to choose

    // w2 -> r5 (c=0) |r6 (c=1) |r7 (c=2)|r9 (c=3)
    [prescribe] !stop & p1=2 & s=1 & w=2 & (q=1) -> (p1'=1) & (c'=3) & (t'=0) & (s'=0)& (q'=2);
    [prescribe] !stop & p1=2 & s=1 & w=2 & (q=1) -> (p1'=1) & (c'=2) & (t'=0) & (s'=0)& (q'=2);
    [prescribe] !stop & p1=2 & s=1 & w=2 & (q=1) -> (p1'=1) & (c'=1) & (t'=0) & (s'=0)& (q'=2);
    [prescribe] !stop & p1=2 & s=1 & w=2 & (q=1) -> (p1'=1) & (c'=0) & (t'=0) & (s'=0)& (q'=2);
```

```
    // w5 → r3 (c=0)| r4 (c=1)| w4 (c=2)
    [prescribe] !stop & pl=2 & s=1 & w=5 & (q=1) → (pl'=1) & (c'=2) & (t'=0) & (s'=0)& (q'=2);
    [prescribe] !stop & pl=2 & s=1 & w=5 & (q=1) → (pl'=1) & (c'=1) & (t'=0) & (s'=0)& (q'=2);
    [prescribe] !stop & pl=2 & s=1 & w=5 & (q=1) → (pl'=1) & (c'=0) & (t'=0) & (s'=0)& (q'=2);

    // w6 → r2 (c=0)| r3 (c=1) |r8 (c=2)
    [prescribe] !stop & pl=2 & s=1 & w=6 & (q=1) → (pl'=1) & (c'=2) & (t'=0) & (s'=0)& (q'=2);
    [prescribe] !stop & pl=2 & s=1 & w=6 & (q=1) → (pl'=1) & (c'=1) & (t'=0) & (s'=0)& (q'=2);
    [prescribe] !stop & pl=2 & s=1 & w=6 & (q=1) → (pl'=1) & (c'=0) & (t'=0) & (s'=0)& (q'=2);

    // at non-check-points, UAV has full autonomy to choose flying route
    [delegate] !stop & pl=2 & s=1 & (w!=2 & w!=5 & w!=6) → (pl'=1) & (t'=0) & (s'=0);

    // operator stops
    [operator_stop] stop & pl=2 → true;

endmodule

// UAV MODEL
module UAV
    // UAV positions:
    // inside a waypoint: w!=0, a=0, r=0
    // fly through certain angle of a waypoint: w!=0, a!=0, r=0
    // fly through a road point: w=0, a=0, r!=0
    w : [0..6] init 1; // waypoint
    r : [0..9] init 0; // road points
    send : bool init true;
    in : bool init true;
    // flag which waypoints were visited
    w1 : bool init true;
    w2 : bool init false;
    w3 : bool init false;
    w4 : bool init false;
    w5 : bool init false;
    w6 : bool init false;
    // if operator delegated decision at checkpoint
    delegated : bool init false;

    // at any waypoint:
    // send image to operator for analysis
    [image] pl=1 & w!=0 & a=0 & r=0 & send → (pl'=2) & (send'=false);
    // wait at the waypoint and send another image
    [wait] !send → (send'=true);
    // fly into a waypoint and take an image
    [camera] pl=1 & w=1 & a!=0 & r=0 & in & !delegated → (send'=true) & (w1'=true);
    [camera] pl=1 & w=2 & a!=0 & r=0 & in & !delegated → (send'=true) & (w2'=true);
    [camera] pl=1 & w=3 & a!=0 & r=0 & in & !delegated → (send'=true) & (w3'=true);
    [camera] pl=1 & w=4 & a!=0 & r=0 & in & !delegated → (send'=true) & (w4'=true);
    [camera] pl=1 & w=5 & a!=0 & r=0 & in & !delegated → (send'=true) & (w5'=true);
    [camera] pl=1 & w=6 & a!=0 & r=0 & in & !delegated → (send'=true) & (w6'=true);
    // fly out of the waypoint via any anglepoint
    [prescribe] w!=0 & a=0 & r=0 → (in'=false);
    [delegate] w!=0 & a=0 & r=0 & (w=2 | w=5 | w=6) → (delegated'=true);
    [delegate] w!=0 & a=0 & r=0 & !(w=2 | w=5 | w=6) → (in'=false);
    // UAV flying plans (based on the road map)
    // checkpoints: receiving commands from the operator
    // w2 → r5 |r6 |r7 |r9
    [delegated] pl=1 & w=2 & a!=0 & r=0 & delegated=true → (in'=false) & (c'=0) & (delegated'=false);
    [delegated] pl=1 & w=2 & a!=0 & r=0 & delegated=true → (in'=false) & (c'=1) & (delegated'=false);
    [delegated] pl=1 & w=2 & a!=0 & r=0 & delegated=true → (in'=false) & (c'=2) & (delegated'=false);
    [delegated] pl=1 & w=2 & a!=0 & r=0 & delegated=true → (in'=false) & (c'=3) & (delegated'=false);
    [fly] pl=1 & c=0 & w=2 & (a!=0) & r=0 & !in → (r'=5);
    [fly] pl=1 & c=1 & w=2 & (a!=0) & r=0 & !in → (r'=6);
    [fly] pl=1 & c=2 & w=2 & (a!=0) & r=0 & !in → (r'=7);
    [fly] pl=1 & c=3 & w=2 & (a!=0) & r=0 & !in → (r'=9);

    // w5 → r3 | r4 | w4 (at any anglepoint)
    [delegated] pl=1 & w=5 & a!=0 & r=0 & delegated=true → (in'=false) & (c'=0) & (delegated'=false);
    [delegated] pl=1 & w=5 & a!=0 & r=0 & delegated=true → (in'=false) & (c'=1) & (delegated'=false);
    [delegated] pl=1 & w=5 & a!=0 & r=0 & delegated=true → (in'=false) & (c'=2) & (delegated'=false);
    [fly] pl=1 & c=0 & w=5 & (a!=0) & r=0 & !in → (r'=3);
    [fly] pl=1 & c=1 & w=5 & (a!=0) & r=0 & !in → (r'=4);
    [fly] pl=1 & c=2 & w=5 & (a!=0) & r=0 & !in → (w'=4) & (r'=0) & (in'=true);

