Formal Techniques for the Verification and Optimal Control of Probabilistic Systems in the Presence of Modeling Uncertainties

by

Alberto Alessandro Angelo Puggelli

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Engineering - Electrical Engineering and Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Alberto L. Sangiovanni-Vincentelli, Chair
Professor Elad Alon
Professor Sanjit A. Seshia
Professor Shmuel Oren

Fall 2014

Formal Techniques for the Verification and Optimal Control of Probabilistic Systems in the Presence of Modeling Uncertainties

Abstract

Formal Techniques for the Verification and Optimal Control of Probabilistic Systems in the Presence of Modeling Uncertainties

by

Alberto Alessandro Angelo Puggelli

Doctor of Philosophy in Engineering - Electrical Engineering and Computer Science

University of California, Berkeley

Professor Alberto L. Sangiovanni-Vincentelli, Chair

We present a framework to design and verify the behavior of stochastic systems whose parameters are not known with certainty but are instead affected by modeling uncertainties, due for example to modeling errors, non-modeled dynamics or inaccuracies in the probability estimation. Our framework can be applied to the analysis of intrinsically randomized systems (e.g., random back off schemes in wireless protocols) and of abstractions of deterministic systems whose dynamics are interpreted stochastically to simplify their representation (e.g., the forecast of wind availability).

In the first part of the dissertation, we introduce the model of Convex Markov Decision Processes (Convex-MDPs) as the modeling framework to represent the behavior of stochastic systems. Convex-MDPs generalize MDPs by expressing state-transition probabilities not only with fixed realization frequencies but also with non-linear convex sets of probability distribution functions. These convex sets represent the uncertainty in the modeling process.

In the second part of the dissertation, we address the problem of formally verifying properties of the execution behavior of Convex-MDPs. In particular, we aim to verify that the system behaves correctly under all valid operating conditions and under all possible resolutions of the uncertainty in the state-transition probabilities. We use Probabilistic Computation Tree Logic (PCTL) as the formal logic to express system properties. Using results on strong duality for convex programs, we present a model-checking algorithm for PCTL properties of Convex-MDPs, and prove that it runs in time polynomial in the size of the model under analysis. The developed algorithm is the first known polynomial-time algorithm for the verification of PCTL properties of Convex-MDPs. This result allows us to lower the previously known algorithmic complexity upper bound for Interval-MDPs from co-NP to P, and it is valid also for the more expressive (convex) uncertainty models supported by the Convex-MDP formalism.

We apply the proposed framework and model-checking algorithm to the problem of formally verifying *quantitative* properties of models of the behavior of human drivers. We first propose a novel stochastic model of the driver behavior based on Convex Markov chains. The model is capable of capturing the intrinsic uncertainty in estimating the intricacies of the human behavior starting from experimentally collected data. We then formally verify properties of the model ex-

pressed in PCTL. Results show that our approach can correctly predict quantitative information about the driver behavior depending on his/her attention state, e.g., whether the driver is attentive or distracted while driving, and on the environmental conditions, e.g., the presence of an obstacle on the road.

Finally, in the third part of the dissertation, we analyze the problem of synthesizing optimal control strategies for Convex-MDPs, aiming to optimize a given system performance, while guaranteeing that the system behavior fulfills a specification expressed in PCTL under all resolutions of the uncertainty in the state-transition probabilities. In particular, we focus on Markov strategies, i.e., strategies that depend only on the instantaneous execution state and not on the full execution history. We first prove that adding uncertainty in the representation of the state-transition probabilities does not increase the theoretical complexity of the synthesis problem, which remains in the class NP-complete as the analogous problem applied to MDPs, i.e., when all transition probabilities are known with certainty. We then interpret the strategy-synthesis problem as a constrained optimization problem and propose the first sound and complete algorithm to solve it.

We apply the developed strategy-synthesis algorithm to the problem of generating optimal energy pricing and purchasing strategies for a for-profit energy aggregator whose portfolio of energy supplies includes renewable sources, e.g., wind. Economic incentives have been proposed to manage user demand and compensate for the intrinsic uncertainty in the prediction of the supply generation. Stochastic control techniques are however needed to maximize the economic profit for the energy aggregator while quantitatively guaranteeing quality-of-service for the users. We use Convex-MDPs to model the decision-making scenario and train the models with measured data, to quantitatively capture the uncertainty in the prediction of renewable energy generation. An experimental comparison shows that the control strategies synthesized using the proposed technique significantly increase system performance with respect to previous approaches presented in the literature.

If the whole world I once could see
On free soil stand, with the people free
Then to the moment might I say,
Linger awhile... so fair thou art.

Johann Wolfgang von Goethe, *Faust*

Contents

List of Figures

List of Tables

Acknowledgments

My experience as a Ph.D. student has lasted longer than I would have ever imagined and it has surprised me in every single moment. More than anything else, it has been incredibly different from how I pictured it in my mind the first time I stepped inside Cory Hall, the house of the Electrical Engineering department at UC Berkeley. Being an engineer, probably more deeply in my heart than what I dare to admit, I have a strong aversion to surprises. But the past five years have been, by far, the most intense of my life, an experience that I will never forget. Interestingly, looking back at these years, I now realize that each single event has been instrumental to bring me where I am now, even though it was hard to see it at that time. And I certainly made it up to writing this dissertation because of my stubbornness, but also thanks the great people that I met along these years. I have always liked meeting and knowing new people, and Berkeley has allowed me to meet some of the most exciting people that I have ever had the pleasure to spend time with. The next few paragraphs are dedicated to them.

First, I would like to thank my adviser prof. Sangiovanni-Vincentelli for all the inspiration that his visionary ideas about the future of the design and control of cyber-physical systems have given to me along these years. He has been the first one to believe in me in Berkeley. He has also opened up for me a wide variety of opportunities and has enabled all the great collaborations with researchers both in Berkeley and in other American institution that have made my experience in Berkeley unique.

Simply to follow the chronological order in joining the research group, I would like to thank second my co-adviser prof. Elad Alon. I have no difficulty to say that I had never met a sharper person in my life before and interacting with him has always been a pleasant and rewarding intellectual challenge. I believe that the intensity and the high technical content of the conversations with him have made me a better engineer, and I am deeply grateful to him for this. And his guidance has gone beyond the technical aspects of my graduate career.

Although not officially an adviser of mine, I would like to express my deepest gratitude to prof. Sanjit Seshia. He has been a brilliant guidance for all the most important projects that I have worked on during my studies and his careful review of all stages of my work has always been extremely useful to improve the quality of the final results. In fact, I owe to a comment of his the intuition that brought to the development of the polynomial-time model-checking algorithm described in the dissertation.

I would also like to thank prof. Seth Sanders for serving in my qualifying exam committee, for the constant support and enthusiasm shown for my research and for his great class on power electronics. Moreover, my gratitude goes to prof. Shmuel Oren for serving on my qualifying exam and dissertation committee and for his precious comments and explanations on the structure of the energy market. I believe that the technical relevance of my dissertation has substantially increased thanks to his help.

There are many reasons why UC Berkeley is considered one of the best universities in the world, but one of them is certainly the exceptional quality of the faculties working here. I would like to thank all the instructors of the classes that I have taken because each of them has enriched my technical knowledge. Moreover, I got inspired by the incredible passion that all the instructors

have always put in their job. In particular, I would like to thank prof. Jaijeet Roychowdhury, prof. Andreas Kuehlmann, prof. James Demmel, prof. Haideh Khorramabadi, prof. Simone Gambini, prof. El Ghaoui and prof. Vladimir Stojanovic.

I had the fortune to work, collaborate or at least interact with many more professors at Berkeley and in other universities. In particular, my thanks also go to prof. Edward Lee, prof. Luca Carloni, prof. Borivoje Nikolić and prof. Andrei Vladimirescu. Special thanks to prof. David Parker to help me integrate the algorithms presented in this dissertation within the PRISM Model Checker.

Also, I would like to thank my managers and co-workers during the two internships at Texas Instruments. Their guidance and vision have deeply motivated me and have shown me how great of an environment a big company in the US can be. Thanks in particular to Fernando Mujica, Tim Fischer, Tom Vrotsos, Yonghui Tang, Chika Soh, Ajith Amerasekera, Arthur Redfern.

Thanks also to the Lion Semiconductor team, Wonyoung Kim, Hanh-Phuc Le, John Crossley, Tao Tong, Fred Chen, Thomas Li. I am quite excited about the future of our adventure!

One of the most positive experiences during my Ph.D. has been the possibility to interact and work with incredibly talented and motivated students. Their passion and knowledge have enriched me professionally. Moreover, and more importantly, they have proved to be extremely nice people from a personal perspective.

I would like to thank first John Crossley and Hanh-Phuc Le. They have been exceptional co-workers and friends outside school. And what we have done together so far might just have been the beginning of a much longer adventure.

My thanks also go to Brian Zimmer. Honestly, he has been the person that I have had the most pleasure working with during these years for his exceptional punctuality and precision and great attitude towards solving the many difficulties that we faced.

Many thanks also to Wenchao Li, Dorsa Sadigh and Katie Driggs-Campbell, the co-authors of the publications reported in this dissertation, for the precious help in developing these technical results and for being great friends also outside school. It has been a great pleasure working and celebrating with you!

Thank you to Matt Spencer, for all his wisdom, for all the lessons of American culture and the fun during the summer in Dallas, and for implicitly reminding me in almost all our conversations that I still have a long way before being really able to understand spoken English. It has been great being roommates!

Thanks to Chung-Wei Lin for the help in preparing for the preliminary exam and his exceptional attitude towards problem-solving.

Besides studying for classes, working on research, and trying to have some real life outside school, UC-Berkely requires also to teach two classes to get a doctoral degree. While teaching has in fact being an extremely positive experience, something that I would really like to do again in the future, my gratitude now goes to my two co-teaching assistants Bonjern Yang and Filip Maksimovic for the precious help in fulfilling this requirement. I have learnt a lot from them and I believe we had quite some fun together. Thanks also to the students of the two classes that I taught, ultimately it is their passion and work to make the classes successful.