    // w6 → r2 | r3 |r8
    [delegated] pl=1 & w=6 & a!=0 & r=0 & delegated=true → (in'=false) & (c'=0) & (delegated'=false);
    [delegated] pl=1 & w=6 & a!=0 & r=0 & delegated=true → (in'=false) & (c'=1) & (delegated'=false);
    [delegated] pl=1 & w=6 & a!=0 & r=0 & delegated=true → (in'=false) & (c'=2) & (delegated'=false);
    [fly] pl=1 & c=0 & w=6 & (a!=0) & r=0 & !in → (r'=2);
    [fly] pl=1 & c=1 & w=6 & (a!=0) & r=0 & !in → (r'=3);
    [fly] pl=1 & c=2 & w=6 & (a!=0) & r=0 & !in → (r'=8);

    // non-checkpoints: fly autonomously
    // w1 → r1 | r9
    [fly] pl=1 & w=1 & (a!=0) & r=0 & !in → (r'=1);
    [fly] pl=1 & w=1 & (a!=0) & r=0 & !in → (r'=9);

    // w3 → r6 | w4 (any anglepoint)
    [fly] pl=1 & w=3 & (a!=0) & r=0 & !in → (r'=6);
```

```
    [fly] pl=1 & w=3 & (a!=0) & r=0 & !in → (w'=4) & (r'=0) & (in'=true);

    // w4 → w3 | w5
    [fly] pl=1 & w=4 & (a!=0) & r=0 & !in → (w'=3) & (r'=0) & (in'=true);
    [fly] pl=1 & w=4 & (a!=0) & r=0 & !in → (w'=5) & (r'=0) & (in'=true);

    // r1 → r2 | w1
    [fly] pl=1 & r=1 → (r'=2);
    [fly] pl=1 & r=1 → (w'=1) & (r'=0) & (in'=true);

    // r2 → r1 | w6
    [fly] pl=1 & r=2 → (r'=1);
    [fly] pl=1 & r=2 → (w'=6) & (r'=0) & (in'=true);

    // r3 → w5 | w6
    [fly] pl=1 & r=3 → (w'=5) & (r'=0) & (in'=true);

    [fly] pl=1 & r=3 → (w'=6) & (r'=0) & (in'=true);

    // r4 → r5 | w5
    [fly] pl=1 & r=4 → (r'=5);
    [fly] pl=1 & r=4 → (w'=5) & (r'=0) & (in'=true);

    // r5 → r4 | w2
    [fly] pl=1 & r=5 → (r'=4);
    [fly] pl=1 & r=5 → (w'=2) & (r'=0) & (in'=true);

    // r6 → w2 | w3
    [fly] pl=1 & r=6 → (w'=2) & (r'=0) & (in'=true);
    [fly] pl=1 & r=6 → (w'=3) & (r'=0) & (in'=true);

    // r7 → w2 | r8
    [fly] pl=1 & r=7 → (w'=2) & (r'=0) & (in'=true);
    [fly] pl=1 & r=7 → (r'=8);

    // r8 → w6 | r7
    [fly] pl=1 & r=8 → (w'=6) & (r'=0) & (in'=true);
    [fly] pl=1 & r=8 → (r'=7);

    // r9 → w1 | w2
    [fly] pl=1 & r=9 → (w'=1) & (r'=0) & (in'=true);
    [fly] pl=1 & r=9 → (w'=2) & (r'=0) & (in'=true);

    // stop if target reached
    [uav_stop] stop & pl=1 → true;

endmodule

// CHOICE OF ANGLEPOINTS
module angle_choice
    a : [0..8] init 0; // anglepoints

    // 3 - 4 - 5
    // |   |   |
    // 2   o   6
    // |   |   |
    // 1 - 8 - 7

    [prescribe] true → (a'=1);
    [prescribe] true → (a'=2);
    [prescribe] true → (a'=3);
    [prescribe] true → (a'=4);
    [prescribe] true → (a'=5);
    [prescribe] true → (a'=6);
    [prescribe] true → (a'=7);
    [prescribe] true → (a'=8);
    [delegate] true → (a'=1);
    [delegate] true → (a'=2);
    [delegate] true → (a'=3);
    [delegate] true → (a'=4);
    [delegate] true → (a'=5);
    [delegate] true → (a'=6);
    [delegate] true → (a'=7);
    [delegate] true → (a'=8);

    [camera] true → (a'=0);
endmodule

// MISSION OBJECTIVE
module waypoint_check
    // check if waypoints w1, w2 and w6 have been visited
    [fly] w1 & w2 & w6 → (stop'=true);
    [camera] w1 & w2 & w6 → (stop'=true);
    [camera] !(w1 & w2 & w6) → true;
    [fly] !(w1 & w2 & w6) → true;
endmodule

rewards "time" // flight time
     [wait] true : 10;
```

```
    [fly] true : 60;
endrewards

rewards "ROZ" // ROZ occupancy
    [fly] roz : 1;
endrewards
```

uav.props

```
const double v_ROZ;
const double v_time;

<<p1>> R{"time"}min=? [ F w1&w2&w6 ]

<<p1>> and(R{"ROZ"}<=v_ROZ [ C ], R{"time"}<=v_time [ C ]);
```

B.3 Aircraft Power Control

power.prism

```
smg

//////////////////////////////// TOP-LEVEL SYTEM (200) ////////////////////////////////////

system "MEV"
    "HVAC_LEFT" || "HVAC_RIGHT"
endsystem

// MODEL PARAMTERS
const double sample_time = 1;   // [s] every sample_time seconds, the reactive loop is performed

// generator dynamics
const N;              // [#] number of timesteps generator configuration stays the same
const double pg = 0.8;   // [prob] probability of generator being healthy per max_gc time steps

// interface dynamics
const double I1_health;  // [prob] probability of interface being powered

// connector dynamics
const del_max;        // [#] maximum turn-off delay in loop itarations of connector, turn-on delay is (floor of) half this

///////////////////////////// HIGH VOLTAGE AC, LEFT (210a) ////////////////////////////////

system "HVAC_LEFT"
    C1_switch || C2_switch || C3_switch || C4_switch || GEN_LEFT || HVAC_LEFT || GEN_COUNTER_LEFT
endsystem

const C1_init = 0;
const C2_init = 0;
const C3_init = 0;
const C4_init = 0;

const C1_turn_off_int = 1; // need to turn off when interface opens
const C2_turn_off_int = 1; // need to turn off when interface opens
const C3_turn_off_int = 0; // no need to turn off when interface opens

module C1_switch
     C1 : [0..1] init C1_init; // connector status for C1
    C1_int : [0..1] init C1_init; // intention for C1
    C1_del : [0..max(del_max, 1)] init del_max; // delay for switching

    // change intention to switch off
    [switch_left]      C1_int!=0 → (C1_int'=0) & (C1_del'=0);
    [switch_left_int_on] C1_int!=0 → (C1_int'=0) & (C1_del'=0);
    // change intention to switch on
    [switch_left]      C1_int!=1              → (C1_int'=1) & (C1_del'=ceil(del_max/2));
    [switch_left_int_on] C1_int!=1 & C1_turn_off_int=0 → (C1_int'=1) & (C1_del'=ceil(del_max/2));
    // do nothing
    [switch_left]      true                 → true;
    [switch_left_int_on] C1_int=0 | C1_turn_off_int=0 → true;

    // resolve connector status
    [conn_res_left] true                  → (C1'=C1_int) & (C1_del'=del_max);
    // not resolved - increase counter
    [conn_res_left] C1_del<del_max & (C1!=C1_int) → (C1_del'=C1_del + 1);
endmodule

module C2_switch = C1_switch [C1=C2, C1_int=C2_int, C1_init=C2_init, C1_del=C2_del, C1_turn_off_int=C2_turn_off_int] endmodule
module C3_switch = C1_switch [C1=C3, C1_int=C3_int, C1_init=C3_init, C1_del=C3_del, C1_turn_off_int=C3_turn_off_int] endmodule
```

```
// switch over interface is special
// C4 should only be switched on if no power could flow out of HVAC_ left,
// so due to delays we need to make sure that C1 and C2 are off,
// and their intention is off as well
module C4_switch
     C4 : [0..1] init C4_init; // connector status for C4
    C4_int : [0..1] init C4_init; // intention for C4
    C4_del : [0..max(del_max, 1)] init del_max; // delay for switching

    // change intention to switch off
    [switch_left]      C4_int!=0 → (C4_int'=0) & (C4_del'=0);
    // change intention to switch on
    [switch_left_int_on] C4_int!=1 → (C4_int'=1) & (C4_del'=ceil(del_max/2));
    // no change if save to leave on
    [switch_left] true → true;

    // resolve connector status
    [conn_res_left] true                    → (C4'=C4_int) & (C4_del'=del_max);
    // not resolved - increase counter
    [conn_res_left] C4_del<del_max & (C4!=C4_int) → (C4_del'=C4_del + 1);
endmodule

module GEN_COUNTER_LEFT
    GC_left : [0..max(N, 1)] init 0;

    [gen_left] GC_left<N → (GC_left'=GC_left + 1);
    [gen_left] GC_left=N → (GC_left'=0);
endmodule

module GEN_LEFT
    // generators
     G224a : [0..1] init (pg=0.0 ? 0 : 1); // G1
     G252a : [0..1] init (pg=0.0 ? 0 : 1); // G2

    [gen_left] GC_left=0 → pg * pg       : (G224a'=1) & (G252a'=1) // both generators work
                         + pg * (1 − pg)     : (G224a'=1) & (G252a'=0) // G2 fails
                         + (1 − pg) * pg     : (G224a'=0) & (G252a'=1) // G1 fails
                         + (1 − pg) * (1 − pg) : (G224a'=0) & (G252a'=0); // both generators fail
    [gen_left] GC_left>0 → true; // no change in generator status
endmodule

// interface I1 (left to right direction)
formula i1l_pow = C4=0 & bus_214a2=1; // need C4=0, because want to avoid circularity
formula i1l_off = !i1l_pow;

module HVAC_LEFT
    // status of power delivered over interface I1 to left side
    I1_left : [0..1] init (I1_health=1.0 ? 1 : 0);
    // players / stages:
    pL : [1..7] init 1;

    // 1: generators
    [gen_left?] pL=1 → (pL'=2);

    // 2: tau for normal form
    [] pL=2 → (pL'=3);

    // 3: contactors
    [switch_left!]      pL=3 → (pL'=4);
    [switch_left_int_on!] pL=3 → (pL'=4);

    // 4: resolve connector status nondeterministically
    [conn_res_left?] pL=4 → (pL'=5);

    // 5: status - also sample interface distribution
    [stat_buses_safe_left?] pL=5 & buses_left & safe_left → I1_health : (pL'=6) + (1 − I1_health) : (pL'=7);
    [stat_buses_left?]     pL=5 & buses_left & !safe_left → I1_health : (pL'=6) + (1 − I1_health) : (pL'=7);
    [stat_safe_left?]      pL=5 & !buses_left & safe_left → I1_health : (pL'=6) + (1 − I1_health) : (pL'=7);
    [stat_left?]         pL=5 & !buses_left & !safe_left → I1_health : (pL'=6) + (1 − I1_health) : (pL'=7);