Being part of two research groups and collaborating with many others, I had the pleasure to interact with a lot more students and researchers, who have also become friends in life. My thanks go

to Rachel Nancollas, Liangpeng Guo, John Finn, Antonio Iannopollo, Nikunj Bajaj, Mehdi Maasoumy, Daniela De Venuto, Michele Petracca, Baruch Sterin, Nathan Narevsky, Nicholas Sutardja, Viki Szortyka, Ali Moin, Greg Lacaille, Luke Calderin, Costis Sideris, Kosta Trotskovsky, DJ Seo, Jaeduk Han, Pengpeng Lu, Seobin Jung, Yue Lu, Lingkai Kong, Yida Duan, Chintan Thakkar, Kwangmo Jung, Kristel Deems, Ben Keller, Stevo Bailey, Yunsup Lee, Milovan Blagojevic, Pi-Feng Chiu, Mohammed Mozumdar, Tobias Welp, Rikky Muller, Steven Callender, Mike Lorek, Fabien Chraim, Sayak Ray, Baruch Sterin, Martin Cochet, all the DOP and BWRC students, in particular the Ravenites and the BAG team.

Thanks to the "steal-the-glass" folks, Filip Maksimovic, Claire Lochner, Nathan Narevsky, David Burnett, Brian Zimmer, Daniel Drew, Brad Wheeler, it was great to collect glasses together all summer long.

Thanks also to all the co-interns during the two summers spent in Dallas working for Texas Instruments. Those experiences have been great largely because of the fun of the time spent together. In particular, thanks to Pradeep Shenoy, Samantha Summerson, Jonathon Spaulding, Douglas Adams, Georgios Angelopoulos, Phil Nadeau.

Many thanks also to the exceptional staff at the DOP center, at BWRC and in the EECS department, in particular to Christopher Brooks, Brian Richards, Bira Coelho, Fred Burghardt, James Dunn, Olivia Nolan, Leslie Nishiyama, Sarah Jordan, Shirley Salanio, Xuan Quach.

I will always be in debt to Eleonora De Re for sharing with me all the challenges and successes of the first years of our graduate carrier. I wish you the best luck for your life!

Thanks also to Elizabeth Liu for all the happiness that she has brought to my life. I will never forget what we have done together.

Thanks also to all my friends in Italy. It has been quite difficult to keep in touch due to the physical distance, but it has been great to know that I could always rely on you to spend awesome time back at home. Thanks in particular to those who have visited, either in Berkeley or somewhere else in the US.

Finally, I would like to thank my family, my mother, father and sister, for the unconditional support in all these years. Being so far away has been a challenge for all of us, but we have shown how great of a family we are by being close to one another in spirit despite the physical distance. Thank you for understanding every time I was too busy to talk or planning to do things together, for coming to visit so often, for all the nice moments spent together, and the fun that you have brought to my life. All these moments have meant a lot to me and I will be always grateful to you for them.

My years of graduate school have been funded by the following research centers and grants.

The author acknowledge the support of the Gigascale Systems Research Center, one of six research centers funded under the Focus Center Research Program (FCRP), a Semiconductor Research Corporation program.

This work was supported in part by the TerraSwarm Research Center, one of six centers supported by the STARnet phase of the Focus Center Research Program (FCRP) a Semiconductor Research Corporation program sponsored by MARCO and DARPA.

The research was partially funded by DARPA Award Number HR0011-12-2-0016.

Chapter 1

Introduction

We begin the chapter by introducing the motivations behind the work described in this dissertation. We then give an overview of the key analyzed topics and summarize the main achieved contributions. Finally, we outline the content of the rest of the dissertation. While the discussion in the following of the chapter is kept informal for ease of exposition, this material should be considered as a guide to facilitate the understanding of the presented material and to introduce also the non-expert reader to the analyzed subject.

1.1 Motivations

A system is referred to as stochastic if its dynamics exhibit a probabilistic behavior, i.e., the state is only known probabilistically, according to some probability density distribution. The study of stochastic systems has great importance for at least three reasons. First, some systems are indeed stochastic in nature because of the presence of actual randomization (for example, random back-off schemes in wireless transmission protocols). Second, they represent a powerful modeling tool to abstract complex deterministic behaviors (for example, the electricity power demand of a city neighborhood can be captured by its mean and standard deviation varying during the day, since capturing the actual deterministic data would be too computationally expensive). Third, interpreting a system as stochastic is strictly more expressive than a deterministic interpretation, since any deterministic behavior can trivially be represented using a delta probability density distribution. The opposite is instead not true, i.e., a deterministic framework cannot represent stochastic behaviors. In this dissertation, we will analyze formal techniques for the functional verification and optimal control of stochastic systems.

Formal verification addresses the need of exhaustively validating the correct functionality of systems that exhibit too many behaviors (due to their complexity) to be fully tested through simulation. Since the cost of putting in the market a faulty product, and then recall it to fix it, is

unbearable for most businesses, formal verification is playing an increasingly bigger role in the development flow of electronic systems. Formal verification techniques for deterministic systems have already reached a reasonable level of maturity, although some level of domain-specific expertise is still often required to abstract the main system behaviors and improve the scalability of the analysis. For example, Electronic Design Automation (EDA) tools are capable of verifying the functionality of digital systems containing billions of integrated devices (e.g., the central processing units of computers), shortening design iteration cycles and empowering the digital revolution era. Although techniques for the verification of the behavior of stochastic systems have not reached the same maturity of their deterministic counterpart, the research community has recently shown an increasing interest on this topic, due to its relevance in many practical applications.

Techniques to control the behavior of a given system (e.g., a plant) have been for centuries at the heart of automation. For example, linear proportional-integral-derivative (PID) controllers have been employed since the XIX century to regulate the speed of the first mechanical engines. Given the present trend in system design towards performance maximization (e.g., maximize the computational capabilities of a digital accelerator) with constrained resource availability (e.g., power), the simple control of a system might no longer be enough. With optimal control, we refer to techniques capable of exhaustively searching the space of available system parameters to synthesize control strategies that achieve the best performance given a set of formally defined constraints on the available resources. While the field of optimal control for linear and non-linear (but well-behaved) systems whose behavior can be captured by ordinary differential equations (ODEs) has been exhaustively researched, there is currently a strong interest in extending the developed techniques also to the optimal control of stochastic systems.

Algorithms for functional verification and for the synthesis of optimal control strategies operate on *models* of the system under analysis, so the guarantees that they generate can only be as accurate as the models themselves. When analyzing stochastic systems, the activity of model creation is particularly critical since model parameters, e.g., state-transition probabilities, capture the dynamics of physical systems or of abstractions of complex deterministic systems, and are often estimated only through a finite number of experimental observations. As a consequence, the value of these parameters can be affected by *uncertainties*, due for example to unmodeled dynamics, measurement errors or approximations of the real system by mathematical models. It would thus beneficial to employ functional verification and optimal control methods capable of taking the uncertainties in the estimation of the model parameters into account.

Motivated by these considerations, in this dissertation we will present formal techniques for the functional verification and optimal control of stochastic systems capable of producing results that are *robust* to uncertainties in the estimation of model parameters.

1.2 Dissertation Overview

Central to the human process of understanding the behavior of the environment around us is the creation of models. If we aim to formally verify and control the performance of stochastic systems, we further need a “language” to unequivocally specify the required system properties. In fact, in

the last two decades, a number of mathematical formalisms to capture the behavior of stochastic systems and formally define their expected performance have been introduced. In Chapter 2, we will introduce the analyzed formalisms to model and specify properties of stochastic systems.

Specifically, we will focus on two modeling formalisms (and their extensions), Discrete-Time Markov Chains (DTMCs) [46] and Markov Decision Processes (MDPs) [26], widely used to formally represent systems that exhibit random or probabilistic behaviors. Properties of deterministic systems can be assessed using *qualitative* analysis which returns a Boolean answer, "yes" or "no", depending on whether the property is satisfied or not. For example, we might be interested in determining whether "the hand-shaking protocol will terminate within 3 clock cycles". Stochastic systems, on the other hand, need *quantitative* analysis [98] to answer questions such as "what is the probability that a request will be eventually served?". Properties of these systems can be expressed and analyzed using logics such as Probabilistic Computation Tree Logic (PCTL) [72] — a probabilistic logic derived from CTL which includes a probabilistic operator $\boldsymbol{P}$ and a reward operator $\boldsymbol{R}$.

Central to the content of this dissertation is the consideration that any formal guarantee on the correct functionality and controlled performance of the system under analysis can be only as good as the *model* of the system on which the verification and the control synthesis routine was run. In particular, part of the presented work aims to provide formalisms to represent errors or uncertainties in the modeling process, so that the generated guarantees can account for them.

In particular, one critical step in the modeling of probabilistic systems is the estimation of state transition probabilities. When modeling randomized protocols (e.g., a randomized consensus protocol), transition probabilities are usually inferred from the ideal behavior of the system. For example, a probability of 0.5 is assigned to the two possible results of a coin toss. On the other hand, when modeling a stochastic physical system (e.g., the quality of a wireless channel), transition probabilities are inferred by performing a measurement campaign and by computing the occurrence frequencies of each possible (discretized) observation of the physical phenomenon. For example, if 4 out of 10 packets sent across a wireless channel drop, the probability of successful transmission will be estimated to be 0.6. While widely adopted for their intrinsic simplicity and computational tractability, these techniques for the estimation of transition probabilities might fail to capture some critical behavior of the system under analysis. For randomized protocols, faulty agents or agents under a security attack might fail to behave as in the ideal scenario. Continuing with the example of the coin toss, a security attack might translate into an equivalent *biased* coin toss, in which, for example, expected probabilities might be 0.4 for "tail" and 0.6 for "head". In the estimation of physical processes, on the other hand, the finite precision in taking measurements and the finite number of measurements that can be taken in a practical scenario limit the accuracy of the inferred transition probabilities and hence introduce uncertainties in the modeling process.