    // 6&7: interfaces - set incoming power according to previously drawn sample
    [i1l_pow__i1r_pow?] pL=6 & i1l_pow → (I1_left'=1) & (pL'=1);
    [i1l_pow__i1r_off?] pL=7 & i1l_pow → (I1_left'=0) & (pL'=1);
    [i1l_off__i1r_pow?] pL=6 & i1l_off → (I1_left'=1) & (pL'=1);
    [i1l_off__i1r_off?] pL=7 & i1l_off → (I1_left'=0) & (pL'=1);

endmodule

// SAFETY SPECIFICATIONS - HVAC LEFT

// no paralleling of sources
formula safe_left = (C1=0 | C3=0 | C2=0) &
            (C1=0 | C3=0 | C4=0 | I1_left=0) &
            (C2=0 | C4=0 | I1_left=0);

// buses
formula bus_214a1 = (G224a=1 & C1=1)
            | (G252a=1 & C3=1 & C2=1)
            | (C3=1 & C4=1 & I1_left=1)
```

```
          ? 1 : 0;
formula bus_214a2 = (G224a=1 & C1=1 & C3=1)
          | (G252a=1 & C2=1)
          | (C4=1 & I1_left=1)
           ? 1 : 0;

// both buses on left side powered
formula buses_left = bus_214a1=1 & bus_214a2=1;


// REWARDS - HVAC LEFT
// defined over actions

rewards "fail_l"
    // left side fails
    [stat_buses_left] true : sample_time;
    [stat_left]     true : sample_time;
endrewards

rewards "i1_l"
    // both buses on left side are powered
    [i1l_pow__i1r_pow] true : sample_time;
    [i1l_pow__i1r_off] true : sample_time;
endrewards

rewards "buses_l"
    // left side powers interface
    [stat_buses_safe_left] true : sample_time;
    [stat_buses_left]    true : sample_time;
endrewards

rewards "time"
    [i1l_pow__i1r_pow] true : sample_time;
    [i1l_pow__i1r_off] true : sample_time;
    [i1l_off__i1r_pow] true : sample_time;
    [i1l_off__i1r_off] true : sample_time;
endrewards


////////////////////////// HIGH VOLTAGE AC, RIGHT (210b) ////////////////////////////


system "HVAC_RIGHT"
    C5_switch || C6_switch || C7_switch || C8_switch || GEN_RIGHT || HVAC_RIGHT || GEN_COUNTER_RIGHT
endsystem

const C5_init = 0;
const C6_init = 0;
const C7_init = 0;
const C8_init = 0;

const C5_turn_off_int = 1; // need to turn off when interface opens
const C6_turn_off_int = 1; // need to turn off when interface opens
const C7_turn_off_int = 0; // no need to turn off when interface opens

module C5_switch = C1_switch [C1=C5, C1_int=C5_int, C1_init=C5_init, C1_del=C5_del, conn_res_left=conn_res_right,
   switch_left=switch_right, switch_left_int_on=switch_right_int_on, C1_turn_off_int=C5_turn_off_int] endmodule
module C6_switch = C1_switch [C1=C6, C1_int=C6_int, C1_init=C6_init, C1_del=C6_del, conn_res_left=conn_res_right,
   switch_left=switch_right, switch_left_int_on=switch_right_int_on, C1_turn_off_int=C6_turn_off_int] endmodule
module C7_switch = C1_switch [C1=C7, C1_int=C7_int, C1_init=C7_init, C1_del=C7_del, conn_res_left=conn_res_right,
   switch_left=switch_right, switch_left_int_on=switch_right_int_on, C1_turn_off_int=C7_turn_off_int] endmodule
module C8_switch = C4_switch [C4=C8, C4_int=C8_int, C4_init=C8_init, C4_del=C8_del, conn_res_left=conn_res_right,
   switch_left=switch_right, switch_left_int_on=switch_right_int_on] endmodule

module GEN_COUNTER_RIGHT = GEN_COUNTER_LEFT[gen_left=gen_right, GC_left=GC_right] endmodule

module GEN_RIGHT = GEN_LEFT[G224a=G224b, G252a=G252b, gen_left=gen_right, GC_left=GC_right] endmodule

// interface I1 (right to left direction)
formula i1r_pow = C8=0 & bus_214b2=1; // need C8=0, because want to avoid circularity
formula i1r_off = !i1r_pow;

module HVAC_RIGHT
    // status of power delivered over interface I1 to right side
    I1_right : [0..1] init (I1_health=1.0 ? 1 : 0);
    // players / stages:
    pR : [1..7] init 1;

    // 1: generators
    [gen_right?] pR=1 -> (pR'=2);

    // 2: tau for normal form
    [] pR=2 -> (pR'=3);

    // 3: contactors
    [switch_right!]      pR=3 -> (pR'=4);
    [switch_right_int_on!] pR=3 -> (pR'=4);

    // 4: resolve connector status nondeterministically
```

```
    [conn_res_right?] pR=4 → (pR'=5);

    // 5: status - also sample interface distribution
    [stat_buses_safe_right?] pR=5 & buses_right & safe_right → I1_health : (pR'=6) + (1 − I1_health) : (pR'=7);
    [stat_buses_right?]    pR=5 & buses_right & !safe_right → I1_health : (pR'=6) + (1 − I1_health) : (pR'=7);
    [stat_safe_right?]     pR=5 & !buses_right & safe_right → I1_health : (pR'=6) + (1 − I1_health) : (pR'=7);
    [stat_right?]        pR=5 & !buses_right & !safe_right → I1_health : (pR'=6) + (1 − I1_health) : (pR'=7);

    // 6&7: interfaces - set incoming power according to previously drawn sample
    [i1l_pow__i1r_pow?] pR=6 & i1r_pow → (I1_right'=1) & (pR'=1);
    [i1l_pow__i1r_off?] pR=7 & i1r_pow → (I1_right'=0) & (pR'=1);
    [i1l_off__i1r_pow?] pR=6 & i1r_off → (I1_right'=1) & (pR'=1);
    [i1l_off__i1r_off?] pR=7 & i1r_off → (I1_right'=0) & (pR'=1);

endmodule

// both buses on right side powered
formula buses_right = bus_214b1=1 & bus_214b2=1;

// SAFETY SPECIFICATIONS - HVAC RIGHT

// no paralleling of sources
formula safe_right = (C5=0 | C8=0 | I1_right=0) &
            (C5=0 | C7=0 | C6=0) &
           (C6=0 | C7=0 | C8=0 | I1_right=0);