Formal statistical techniques to capture the uncertainty in the estimation of transition probabilities do exist and they have been widely studied in the statistics and optimal control community [125]. Most of these techniques assume that the transition probabilities are not known with precision but only lie in an uncertainty set of potentially *observable* probabilities. To keep computation tractable, these uncertainty sets are usually convex, e.g., a closed interval, an ellipsoid or a likelihood region. Also in the verification community, the concept of transition uncertainty has

been proposed. Interval-valued Discrete-Time Markov Chains (IDTMCs) have been introduced to capture modeling uncertainties [93]. IDTMCs are DTMC models where each transition probability is assumed to lie within a compact range. Two semantic interpretations have been proposed for these models [150]: Uncertain Markov Chains (UMCs) and Interval Markov Decision Processes (IMDPs). A UMC is interpreted as a family of (possibly uncountably many) DTMCs, where each member of the family is a DTMC whose transition probabilities lie within the interval range given in the UMC. In IMDPs, the uncertainty is resolved through non-determinism. Each time a state is visited, a transition distribution within the interval range is adversarially picked, and a probabilistic step is taken accordingly. Thus, IMDPs allow modeling a non-deterministic choice made from a set of (possibly uncountably many) choices. For both semantics, the verification problem amounts to determine whether a desired property holds also under the worst-case resolution of uncertainty. In this dissertation, we will not consider UMCs and focus on IMDPs. From a modeling standpoint, IMDP semantics represents the worst-case scenario of UMCs, since in IMDPs a new adversarial state-transition probability distribution is chosen at each step, while in UMCs the adversarial transition probability distribution is chosen only once. Further the development of verification algorithms for UMCs has been proven by Chatterjee et al. [38] to be harder than for IMDPs, so UMCs are less amenable to a scalable analysis.

An interval model of uncertainty may appear to be the most intuitive to analyze. However, there may be significant advantages in being able to accommodate more expressive (and less pessimistic) uncertainty sets in addition to intervals. Bental et al. analyzed a financial portfolio optimization case-study in which uncertainty arises from estimating the rate of return for each asset [23]. The authors claim that the interval model of uncertainty is too conservative in this scenario, because it would suggest to invest the whole capital into the asset with the largest worst-case return. The ellipsoidal model of uncertainty proposed in that paper returns instead a solution that spreads the capital across multiple assets, a more profitable strategy in the long run. As a further example, Kreinovich et al. [94] consider the problem of estimating transition probabilities from measurements. If the probabilities $\boldsymbol{x}$ are normally distributed with mean $\boldsymbol{\mu}$, and covariance matrix Σ, the ellipsoid defined by $(\boldsymbol{x}-\boldsymbol{\mu})^T\Sigma^{-1}(\boldsymbol{x}-\boldsymbol{\mu}) \leq 1$ is a valuable approximation that eliminates "almost" impossible outliers, which would otherwise be included by an interval approximation. Further, depending on the field, researchers use different models to represent uncertainty. Maximum likelihood models are often used, for example, to estimate chemical reaction parameters [11]. Entropy models are instead used when the available data points are limited or it is costly to collect them, e.g., in text segmentation [118] and political vote prediction [63].

In order to increase the expressiveness of the model under verification, we will thus introduce in Chapter 2 the model of *Convex-MDP (CMDP)*, i.e., an MDP whose state transition probabilities are only known to lie within *convex* uncertainty sets.

Once the system under analysis has been appropriately modeled, it is possible to formally verify its properties and to synthesize optimal control strategies to maximize its performance. Chapter 4 and Chapter 6 will present theoretical and algorithmic results on the problem of developing techniques for the functional verification and optimal control of stochastic systems, respectively.

In the area of formal verification, we will focus on the popular technique of *model check-*

Table 1.1: Known Upper-Bound on the Complexity of PCTL model checking

Model	**Complexity**	**Reference**
DTMC	P	[72]
IMDP	co-NP	[38]
IMDP	P	[ours]
Convex-MDP	P	[ours]

ing [41]. The *model-checking* problem can be formalized as a decision problem. Given a model M and property ϕ, the model-checking problem answers whether M satisfies property ϕ. If M does not satisfy ϕ, the model-checking algorithm produces a counterexample. This formal verification technique is a rigorous mathematical approach in proving the correctness of a system based on state space exploration, and it has been widely applied both to deterministic [41] and to stochastic [99] systems.

Before the work presented in this dissertation, the upper-bound on the complexity of model checking PCTL properties on IMDPs was known to be PSPACE [150], and the result was later improved to co-NP [38]. These results rely on the construction of an MDP encoding all the behaviors of the IMDP under analysis. For each state in the new MDP, the set of available transitions is mapped to the Basic Feasible Solutions (BFS) of the set of inequalities specifying the transition probabilities of the IMDP. Since, in the worst case, the number of BFS is exponential in the number of states in the IMDP, the equivalent MDP can have size exponential in the size of the IMDP. Given the aforementioned results, no computationally efficient algorithm was known for the verification of formal properties of systems whose transition probabilities were captured by uncertainty sets.

In this dissertation, we present the first-known *polynomial-time algorithm* (in both size of the model and size of the formula) for the verification of IMDP properties expressed using the same fragment of PCTL considered by the authors of [38, 150] (the operators with a finite time horizon are disallowed). This shows that the problem is in the complexity class P (also known as PTIME). We then extend the algorithm to full PCTL (allowing also operators with finite time horizon), and show that its time complexity only increases to pseudo-polynomial in the maximum integer time horizon. Furthermore, we show that the proposed algorithms can be extended to verify Convex-MDPs for a wide and relevant class of convex uncertainty sets (e.g., sets expressed with the ellipsoidal, likelihood and entropy models of uncertainty), while maintaining the same complexity results proven for IMDPs. Moreover, different models of uncertainty can be used within the same Convex-MDP to represent different sources of uncertainty, thus further increasing the expressiveness of the proposed approach. We also note that the complexity results presented in previous work [38, 150] cannot be trivially extended to the verification of Convex-MDPs. This is because BFS are not defined for generic convex inequalities, so the construction of the MDP encoding all the behaviors of the IMDP would not be possible. The complexity results are compared in Table 1.1.

In the area of optimal control, we tackle the problem of synthesizing strategies to maximize

system performances expressed in terms of rewards (equivalently, the same results can be proven for the dual problem of minimizing a cost). We focus on finite-horizon History-dependent Deterministic (HD) strategies, i.e., for each state an optimal action to take is chosen deterministically, based on the entire (finite) execution history of the decision process. Although the limitation to finite-horizon strategies may be restrictive in some applications, control techniques operating on a finite (receding) horizon are widely spread nowadays, for example in the context of model predictive control [62]. As it will be explained in further details in Chapter 3, History-dependent Random (HR) strategies are in general more powerful than deterministic strategies, i.e., they can produce a higher expected reward. Nevertheless, we focus on deterministic strategies since they are usually easier to implement in practical scenarios, like the one presented in Chapter 7, because they guarantee the repeatability of the controller behavior. Using a classical construction [142], we will then show that the problem of synthesizing finite-horizon history-dependent deterministic strategies can be reformulated as the synthesis problem for Markov-deterministic (MD) strategies, i.e., strategies for which an optimal action for each state is taken based only on the current state.

Leveraging results previously described about the formal verification of Convex-MDPs, we will prove that the problem of computing an MD strategy for a Convex-MDP model, with total expected reward higher than a given threshold and constrained to specifications expressed in PCTL, is NP-complete. We will then focus our attention to the *optimization* version of the problem, i.e., maximize the reward of the Convex-MDP while guaranteeing that the model obeys to an arbitrary PCTL specification, and present the first sound and complete algorithm to solve this problem. The key advantage of the proposed algorithm is its capability of *ranking* candidate strategies by the value of their reward. The first proposed strategy that satisfies the PCTL specification for all resolutions of uncertainty is the solution of the synthesis problem. Although the algorithm worst-case running time is exponential in the size of the model, this capability may allow considerable speed-ups in practical scenarios. Finally, we note that, while the presented theoretical and algorithmic results are valid for all models of uncertainty analyzed in this dissertation, the applicability of the proposed algorithm is, in fact, limited to the interval and ellipsoidal uncertainty sets. This is due to the lack of availability, at the time of writing, of off-the-shelf optimization engines capable of solving the convex problems that get formulated when other models of uncertainty, e.g., likelihood or entropy, are considered.

Many other modeling formalisms and algorithms for the verification and synthesis of control strategies for stochastic systems have been presented in the literature. While a comparison of the newly proposed results with previous work is interesting from a theoretical perspective, the ultimate metric to evaluate the relevance of the proposed work is represented by its capability of enabling an effective analysis and understanding of real-world systems and of their properties. In Chapter 5 and Chapter 7, we will show how we applied the proposed algorithms to two case studies of high practical relevance.

In Chapter 5, we present a model of the performance of individuals while driving a car and *quantitatively* analyze its properties. In particular, our approach enables a *personalized* assessment of the driving performance of each single individual. The applications for the analysis of the human behavior while driving are numerous, ranging from personalized teaching strategies, aiming to

address specific weaknesses of a driver, and personalized rates for car insurance, to, in the longer term, real-time automated assistance to the driver while performing a maneuver. In our approach, we first develop a *stochastic model* of the driver performance, capable of predicting the expected driven trajectories. We use the formalism of Convex Markov Chains (Convex-MCs) to create such a model. The prediction is based on the future environment surrounding the car, the state of the driver (i.e., attentive or distracted), and the history of steering maneuvers for a given individual, which was collected using a car simulator [35]. Further, we express the transition probabilities of the generated Convex-MC model using uncertainty sets, to capture the inherent ambiguity in capturing the complexity of the human behavior using a finite set of measured data. Finally, using PCTL as our formal language, we verify logical properties of the model of the driver. Our main focus is to quantify the effects of different *attention levels* on the quality of driving, by analyzing the driver behaviors while they are either attentive or distracted. For example, we will evaluate the "maximum probability of exiting the road for a distracted driver while performing a complex maneuver like a double turn".

As a second case study, in Chapter 7, we analyze the problem of synthesizing optimal energy pricing and purchasing strategies for a for-profit energy aggregator whose portfolio of energy supplies includes renewable sources, e.g., wind. It is nowadays commonly accepted that the exploitation of renewable sources of energy will be necessary to sustain the ever increasing energy demand while avoiding the irremediable pollution of the environment caused by fossil sources [65]. Nevertheless, many difficulties need to be solved before enabling a full takeover of renewables over fossil supplies. One of these difficulties lies in the unpredictability of the availability of renewable supplies, which forces power-networks and business entities managing the production and delivery of energy to still heavily rely on fossil supplies to guarantee energy availability to users at all times. It is predicted that in the (near) future, smart-grids will employ real-time control systems capable of reacting to changes in renewables availability and allow a higher exploitation of these green resources [167]. We contribute to the effort of developing such real-time control strategies. First, we develop a novel stochastic model to synthesize energy pricing and purchasing strategies to be adopted by an energy aggregator whose portfolio of energy supplies includes renewable sources. The model is an Ellipsoidal-MDP, i.e., a Convex-MDP with ellipsoidal uncertainty sets. We use measured data (from the wind farm at Lake Benton, Minnesota, USA [170]) to train a likelihood model of the wind generation. We then approximate the likelihood region with an ellipsoidal model, which is more accurate than the linear ones often used in the literature, while remaining computationally tractable. Our empirical approach has the promise of more faithfully representing the probability distribution of the generated energy because it is tailored to the specific wind farm under analysis, and it is robust to forecast errors. Second, we cast the constrained optimization problem as the strategy synthesis problem for Ellipsoidal-MDPs, with the goal of maximizing the total expected reward of the model while constraining its behavior to satisfy a PCTL specification. In this specific scenario, the optimization aims to maximize the economical profit of the energy aggregator while guaranteeing the desired quality-of-service for the users.

1.3 Main Contributions

In this dissertation we argue that:

It is fundamental to take uncertainties *in the estimation of model parameters into account when creating models of stochastic processes describing the behavior of physical systems or abstractions of complex deterministic systems, since both model-checking and optimal control results can be highly sensitive to parameter variations. The theoretical and algorithmic techniques presented in the rest of the dissertation enable such robust analysis, and are demonstrated with two relevant case studies.*

We divide the contributions presented in this dissertation into three main categories.

Theory. We present new results about the theoretical complexity of model checking and optimal control strategy synthesis for Convex-MDPs, i.e., MDPs whose transition probabilities are only known to lie in convex uncertainty sets.

First, we prove that the problem of model checking a PCTL property of a Convex-MDP is in the complexity class P (also known as PTIME), if we disallow the operators with a finite time horizon.

Second, we prove that the problem of determining Markov-Deterministic strategies for Convex-MDPs with total expected reward larger or equal to a given value and satisfying a PCTL specification is NP-complete.

Algorithms. We propose scalable algorithms to model check and to optimally control Convex-MDPs.

First, we present the first-known *polynomial-time algorithm* (in both size of the model and size of the formula) for the verification of PCTL properties in which we disallow the operators with a finite time horizon. We then extend the algorithm to full PCTL (allowing also operators with a finite time horizon) and show that the algorithmic time complexity only increases to pseudo-polynomial in the maximum integer time horizon.

Second, we propose the first sound and complete algorithm to synthesize Markov-Deterministic strategies for Convex-MDPs. The synthesized strategies fulfill specifications expressed using the full PCTL syntax and maximize the expected reward of the Convex-MDP (or, dually, they minimize the expected cost).

Applications. Finally, we demonstrate the relevance of our approach by applying the proposed techniques to two case studies.

First, we model and quantitatively analyze properties of the performance of individuals while driving a car. The model is a Convex Markov chain where the uncertainty stems from the ambiguity of capturing the complexity of the human behavior in a finite model. We then analyze quantitative properties of the model like the "maximum probability of exiting the road for a distracted driver while performing a complex maneuver like a double turn".

Second, we analyze the problem of synthesizing optimal energy pricing and purchasing strategies for a for-profit energy aggregator whose portfolio of energy supplies includes renewable sources. We use a Convex-MDP to model the decision-making scenario and aim to maximize the economical profit for the energy aggregator, while guaranteeing the desired quality-of-service

for the users. The synthesized strategies are robust to the uncertainty in the prediction of the availability of renewable sources.

1.4 Dissertation Outline

The rest of the dissertation is organized as follows.

In **Chapter 2**, we introduce the framework used in this dissertation to model the behavior of probabilistic systems. We first formally introduce the concept of Convex-MDPs, as first introduced in the work developed jointly with W. Li, A. L. Sangiovanni-Vincentelli and S. A. Seshia [139]. We then describe the analyzed convex models of uncertainty. Finally, we define the Probabilistic Computation Tree Logic (PCTL), i.e., the logic used in this work to formally express properties of Convex-MDPs.

In **Chapter 3**, we survey related work in the literature both in the fields of formal verification and optimal control for stochastic systems. In this chapter, we focus on the theoretical results presented in the literature, while we postpone the presentation of the related work about the case studies to the corresponding chapters. We will first review several formalisms proposed in the literature to capture the behavior and express properties of stochastic systems. We then report techniques for the functional verification and control-strategy synthesis of these systems.

In **Chapter 4**, we present the main theoretical results about the model checking of PCTL properties for Convex-MDPs, developed jointly with W. Li, A. L. Sangiovanni-Vincentelli and S. A. Seshia [139]. We first formally define the model-checking problem for Convex-MDPs. We then prove results about its theoretical complexity and give details about the proposed model-checking algorithm. Finally, we experimentally evaluate the runtime properties of the model-checking algorithm on case studies.

In **Chapter 5**, we present results about the quantitative analysis of the performance of a car driver. We first formally introduce the analyzed problem and motivate its relevance. We then survey related work in the field of modeling of the human behavior. Finally, we present the proposed Convex-MC model of the car driver and report results about the quantitative analysis of the behavior of individual drivers performing a complex maneuver. This work has been developed in collaboration with D. Sadigh, K. Driggs-Campbell, W. Li, V. Shia, R. Bajcsy, A. L. Sangiovanni-Vincentelli, S. S. Sastry, and S. A. Seshia [147]. In particular, the idea of using a Convex-MC model to represent the performance of a car driver is due to D. Sadigh and K. Driggs-Campbell.

In **Chapter 6**, we present the main theoretical results about the synthesis of optimal control strategies for Convex-MDPs constrained to specifications expressed in PCTL, developed jointly with A. L. Sangiovanni-Vincentelli and S. A. Seshia [141]. We first formally define the strategy-synthesis problem. We then prove the result about the theoretical complexity of the problem and give details about the proposed algorithm for the synthesis of Markov deterministic control strategies.

In **Chapter 7**, we present the results about the synthesis of control strategies to regulate the pricing and purchasing of energy for a for-profit energy aggregator whose portfolio of energy supplies includes renewable sources. We first introduce the analyzed problem and motivate its rel-

evance. We then survey related work in the field of energy dispatch and pricing in smart-grids with renewables. Finally, we present the proposed Ellipsoidal-MDP model of the analyzed decision-making scenario and report results about the system performance when regulated according to the synthesized control strategies.

Lastly, in **Chapter 8**, we draw a few final conclusions about the presented work and discuss future research directions both from a theoretical and from an application perspective.

1.5 Related Publications

The material presented in this dissertation is an extended version of the results reported in the following publications.

- [139] A. Puggelli, W. Li, A. L. Sangiovanni-Vincentelli, and S. A. Seshia. *Polynomial-Time Verification of PCTL Properties of MDPs with Convex Uncertainties*. In: Computer Aided Verification (CAV). Ed. by N. Sharygina and H. Veith. Vol. 8044. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2013, pp. 527-542. July 2013.

- [141] A. Puggelli, A. L. Sangiovanni-Vincentelli, and S. A. Seshia. *Robust Strategy Synthesis for Probabilistic Systems Applied to Risk-Limiting Renewable-Energy Pricing*. In: Proceedings of the ACM/IEEE International Conference on Embedded Software (EMSOFT). October 2014.

- [147] D. Sadigh, K. Driggs-Campbell, A. Puggelli, W. Li, V. Shia, R. Bajcsy, A. L. Sangiovanni-Vincentelli, S. S. Sastry, and S. A. Seshia. *"Data-Driven Probabilistic Modeling and Verification of Human Driver Behavior"*. In: Formal Verification and Modeling in Human-Machine Systems. AAAI Spring Symposium Series. March 2014.

Chapter 2

A Framework to Model Probabilistic Systems

We formally introduce the framework used in this work to model the behavior and express properties of stochastic systems. The chapter is divided into two main sections. In the first section, after reporting some basic definitions of concepts from convex theory, we introduce the model of Convex-MDPs, i.e., Markov Decision Processes whose state-transition probabilities are only known to lie in convex uncertainty sets. We then provide details about the mathematical representation of the four convex models of uncertainty commonly used in statistics that we analyzed in this work. We end the section by describing the procedures to resolve the non-determinism and the uncertainty in a Convex-MDP. In the second section, we introduce the Probabilistic Computation Tree Logic (PCTL), i.e., the formal logic that we use to express properties of Convex-MDPs. After describing the logic syntax, we formally define the problems of model checking and optimal control for Convex-MDPs to underline the differences in the semantics of PCTL between the verification and the control problem. We conclude the section by defining the concepts of soundness and completeness of algorithms for the model checking and optimal control of Convex-MDPs.

2.1 Convex Markov Decision Processes (Convex-MDPs)

Markov Decision Processes (MDPs) have been widely used to model systems that exhibit a stochastic behavior, with applications ranging from robot path-planning and sensor-network packet scheduling to supply-chain management and financial-portfolio optimization [142]. We chose this modeling formalism due to its promise of having a large impact in the design and verification of real-world systems. Our work builds upon these previous results and develops novel techniques for the formal verification and optimal control of Convex-MDPs, i.e., Markov Decision Processes whose state transitions are only known to lie in convex uncertainty sets. In order to have the instruments

to mathematically prove the results described in the rest of the dissertation, this section formally introduces the model of Convex-MDPs.