// buses
formula bus_214b1 = (C8=1 & I1_right=1) |
            (G252b=1 & C5=1) |
            (G224b=1 & C7=1 & C6=1)
            ? 1 : 0;
formula bus_214b2 = (G224b=1 & C6=1) |
            (I1_right=1 & C7=1 & C8=1) |
            (G252b=1 & C5=1 & C7=1)
             ? 1 : 0;

// safe and buses powered
formula pow_safe_right = safe_right & bus_214b1=1 & bus_214b2=1;


// REWARDS - HVAC RIGHT
// defined over actions

rewards "fail_r"
    // right side fails
    [stat_buses_right] true : sample_time;
    [stat_right]     true : sample_time;
endrewards

rewards "i1_r"
    // right side powers interface
    [i1l_pow__i1r_pow] true : sample_time;
    [i1l_off__i1r_pow] true : sample_time;
endrewards

rewards "buses_r"
    // both buses on right side are powered
    [stat_buses_safe_right] true : sample_time;
    [stat_buses_right]    true : sample_time;
endrewards
```

power.props

```
// Compositional top-level property of full system, without interface
comp("P210a", "P210b");

// Compositional top-level property of full system, using interface
comp("P210a_int", "P210b_int");

// Global top-level property of full system

////////////////////////////////////////////////////////////////////////////////////////////
// Properties of HVAC left (210a)

const double u_fail_l; // guar: upper bound on failures
const double l_buses_l; // guar: lower bound on powering the buses
const double l_i1_l; // guar: lower bound on the left side delivering power over I1

// minimise failures
"safe_210a" : P>=1 [ R{path}{"fail_l"}/{"time"}<=u_fail_l [ S ] ]

// maximise uptime for buses
"buses_210a" : P>=1 [ R{path}{"buses_l"}/{"time"}>=l_buses_l [ S ] ]

// maximise power over interface
"I1_210a" : P>=1 [ R{path}{"i1_l"}/{"time"}>=l_i1_l [ S ] ]
```

```
// full property of HVAC left (210a) without interface
"P210a" : <<1>> ("safe_210a" & "buses_210a")

// full property of HVAC left (210a) with interface
"P210a_int" : <<1>> ("safe_210a" & "buses_210a" & "I1_210a")


////////////////////////////////////////////////////////////////////////////////////////
// Properties of HVAC right (210b)

const double u_fail_r; // guar: upper bound on failures
const double l_buses_r; // guar: lower bound on powering the buses
const double l_i1_r; // guar: lower bound on the right side delivering power over I1

// minimise failures
"safe_210b" : P>=1 [ R(path){"fail_r"}/{"time"}<=u_fail_r [ S ] ]

// maximise uptime for buses
"buses_210b" : P>=1 [ R(path){"buses_r"}/{"time"}>=l_buses_r [ S ] ]

// maximise power over interface
"I1_210b" : P>=1 [ R{"i1_r"}/{"time"}>=l_i1_r [ S ] ]

// full property of HVAC right (210b) without interface
"P210b" : <<1>> ("safe_210b" & "buses_210b")

// full property of HVAC right (210b) with interface
"P210b_int" : <<1>> ("safe_210b" & "buses_210b" & "I1_210b")
```

B.4 Room Temperature Control

temperature.m

```
% Temperature Control Case Study

clc; close all;

%================= Discretisation ============================
rooms = 3;
w = [3, 3, 3];      % room width (m)
l = [3, 3, 3];      % room length (m)
h = [2, 2, 2];      % room height (m)
Aw = 1.4 * 1.4;     % window size (m^2)
%    w       w      w
% |------|-------|------|
% |  1   |   2   |  3   | l
% |------|-------|------|
%

V_el = 10*10^-3;   % chilled water volume (10 liters)
V = V_el;

r = 0.01;               % pipeline radius
A = 2 * pi * r * w;     % surface area of cooler
h_copp = 13.1;          % overall cooper transfer coefficient [W/m^2K]
Kcw = A * h_copp;       % chilled water thermal convection coefficient

rho_a = 1.2041;   % air density (20?C) [kg/m^3]
cw_a = 1.012e3;   % specific heat capacity of air [J/kgK]
Cr = (w .* l .* h) * rho_a * cw_a;  % room thermal capacity

% thermal convection coefficients
K1 = ([1,1,1].*w.*h * 2 + [1,0,1].*l.*h) * 0.1;  % closed window
K2 = [1,1,1]* Aw * 0.4;                          % change for open window
Kwall = [1,1,1].*w.*h * 0.2;                     % inner wall

Tset = 20;        % Temperature setpoint
Tout = 30;        % Nominal outside temperature
Tcw  = 10;        % Nominal cooling water temperature

dt=20*60; % period length in seconds
VarTA=1/65^2; % Disturbance variance
Sigma=VarTA*dt; % Resulting Sigma

% truncating the state space to boxes
thr = [2, 2, 2];
X_l = Tset-thr;
X_u = Tset+thr;
% bins per dimension
bins = 5;
n = [bins,bins,bins];
% diameter of bins
delta = (X_u-X_l)./n;
% boundary points of the partition
X = zeros(rooms,bins+1);
X_rep = zeros(rooms,bins);
```

```
for r = 1:rooms
    X(r,:) = X_l(r):delta(r):X_u(r);
    X_rep(r,:) = X(r,1:n(r))+delta(r)/2;
end

% computing transition probabilities
m1 = 5; % control options (valve)
m2 = 2; % control options (window)