2.1.1 Preliminary Definitions

We begin by reporting a few definitions from statistics and convex theory, with the goal of keeping the presented material as self-contained as possible. In particular, we give the definitions of random variable, probability distribution, expected value, convex set, convex function, and jointly-convex function, which will be used in the rest of the dissertation.

Definition 2.1. ***Random Variable.*** *A random variable is a function $X : E \to \mathbb{R}$ which associates to each event e in the sample space E a real number. We distinguish between* discrete *and* continuous *random variables depending on whether the set E contains a finite or countably infinite (discrete) or uncountably infinite (continuous) number of elements, respectively.*

Definition 2.2. ***Probability Distribution.*** *The Probability Distribution (PD) for a discrete random variable X defined over a finite set E of cardinality n is a vector $\boldsymbol{\mu} \in \mathbb{R}^n$ satisfying:*

$$\begin{aligned} \mathbf{0} \leq \boldsymbol{\mu} \leq \mathbf{1}, \quad \textit{and} \\ \mathbf{1}^T \boldsymbol{\mu} = 1. \end{aligned}$$

where we use $\mathbf{1}^T \boldsymbol{\mu} = \sum_{i=0}^{n-1} \boldsymbol{\mu}[i]$. The element $\mu_i = \boldsymbol{\mu}[i]$ represents the probability of realization of the event e_i. We call $Dist(E)$ the set of distributions over E.

Intuitively, we interpret a PD $\boldsymbol{\mu}$ as a point in the positive orthant of $\mathbb{R}^n$, the i^{th} coordinate of which is the probability of realization of the event e_i. Further, we require the sum of the elements of $\boldsymbol{\mu}$ to be equal to 1. A graphical representation of a PD in $\mathbb{R}^2$ is shown in Figure 2.1(a).

Definition 2.3. ***Expected Value.*** *The expected value of a discrete random variable X defined over a finite set E of cardinality n with probability distribution $\boldsymbol{\mu}$ is the sum of the values of $X(e_i), \forall e_i \in E$, each weighted by the probability μ_i of event e_i to happen. Formally:*

$$\mathbb{E}[X] = \sum_{i=1}^{n} \mu_i \cdot X(e_i)$$

Intuitively, the expected value $\mathbb{E}[X]$ measures the value of X that is to be expected when performing a random experiment on the sample space E.

Definition 2.4. ***Convex Set.*** *A set C is convex if the line segment between any two points in C lies entirely in C, i.e., if for any $c_i, c_j \in C$ and any α with $0 \leq \alpha \leq 1$, the following holds [30]:*

$$\alpha c_i + (1 - \alpha) c_j \in C$$

A graphical representation of a convex set in $\mathbb{R}^2$ is shown in Figure 2.1(b).

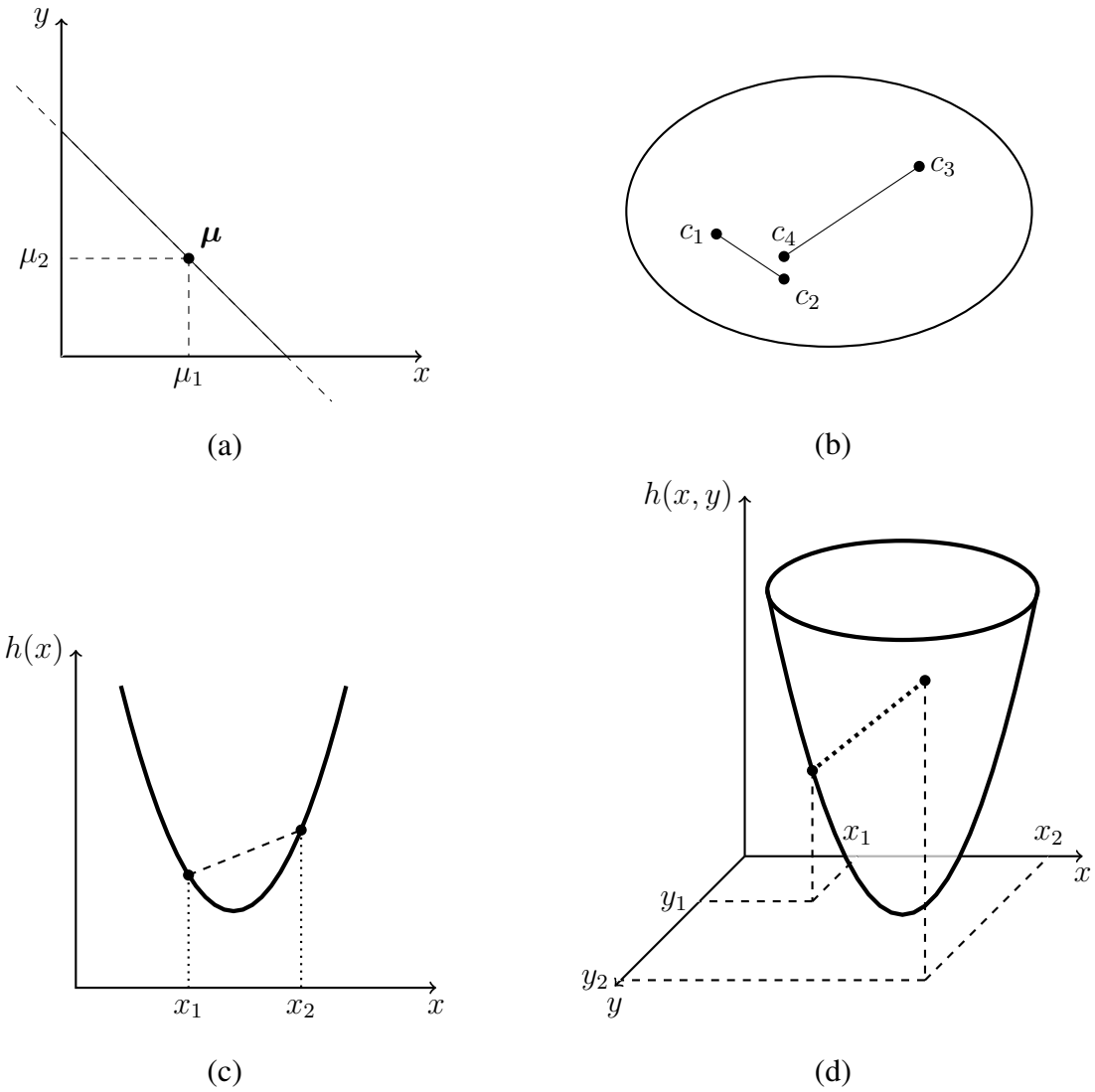


Figure 2.1: (a) A probability distribution in $\mathbb{R}^2$ can be interpreted as a point $\boldsymbol{\mu}$ lying on the line $\mu_1 + \mu_2 = 1$. (b) Example of convex set. Each line segment connecting two points in the set lies entirely in the set. (c) Example of convex function. Each line segment connecting two points belonging to the function lies entirely above the function. (d) Example of a function jointly-convex in x and y, i.e., a convex function in which variables x and y have been explicitly isolated.

Definition 2.5. ***Convex Function.*** *A function $h : \mathbb{R}^N \rightarrow \mathbb{R}$ is convex if its domain D is a convex set, and for all $\mathbf{x_1}, \mathbf{x_2} \in D$ and α with $0 \leq \alpha \leq 1$, the following inequality holds [30]:*

$$h\left(\alpha \mathbf{x_1} + (1-\alpha)\mathbf{x_2}\right) \leq \alpha h(\mathbf{x_1}) + (1-\alpha) h(\mathbf{x_2})$$

Intuitively, this definition states that function h is convex if and only if any line segment that connects two points belonging to h lies entirely above the function itself. A graphical representa-

tion of a convex function is shown in Figure 2.1(c).

Definition 2.6. ***Jointly-Convex Function.*** *A function $h : \mathbb{R}^N \times \mathbb{R}^M \to \mathbb{R}$ is jointly-convex in $\mathbf{x} \in \mathbb{R}^N$ and $\mathbf{y} \in \mathbb{R}^M$ if its domain $D = D_x \times D_y$ is a convex set, and for all $\mathbf{x_1}, \mathbf{x_2} \in D_x$, $\mathbf{y_1}, \mathbf{y_2} \in D_y$ and α with $0 \leq \alpha \leq 1$, the following inequality holds:*

$$h\left(\alpha\mathbf{x_1} + (1-\alpha)\mathbf{x_2}, \alpha\mathbf{y_1} + (1-\alpha)\mathbf{y_2}\right) \leq \alpha h(\mathbf{x_1}, \mathbf{y_1}) + (1-\alpha)h(\mathbf{x_2}, \mathbf{y_2})$$

Intuitively, this definition extends the previous result about convex functions to functions in which two (each possibly vectorial) variables have been explicitly isolated. A graphical representation of a function jointly convex in variables x and y is shown in Figure 2.1(d).

2.1.2 The Modeling Formalism

We now formally define the Convex-MDP model. The definition extends the one for MDPs by introducing the concept of uncertainty set of probability distributions. This material builds upon work by Nilim and El Ghaoui [125], and it represents a novel contribution of this dissertation.

Definition 2.7. ***Convex-MDP.*** *A Convex-MDP is a tuple $\mathcal{M}_\mathcal{C} = (S, S_0, A, \Omega, \mathcal{F}, \mathcal{A}, \mathcal{X}, L)$, where:*

S	*is a finite set of states with cardinality $N = \lvert S \rvert$;*
S_0	*is the set of initial states;*
A	*is a finite set of actions with cardinality $M = \lvert A \rvert$;*
Ω	*is a finite set of atomic propositions;*
$\mathcal{F}$	*is a finite set of convex sets of transition PDs;*
$\mathcal{A} : S \to 2^A$	*is a function that maps each state to the set of actions available at that state;*
$\mathcal{X} = S \times A \to \mathcal{F}$	*is a function that associates to state s and action a the corresponding convex set $\mathcal{F}_s^a \in \mathcal{F}$ of transition PDs, and*
$L : S \to 2^\Omega$	*is a labeling function.*

The set $\mathcal{F}_s^a = Dist_s^a(S)$ represents the uncertainty in defining a transition distribution for $\mathcal{M}_\mathcal{C}$ given state s and action a. We call $\mathbf{f}_s^a \in \mathcal{F}_s^a$ an *observation* of this uncertainty. A graphical example of a convex uncertainty set of transition distributions is shown in Figure 2.2(a). Also, $\mathbf{f}_s^a \in \mathbb{R}^N$ and we can collect the vectors $\mathbf{f}_s^a, \forall s \in S$ into an observed transition matrix $F^a \in \mathbb{R}^{N \times N}$. Abusing terminology, we call $\mathcal{F}^a$ the uncertainty set of the transition matrices, and $F^a \in \mathcal{F}^a$. $\mathcal{F}_s^a$ is interpreted as the row of $\mathcal{F}^a$ corresponding to state s. Finally, $f_{s_i s_j}^a = \mathbf{f}_{s_i}^a[j]$ is the observed probability of transitioning from s_i to s_j when action a is selected.

In general, the data-type of $a \in \mathcal{A}(s_i)$ can be different from the one of $b \in \mathcal{A}(s_j)$, if $s_i \neq s_j$. In fact, the data-type of $a \in \mathcal{A}(s_i)$ does not play any role in the process of selecting an action. Each action can just be considered as a label attached to the corresponding uncertainty set of transition PDs, chosen by the designer such that it is easier to associate a physical meaning to it.

Remark 2.1. *Although we consider Convex-MDPs as the underlying modeling formalism, the proposed techniques can be trivially extended also to Convex Markov Chains (Convex-MCs), which can be seen as Convex-MDPs with a single action, i.e., $M = 1$.*

A transition between state s to state s' in a Convex-MDP occurs in three steps as shown in Figure 2.2(b). First, an action $a \in \mathcal{A}(s)$ is chosen. The selection of a is nondeterministic. Secondly, an observed PD $\mathbf{f}_s^a \in \mathcal{F}_s^a$ is chosen. The selection of $\mathbf{f}_s^a$ models uncertainty in the transition. Lastly, a successor state s' is chosen randomly, according to the transition PD $\mathbf{f}_s^a$.

A path π in $\mathcal{M}_\mathcal{C}$ is a finite or infinite sequence of states of the form

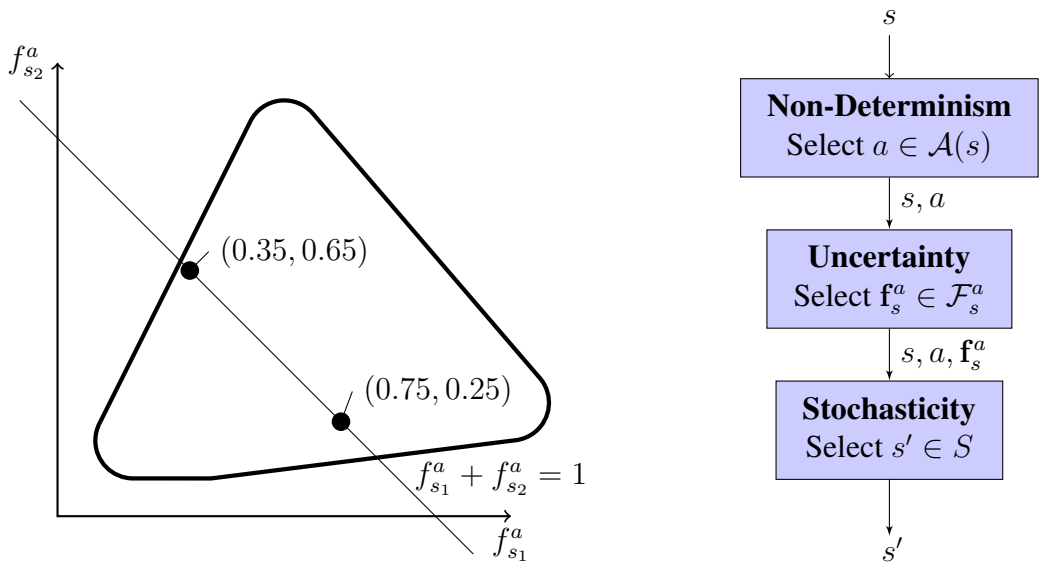
$$s_0 \xrightarrow{f_{s_0 s_1}^{a_0}} s_1 \xrightarrow{f_{s_1 s_2}^{a_1}} s_2 \xrightarrow{f_{s_2 s_3}^{a_2}} \cdots,$$

where $s_i \in S$, $a_i \in \mathcal{A}(s_i)$ and $f_{s_i,s_{i+1}}^{a_i} > 0,\ \forall i \geq 0$. We indicate with Π_{fin} (Π_{inf}) the set of all finite (infinite) paths of $\mathcal{M}_\mathcal{C}$. $\pi_s[i]$ ($\pi_a[i]$) is the i^{th} state (selected action) along the path and, for finite paths, $last(\pi)$ is the last state visited in $\pi \in \Pi_{fin}$. $\Pi_s = \{\pi \mid \pi[0] = s\}$ is the set of paths starting in state s.

A Convex-MDP $\mathcal{M}_\mathcal{C}$ has *finite-horizon* if the number of decision epochs of the model is finite. In other words, the set of infinite paths is empty, i.e., $\Pi_{inf} = \emptyset$. $\mathcal{M}_\mathcal{C}$ is *infinite-horizon* otherwise.

Definition 2.8. ***Absorbing State.*** *State $s \in S$ is called absorbing if for any action $a \in \mathcal{A}(s)$, the Convex-MDP never leaves s after entering it. Formally, $s \in S$ is absorbing if $f_{ss}^a = 1, \forall \mathbf{f}_s^a \in \mathcal{F}_s^a, \forall a \in \mathcal{A}(s)$.*

Size of a Convex-MDP. We determine the size $\mathcal{R}$ of the Convex-MDP $\mathcal{M}_\mathcal{C}$ as follows. $\mathcal{M}_\mathcal{C}$ has N states, $O(M)$ actions per state and $O(N^2)$ transitions for each action. Let D_s^a denote the



(a) (b)

Figure 2.2: (a) Example of a convex uncertainty set $\mathcal{F}_s^a$ in the two-dimensional space $\mathbb{R}^2$. Any point $\mathbf{f}_s^a$ of the plane within the thick-line shape and lying on the probability simplex is a valid PD for the Convex-MDP. (b) The three steps that occur during a transition between state $s \in S$ and $s' \in S$ in a Convex-MDP.

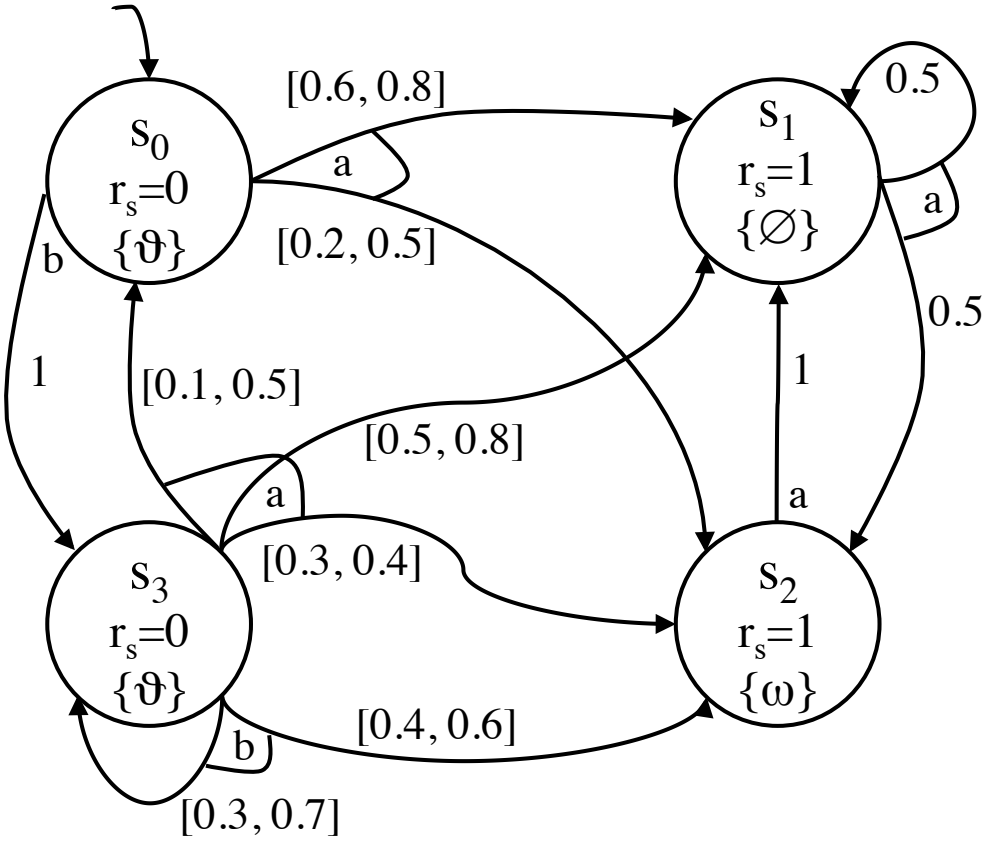


Figure 2.3: Example of a Convex-MDP $\mathcal{M}_\mathcal{C}$. In particular, $S = \{s_0 \cdots s_3\}$, $S_0 = \{s_0\}$, $A = \{a, b\}$, $\Omega = \{\omega, \vartheta\}$, $\mathcal{A} : \{s_1, s_2\} \rightarrow \{a\}$;$\{s_0, s_3\} \rightarrow \{a, b\}$, $L : \{s_0, s_3\} \rightarrow \vartheta$;$\{s_2\} \rightarrow \omega$. Uncertainty in transition probabilities is captured in the intervals shown next to each transition. For example, $\mathcal{F}^a_{s_0} = \{\mathbf{f}^a_{s_0} \in \mathbb{R}^N \mid [0, 0.6, 0.2, 0] \leq \mathbf{f}^a_{s_0} \leq [0, 0.8, 0.5, 0], \sum_{s' \in S} f^a_{ss'} = 1\}$. The reward structure r associated to $\mathcal{M}_\mathcal{C}$ is as follows: $r_s : \{s_0, s_3\} \rightarrow 0$;$\{s_1, s_2\} \rightarrow 1$, $r_a : \{(s_0, b), (s_2, a)\} \rightarrow 0$;$\{(s_0, a), (s_3, a), (s_1, a)\} \rightarrow 1$;$\{(s_3, b)\} \rightarrow 2$. The values of r_a are not shown in the figure to avoid clutter.