% Room 1 -- neighbour is room 2
r = 1;
P1 = zeros(n(r),n(r),n(2),m1,m2);
for u2 = 0:m2-1
    for u1 = 0:m1-1
        for ir=1:n(r)
            for i2=1:n(2)
                % Expected Value of the Room Temperature
                E_xbar=X_rep(r,ir) ...                                      % previous temperature
                    + (dt/Cr(r))*Kwall(r)*(X_rep(2,i2)-X_rep(r,ir)) ...  % change due to neighbouring room
                    + (dt/Cr(r))*u1/(m1-1)*Kcw(r)*(Tcw-X_rep(r,ir)) ...  % change due to valve setting
                    + (dt/Cr(r))*(K1(r)+K2(r)*u2)*(Tout-X_rep(r,ir));      % change due to window setting
                T = normcdf(X(r,:),E_xbar,Sigma);
                tn = (T(2:n(r)+1) - T(1:n(r)));
                P1(1:n(r),ir,i2,u1+1,u2+1) = tn/sum(tn); % normalization because of truncation
            end
        end
    end
end
% error for cost function C(x_1) = [x_1-Ts]^2
h_c1 = 2*max(abs(X_l(r)-Tset),abs(X_u(r)-Tset)); % differential within box
alpha1 = 1 - (dt/Cr(r))*Kwall(r) - (dt/Cr(r))*Kcw(r) - (dt/Cr(r))*(K1(r)+K2(r));
alpha2 = (dt/Cr(r))*Kwall(r);
error1 = h_c1*delta(r) + h_c1*(abs(alpha1)*delta(r) + abs(alpha2)*delta(2))/(1-abs(alpha1));


% Room 2 -- neighbours are room 1 and 3
r = 2;
P2 = zeros(n(r),n(r),n(1),n(3),m1,m2);
for u2 = 0:m2-1
    for u1 = 0:m1-1
        for ir=1:n(r)
            for i1=1:n(1)
                for i3=1:n(3)
                    % Expected Value of the Room Temperature
                    E_xbar=X_rep(r,ir) ...                                      % previous temperature
                        + (dt/Cr(r))*Kwall(r)*(X_rep(1,i1)-X_rep(r,ir)) ...  % change due to neighbouring room 1
                        + (dt/Cr(r))*Kwall(r)*(X_rep(3,i3)-X_rep(r,ir)) ...  % change due to neighbouring room 3
                        + (dt/Cr(r))*u1/(m1-1)*Kcw(r)*(Tcw-X_rep(r,ir)) ...  % change due to valve setting
                        + (dt/Cr(r))*(K1(r)+K2(r)*u2)*(Tout-X_rep(r,ir));      % change due to window setting
                    T = normcdf(X(r,:),E_xbar,Sigma);
                    tn = (T(2:n(r)+1) - T(1:n(r)));
                    P2(1:n(r),ir,i1,i3,u1+1,u2+1) = tn/sum(tn); % normalization because of truncation
                end
            end
        end
    end
end
% error for cost function C(x_2) = [x_2-Ts]^2
h_c2 = 2*max(abs(X_l(r)-Tset),abs(X_u(r)-Tset)); % differential within box
alpha1 = (dt/Cr(r))*Kwall(r);
alpha2 = 1 - 2*(dt/Cr(r))*Kwall(r) - (dt/Cr(r))*Kcw(r) - (dt/Cr(r))*(K1(r)+K2(r));
alpha3 = (dt/Cr(r))*Kwall(r);
error2 = h_c2*delta(r) + h_c2*(abs(alpha1)*delta(1) + abs(alpha2)*delta(r) + abs(alpha3)*delta(3))/(1-abs(alpha2)
    );

% Room 3 -- neighbour is room 2
r = 3;
P3 = zeros(n(r),n(r),n(2),m1,m2);
for u2 = 0:m2-1
    for u1 = 0:m1-1
        for ir=1:n(r)
            for i2=1:n(2)
                % Expected Value of the Room Temperature
                E_xbar=X_rep(r,ir) ...                                      % previous temperature
                    + (dt/Cr(r))*Kwall(r)*(X_rep(2,i2)-X_rep(r,ir)) ... % change due to neighbouring room
                    + (dt/Cr(r))*u1/(m1-1)*Kcw(r)*(Tcw-X_rep(r,ir)) ... % change due to valve setting
                    + (dt/Cr(r))*(K1(r)+K2(r)*u2)*(Tout-X_rep(r,ir));     % change due to window setting
                T = normcdf(X(r,:),E_xbar,Sigma);
                tn = (T(2:n(r)+1) - T(1:n(r)));
                P3(1:n(r),ir,i2,u1+1,u2+1) = tn/sum(tn); % normalization because of truncation
            end
        end
    end
end
% error for cost function C(x_3) = [x_3-Ts]^2
h_c3 = 2*max(abs(X_l(r)-Tset),abs(X_u(r)-Tset)); % differential within box
alpha2 = (dt/Cr(r))*Kwall(r);
alpha3 = 1 - (dt/Cr(r))*Kwall(r) - (dt/Cr(r))*Kcw(r) - (dt/Cr(r))*(K1(r)+K2(r));
error3 = h_c3*delta(r) + h_c3*(abs(alpha2)*delta(2) + abs(alpha3)*delta(r))/(1-abs(alpha3));

%======================Export to PRISM==========================

% MODEL FILE
fid = fopen('temperature.prism','w');

% smg preamble
fprintf(fid,'smg\r\n');
```

```
% top level system
fprintf(fid,'\r\n// top level system\r\n');
fprintf(fid,'system\r\n');
fprintf(fid,'\t"S1" || "S2" || "S3" \r\n');
fprintf(fid,'endsystem\r\n');

comm_seq = [1,2,3];
export_room(fid, P1, 1, Tset, X_rep, 2, -1, comm_seq, error1);
export_room(fid, P2, 2, Tset, X_rep, 1,  3, comm_seq, error2);
export_room(fid, P3, 3, Tset, X_rep, 2, -1, comm_seq, error3);

% time progress
fprintf(fid,'\r\n// time\r\n');
fprintf(fid,'rewards "time"\r\n');
for i1=1:n(1)
    fprintf(fid,'\t[temp2_%02u] true : 1;\r\n', i1);
end
fprintf(fid,'endrewards\r\n\r\n');

fclose(fid);