number of constraints required to express the uncertainty set $\mathcal{F}^a_s$ (e.g., $D^a_s = O(2N)$ for the interval model, to express the upper and lower bounds of the transition probabilities from state s to all states $s' \in S$), and let D the maximum number of such constraints across all states and actions of $\mathcal{M}_\mathcal{C}$, formally:

$$D = \max_{s \in S, a \in A} D^a_s$$

The overall size $\mathcal{R}$ of $\mathcal{M}_\mathcal{C}$ is thus:

$$\mathcal{R} = O(N^2 M + NMD)$$

2.1.2.1 Rewards

Rewards allow modeling additional quantitative measures of a Convex-MDP, e.g., a profit associated to reaching a specific state. We associate rewards to states and to actions available in each state.

Definition 2.9. ***Reward Structure.*** *A reward structure for a Convex-MDP $\mathcal{M}_{\mathcal{C}}$ is a tuple $r = (r_s, r_a)$ comprising a state reward function $r_s : S \to \mathbb{R}_{\geq 0}$ and an action reward function $r_a : S \times A \to \mathbb{R}_{\geq 0}$.*

We will only consider Convex-MDPs with positive rewards. Nevertheless, we note that it is possible to apply the algorithms proposed in this dissertation also when interpreting rewards as quantities to minimize (e.g., costs). An extension to other classes of Convex-MDPs, e.g., with both positive and negative rewards, is possible in some specific cases, but not considered in this dissertation. For more details, the interested reader is referred to the work by Puterman [142].

Definition 2.10. ***Path Reward.*** *Given a (possibly infinite) path π with horizon $k \in \mathbb{N} \cup +\infty$, the path reward for π is:*

$$rew_r(\pi, k) = \sum_{i=0}^{k} \left(r_s(\pi_s[i]) + r_a(\pi_s[i], \pi_a[i])\right)$$

We will only consider Convex-MDPs such that $rew_r(\pi, k)$ exists and it is finite $\forall \pi \in \Pi_{fin} \cup \Pi_{inf}$. These include finite and infinite-horizon Convex-MDPs ($k \in \mathbb{N} \cup \{+\infty\}$) with zero-reward absorbing states. If the Convex-MDP contains states $s \in S$ such that the path reward $rew_r(\pi, k) = +\infty$ for some path $\pi \in \Pi_s$, we will preprocess the Convex-MDP underlying graph to disconnect those states, and then compute the path reward for the remaining states on the new graph.

Figure 2.3 shows a simple Convex-MDP $\mathcal{M}_{\mathcal{C}}$, where intervals have been used to express uncertainty in the transitions.

2.1.2.2 Modeling Assumptions

In the previous section, we introduced the most general definition of Convex-MDPs. In order to derive provably-correct algorithms for the verification and control of these models, we need to partially restrict the set of allowable behaviors that can be expressed using this formalism. In this section, we list the assumptions under which we developed the algorithms presented in the rest of the dissertation. In general, only models that satisfy all these assumptions can be verified and optimally controlled using the algorithms presented in Chapter 4 and Chapter 6.

Assumption 2.1. ***Rectangular Uncertainty.*** *$\mathcal{F}^a$ can be factored as the Cartesian product of its rows, i.e., its rows are uncorrelated. Formally:*

$$\mathcal{F}^a = \mathcal{F}^a_{s_0} \times \cdots \times \mathcal{F}^a_{s_{N-1}}, \quad \forall a \in A$$

Nilim and El Ghaoui refer to this assumption as rectangular uncertainty *[125].*

The assumption of rectangular uncertainty guarantees that each row of the uncertain transition matrix $\mathcal{F}^a$ can be treated separately from the others. This assumption thus largely simplifies the problem of selecting the observed state-transition probability distribution $\mathbf{f}_s^a \in \mathcal{F}_s^a$, because it allows the selection of one such distribution at a time without considering the others. Since the tasks of model checking and optimally controlling properties of Convex-MDPs indeed involve determining the optimal state-transition probability distributions, as explained in in Chapter 4 and Chapter 6, this assumption also simplifies the development of algorithms for such problems.

Assumption 2.2. ***Transition Persistency.*** *If the probability of a transition is zero (non-zero) for at least one PD in the uncertainty set, then it is zero (non-zero) for all PDs. Formally:*

$$\exists \mathbf{f}_s^a \in \mathcal{F}_s^a : f_{ss'}^a = (\neq)0 \implies \forall \mathbf{f}_s^a \in \mathcal{F}_s^a : f_{ss'}^a = (\neq)0$$

The assumption guarantees the correctness of the preprocessing verification routines used later in this work, which rely on the reachability of the states of the Convex-MDP underlying graph.

We note that this assumption is not particularly restrictive and it can instead be interpreted as a good modeling guideline. Only state transitions that can never happen for physical or implementation reasons should be assigned probability equal to 0, while state transitions that might happen should always be assigned some non-zero probability. If the transition is unlikely to happen the probability should just be appropriately low.

Assumption 2.3. ***Convex-MDP Semantics.*** *Convex-MDPs model nondeterministic choices made from a convex set of uncountably many choices. Each time a state is visited, a transition distribution within the set is adversarially picked, and a probabilistic step is taken accordingly.*

The same semantics is defined for IMDPs by Sen et al. [150].

2.1.3 Models of Uncertainty

In this section, we formally introduce the convex models of uncertainty analyzed in this dissertation. We note that, in general, our results are not valid for arbitrary convex uncertainty models. To explain the condition that a model of uncertainty needs to satisfy in order to be analyzable using the proposed algorithms, we report the optimization problems that will be developed in details in Section 4.2.3 and Section 6.5.2. To model check and optimally control properties of Convex-MDPs, we will need to solve optimization problems in the form:

$$\text{Primal: } \nu^*(\mathbf{x}) = \max_{\mathbf{f}_s^a \in \mathcal{F}_s^a} \mathbf{x}^T \mathbf{f}_s^a \qquad \Rightarrow \qquad \text{Dual: } d^*(\mathbf{x}) = \min_{\boldsymbol{\lambda}_s^a \in \mathcal{D}_s^a} g(\boldsymbol{\lambda}_s^a, \mathbf{x}) \tag{2.1}$$

for which we give both the primal and dual formulation, both parametrized by vector $\mathbf{x}$. In Problem (2.1) the primal variable is $\mathbf{f}_s^a \in \mathcal{F}_s^a$ and the dual variable is $\boldsymbol{\lambda}_s^a$.

We are now ready to state the condition that a model of uncertainty needs to satisfy to be analyzable using the proposed algorithms:

Assumption 2.4. ***Joint-Convexity.*** *Given a Convex-MDP $\mathcal{M}_{\mathcal{C}}$, for all convex uncertainty sets $\mathcal{F}_s^a \in \mathcal{F}$, the dual function $g(\boldsymbol{\lambda}_s^a, \mathbf{x})$ in Problem (2.1) is jointly-convex in both $\boldsymbol{\lambda}_s^a$ and $\mathbf{x}$.*

This assumption holds for all the uncertainty models analyzed by Nilim and El Ghaoui [125], which represent a wide and expressive collection of uncertainty models, commonly used in statistics. As a consequence, we believe that this assumption does not limit the expressivity of the proposed approach in practical applications.

Among the models of uncertainty presented by Nilim and El Ghaoui [125], we introduce in the rest of the section the interval, likelihood, ellipsoidal and entropy models.

2.1.3.1 Interval Model

The interval model of uncertainty has been the first one introduced in the verification literature [93]. Intuitively, closed intervals can be used to capture lower and upper bounds on the estimated transition probabilities. These bounds can be formally collected in uncertainty sets $\mathcal{F}_s^a$, as follows:

$$\mathcal{F}_s^a = \{\mathbf{f}_s^a \in \mathbb{R}^N \mid \mathbf{0} \leq \underline{\mathbf{f}}_s^a \leq \mathbf{f}_s^a \leq \overline{\mathbf{f}}_s^a \leq \mathbf{1}, \sum_{i=0}^{N-1} \mathbf{f}_s^a[i] = 1\} \tag{2.2}$$

where $\underline{\mathbf{f}}_s^a, \overline{\mathbf{f}}_s^a \in \mathbb{R}^N$ are the element-wise lower and upper bounds of $\mathbf{f}_s^a$. Intuitively, the wider the interval, the higher the uncertainty in estimating the transition probabilities. This model is suitable when the transition matrix components are individually estimated by statistical data. An example of $\mathcal{F}_s^a$ for a two-dimensional case ($N = 2$) is graphically represented in Figure 2.4(a).

We can now define:

Definition 2.11. ***Interval-MDP.*** *An Interval-MDP is a Convex-MDP $\mathcal{M}_{\mathcal{C}}$ for which every uncertainty set $\mathcal{F}_s^a \in \mathcal{F}$ is expressed using Set (2.2).*

To avoid confusion in the following, we note here that an Interval-MDP is a special case of Convex-MDP, as defined in Definition 2.11, while an IMDP is a special case of Convex-MC, as introduced by Sen et al. [150].