% PROPERTIES FILE
pid = fopen('temperature.props','w');
for r=1:rooms
    fprintf(pid, 'const double tv%u = 0.8;\r\n', r);
    fprintf(pid, 'const double va%u = 0.8;\r\n', r);
    fprintf(pid, 'const double tw%u = 0.3;\r\n', r);
    if r==2
        fprintf(pid, 'const double wi%u = 0.2;\r\n', r);
    else
        fprintf(pid, 'const double wi%u = 0.3;\r\n', r);
    end
end
fprintf(pid, '\r\n');

% room 1
fprintf(pid, '"phi1" : <<1>> ((R{"window1"}/{"time"}<=wi1 [ S ] & R{"tempdev2"}/{"time"}<=tw2 [ S ]) => R{"
    tempdev1"}/{"time"}<=tw1 [ S ])\r\n');
fprintf(pid, '"psi1" : <<1>> (R{"valve1"}/{"time"}<=va1 [ S ] & (R{"tempdev2"}/{"time"}<=tv2 [ S ] => R{"tempdev1
    "}/{"time"}<=tv1 [ S ]))\r\n');
% room 2
fprintf(pid, '"phi2" : <<1>> (R{"window2"}/{"time"}<=wi2 [ S ] => R{"tempdev2"}/{"time"}<=tw2 [ S ])\r\n');
fprintf(pid, '"psi2" : <<1>> (R{"valve2"}/{"time"}<=va2 [ S ] & R{"tempdev2"}/{"time"}<=tv2 [ S ])\r\n');
% room 3
fprintf(pid, '"phi3" : <<1>> ((R{"window3"}/{"time"}<=wi3 [ S ] & R{"tempdev2"}/{"time"}<=tw2 [ S ]) => R{"
    tempdev3"}/{"time"}<=tw3 [ S ])\r\n');
fprintf(pid, '"psi3" : <<1>> (R{"valve3"}/{"time"}<=va3 [ S ] & (R{"tempdev2"}/{"time"}<=tv2 [ S ] => R{"tempdev3
    "}/{"time"}<=tv3 [ S ]))\r\n');

% compositional
fprintf(pid, '\r\n"phi" : comp("phi1", "phi2", "phi3")\r\n');
fprintf(pid, '"psi" : comp("psi1", "psi2", "psi3")\r\n');

fclose(pid);

% BASH FILE
sid = fopen('temperature.sh','w');
acc = [100, 500, 1000];
eps = [0.05,0.02, 0.01];
citer = [100,250];
diter = [20,10];
name = ['phi'; 'psi'];
fprintf(sid, 'TIMEOUT=240\n\n'); % timeout in minutes
fprintf(sid, 'if hash timeout 2>/dev/null; then\n\tTO=timeout\nelse\n\tTO=gtimeout\nfi\n\n'); % timeout program
fprintf(sid, '{\n'); % sequencing
for j = 1:2
    for i = 1:3
        fprintf(sid, '$TO $((TIMEOUT))m \\\n'); % timeout
        fprintf(sid, '../../prism/bin/prism temperature{.prism,.props} \\\n'); % model and properties
        fprintf(sid, '\t-prop %u \\\n', 6+j); % property index
        fprintf(sid, '\t-multimaxciter %u -multimaxditer %u -gs \\\n', citer(j), diter(j)); % iteration bounds
            and Gauss-Seidel
        fprintf(sid, '\t-multiminm 2 -multimaxm 10000 \\\n'); % box size
        fprintf(sid, '\t-baselineaccuracy %u -increasefactor 1.0 -multirounding \\\n', acc(i)); % rounding
        fprintf(sid, '\t-logcpareto -logdpareto \\\n'); % logging
        fprintf(sid, '\t-exportstrat temp_%s_%u.strat \\\n', name(j,:), i); % strategy export
        fprintf(sid, '\t-paretoepsilon %.2f \\\n', eps(i)); % stopping accuracy
        fprintf(sid, '\t2>&1 1> temp_%s_%u.log ; \n\n', name(j,:), i);
    end
end
fprintf(sid, '} &\n'); % end sequencing and execute in background
fclose(sid);

%====================End of the code==============================
```

export_room.m

```
function [] = export_room(fid, P, roomID, Tset, X_rep, neighborID1, neighborID2, comm_seq, error)

if(neighborID2 < 1) % only one neighbour
    ns = 1;
else
    ns = 2;
end
```

```
if(ns==1)
    n1 = size(P,1);
    n2 = size(P,3);
    n3 = 1;
    m1 = size(P,4);
    m2 = size(P,5);
else
    n1 = size(P,1);
    n2 = size(P,3);
    n3 = size(P,4);
    m1 = size(P,5);
    m2 = size(P,6);
end

fprintf(fid,'\r\n////////////////////// ROOM %u ////////////////////////////////\r\n', roomID);
fprintf(fid,'\r\nsystem "S%u"\r\n', roomID);
fprintf(fid,'\tG%u\r\n', roomID);
fprintf(fid,'endsystem\r\n');

fprintf(fid,'\r\nmodule G%u', roomID);
fprintf(fid,'\r\n\t// discretisation error estimate: %f\r\n\r\n', error);

% Definition of the states
fprintf(fid,'\t// state variables\r\n');
fprintf(fid,'\tx%u : [1..%u] init 1; // room %u temperature\r\n', roomID, n1, roomID);
fprintf(fid,'\ty%u : [1..%u] init 1; // ambient temperature (room %u)\r\n', roomID, n2, neighborID1);
if ns==2
    fprintf(fid,'\tz%u : [1..%u] init 1; // ambient temperature (room %u)\r\n', roomID, n3, neighborID2);
end
fprintf(fid,'\tv%u  : [0..%u] init 0;  // valve setting\r\n', roomID, m1-1);
fprintf(fid,'\tw%u  : [0..%u] init 0;  // window setting\r\n', roomID, m2-1);
fprintf(fid,'\tp%u  : [1..%u] init %u;  // stage\r\n', roomID, 5+ns, 2);

% work out communication sequence
c1 = 4+find(comm_seq==roomID);
c2 = 4+find(comm_seq==neighborID1);
c3 = 4+find(comm_seq==neighborID2);
if(ns==1)
    if ((c1==6) && (c2==7))
        c1=5; c2=6;
    elseif ((c1==7) && (c2==6))
        c1=6; c2=5;
    elseif ((c1==5) && (c2==7))
        c2=6;
    elseif ((c1==7) && (c2==5))
        c1=6;
    end
end

% Stage 4: player 2 sets window
fprintf(fid,'\r\n\t// Stage 4: set window\r\n');
for u2 = 0:m2-1
    fprintf(fid,'\t[a%u_w%u?]    p%u=4 -> (w%u''=%u) & (p%u''=5);\r\n',roomID,u2, roomID, roomID,u2, roomID);
end

% Stage 3: player 1 sets valve
fprintf(fid,'\r\n\t// Stage 3: set valve\r\n');
for u1 = 0:m1-1
    fprintf(fid,'\t[a%u_v%u!]    p%u=3 -> (v%u''=%u) & (p%u''=4);\r\n',roomID,u1, roomID, roomID,u1, roomID);
end

% Stage c1: player 2 sends
fprintf(fid,'\r\n\t// Stage %u: send temperature\r\n', c1);
for i1 = 1:n1
    % Stage send_stage: send over interface
    fprintf(fid,'\t[temp%u_%02u?] x%u=%u & p%u=%u -> (p%u''=%u);\r\n',roomID,i1, roomID,i1, roomID,c1, roomID,mod
        (c1,ns+5)+1);
end

% Stage c2: player 2 receives from neighbour 1
fprintf(fid,'\r\n\t// Stage %u: receive temperature\r\n', c2);
for i2 = 1:n2
    fprintf(fid,'\t[temp%u_%02u?] p%u=%u  -> (y%u''=%u) & (p%u''=%u);\r\n', neighborID1,i2, roomID,c2, roomID,i2,
        roomID,mod(c2,ns+5)+1);
end

% Stage c3: player 2 receives from neighbour 2
if ns==2
    fprintf(fid,'\r\n\t// Stage %u: receive temperature\r\n', c3);
    for i2 = 1:n2
        fprintf(fid,'\t[temp%u_%02u?] p%u=%u  -> (y%u''=%u) & (p%u''=%u);\r\n', neighborID2,i2, roomID,c3, roomID
            ,i2, roomID,mod(c3,ns+5)+1);
    end
end

% Stage 1: temperature dynamics
fprintf(fid,'\r\n\t// Stage 1: temperature dynamics\r\n');
for i1=1:n1
    for i2=1:n2
        for i3=1:n3
            for u2 = 0:m2-1
                for u1 = 0:m1-1
                    % use rounded distribution
                    if ns==1
                        prob = round(P(1:n1,i1,i2,u1+1,u2+1)*1000)/1000;
                        fprintf(fid,'\t[temp%u?]    x%u=%u & y%u=%u & v%u=%u & w%u=%u & p%u=1 -> ',roomID, roomID
                            ,i1, roomID,i2, roomID,u1, roomID,u2, roomID);
                    elseif ns==2
                        prob = round(P(1:n1,i1,i2,i3,u1+1,u2+1)*1000)/1000;
```

```
                    fprintf(fid,'\t[temp%u?]    x%u=%u & y%u=%u & z%u=%u & v%u=%u & w%u=%u & p%u=1 -> ',
                        roomID, roomID,i1, roomID,i2, roomID,i3, roomID,u1, roomID,u2, roomID);
                end
                % re-normalise rounded distribution
                if(sum(prob)~=1.0)
                    [val,index] = max(prob);
                    prob(index) = val+(1.0-sum(prob));
                end

                % Stage 5: set temperature (one action per state)
                trans_added = false;
                for i1n=1:n1
                    if prob(i1n) > 0.0
                        if trans_added
                            fprintf(fid, ' + ');
                        end
                        trans_added = true;
                        fprintf(fid,'%.3f : (x%u''=%u) & (p%u''=2)',prob(i1n), roomID,i1n, roomID);
                    end
                end
                fprintf(fid,';\r\n');
            end
        end
    end
    fprintf(fid,'\r\n');
end

% Stage 2: tau for normal form
fprintf(fid,'\r\n\t// Stage 2: tau for normal form\r\n');
fprintf(fid,'\t[tau?]      p%u=2        -> (p%u''=3);\r\n',roomID, roomID);

fprintf(fid,'\r\nendmodule\r\n\r\n');


% Rewards

% temperature of room
fprintf(fid,'\r\n// temperature room %u\r\n', roomID);
fprintf(fid,'rewards "temp%u"\r\n', roomID);
for i1=1:n1
    fprintf(fid,'\t[temp%u_%02u] true : %.3f;\r\n', roomID, i1, X_rep(roomID,i1));
end
fprintf(fid,'endrewards\r\n\r\n');

% temperature deviation of room
fprintf(fid,'\r\n// temperature deviation room %u\r\n', roomID);
fprintf(fid,'rewards "tempdev%u"\r\n', roomID);
for i1=1:n1
    fprintf(fid,'\t[temp%u_%02u] true : %.3f;\r\n', roomID, i1, (Tset - X_rep(roomID,i1))^2);
end
fprintf(fid,'endrewards\r\n\r\n');

% valve setting
fprintf(fid,'\r\n// valve room %u open\r\n', roomID);
fprintf(fid,'rewards "valve%u"\r\n', roomID);
for i1=0:m1-1
    fprintf(fid,'\t[a%u_v%u] true : %.3f;\r\n', roomID, i1, i1/(m1-1));
end
fprintf(fid,'endrewards\r\n\r\n');

% window setting
fprintf(fid,'\r\n// window room %u open\r\n', roomID);
fprintf(fid,'rewards "window%u"\r\n', roomID);
for i2=0:m2-1
    fprintf(fid,'\t[a%u_w%u] true : %.3f;\r\n', roomID, i2, i2/(m2-1));
end
fprintf(fid,'endrewards\r\n\r\n');
```