2.1.3.2 Likelihood Model

The likelihood model is appropriate when the transition probabilities between states are determined experimentally. The resulting empirical frequencies of transition associated to action $a \in A$ are collected in matrix H^a. Uncertainty in the transition matrices can then be described by the likelihood region [108]:

$$\mathcal{F}^a = \{F^a \in \mathbb{R}^{N \times N} \mid F^a \succeq 0, F^a \mathbf{1} = \mathbf{1}, \textstyle\sum_{s,s'} h_{ss'}^a \log(f_{ss'}^a) \geq \beta^a\}$$

where $\beta^a < \beta_{max}^a = \sum_{s,s'} h_{ss'}^a \log(h_{ss'}^a)$ represents the uncertainty level. Since the likelihood region above does not satisfy Assumption 2.1, it must be approximated by a projection onto each

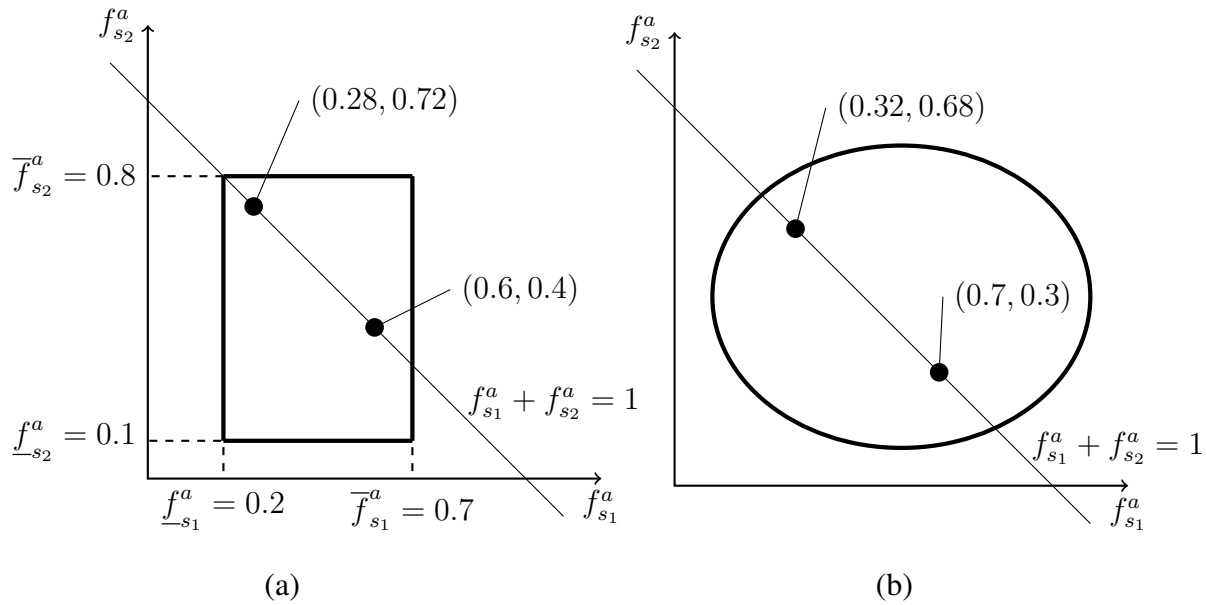


Figure 2.4: (a) Example of an interval uncertainty set $\mathcal{F}^a_s$ in the two-dimensional space $\mathbb{R}^2$. Any point $\mathbf{f}^a_s$ of the plane within the thick-line rectangle and lying on the probability simplex is a valid PD for the Interval-MDP. (b) Example of an ellipsoidal uncertainty set $\mathcal{F}^a_s$ in the two-dimensional space $\mathbb{R}^2$. Any point $\mathbf{f}^a_s$ of the plane within the thick-line ellipsoid and lying on the probability simplex is a valid PD for the Ellipsoidal-MDP.

row of the transition matrix. We obtain:

$$\mathcal{F}^a_s = \{\mathbf{f}^a_s \in \mathbb{R}^N \mid \mathbf{f}^a_s \geq \mathbf{0}, \sum_{i=0}^{N-1} \mathbf{f}^a_s[i] = 1, \sum_{s'} h^a_{ss'} \log(f^a_{ss'}) \geq \beta^a_s\} \tag{2.3}$$

with:

$$\beta^a_s < \beta^a_{s,max} = \sum_{s'} h^a_{ss'} \log(h^a_{ss'}) \tag{2.4}$$

Intuitively, the larger (in absolute value) the parameter β^a_s, the higher the uncertainty in estimating the transition probabilities. Even with this approximation, likelihood regions are less conservative uncertainty representations than intervals, which arise from a further projection onto the row components.

We define:

Definition 2.12. ***Likelihood-MDP.*** *A Likelihood-MDP is a Convex-MDP $\mathcal{M}_\mathcal{C}$ for which every uncertainty set $\mathcal{F}^a_s \in \mathcal{F}$ is expressed using Set (2.3).*

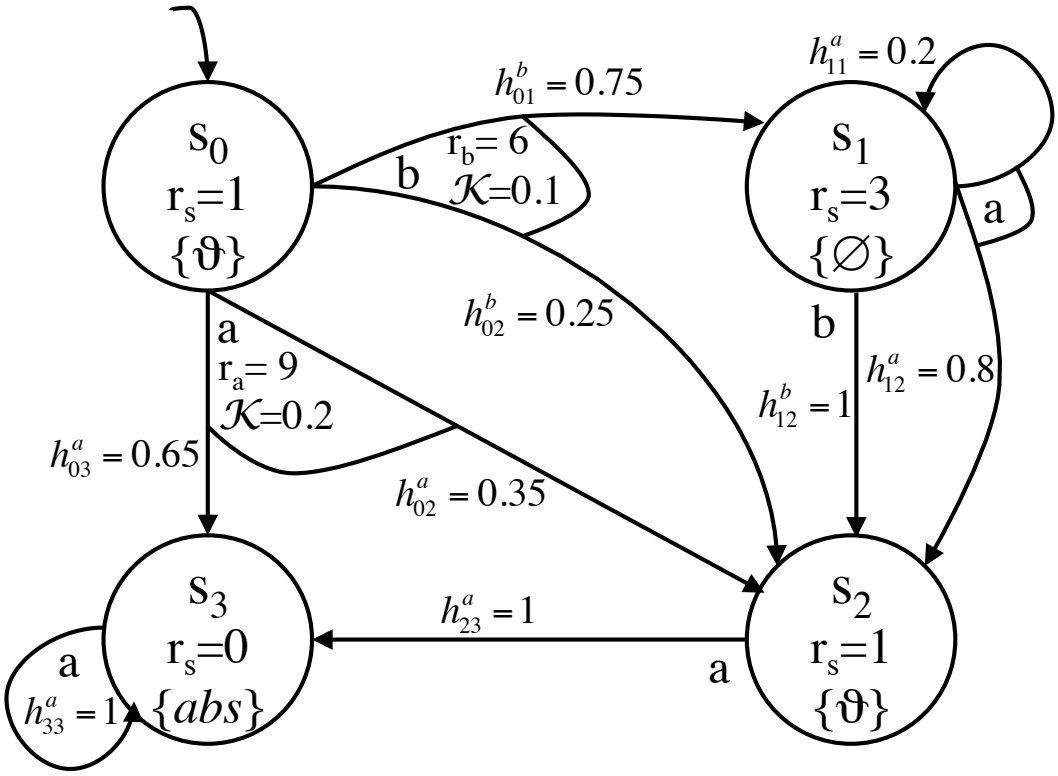


Figure 2.5: Example of an Ellipsoidal-MDP $\mathcal{M}_E$. In particular, $S = \{s_0 \cdots s_3\}$, $S_0 = \{s_0\}$, $A = \{a, b\}$, $\Omega = \{\vartheta, abs\}$, $\mathcal{A} : \{s_0, s_1\} \rightarrow \{a, b\}$ $; \{s_2, s_3\} \rightarrow \{a\}$, $L : \{s_0, s_2\} \rightarrow \vartheta$ $; \{s_3\} \rightarrow abs$. The parameters of the ellipsoids expressing the uncertainty in the estimation of transition probabilities are shown next to each transition. Finally, the reward structure r associated to $\mathcal{M}_E$ is as follows: $r_s : \{s_3\} \rightarrow 0$ $; \{s_0, s_2\} \rightarrow 1$ $; \{s_1\} \rightarrow 3$, $r_a : \{(s_1, a), (s_2, a), (s_3, a)\} \rightarrow 0$ $; \{(s_0, a)\} \rightarrow 9$ $; \{(s_0, b)\} \rightarrow 6$. Only the non-zero values of r_a are shown in the figure to avoid clutter.

2.1.3.3 Ellipsoidal Model

Ellipsoidal models can be seen as a second-order approximation of the likelihood model [125]. The transition frequencies associated to action $a \in A$ are collected in matrix H^a. Uncertainty in each row of H^a can be described by the likelihood region g^a_s:

$$g^a_s = \{\mathbf{f}^a_s \in \mathbb{R}^N | \sum_{s'} h^a_{ss'} \log(f^a_{ss'}) \geq \beta^a_s\}$$

where $\beta^a_s < \beta^a_{s,max} = \sum_{s'} h^a_{ss'} \log(h^a_{ss'})$ represents the uncertainty level. The second-order approximation of g^a_s is [125]:

$$g^a_s \approx \left\{ \mathbf{f}^a_s \in \mathbb{R}^N \mid \sum_{s'} \frac{(f^a_{ss'} - h^a_{ss'})^2}{h^a_{ss'}} \leq (\mathcal{K}^a_s)^2 \right\} \tag{2.5}$$

with:

$$\mathcal{K}_s^a = 2(\beta_{s,max}^a - \beta_s^a) \geq 0 \tag{2.6}$$

representing the uncertainty level. Intuitively, the larger the parameter $\mathcal{K}_s^a$, the higher the uncertainty in estimating the transition probabilities. We then write in conic form the approximation of g_s^a in Equation (2.5), and intersect it with the probability simplex

$$\Delta_N = \{\mathbf{f}_s^a \in \mathbb{R}^N \mid \sum_{i=0}^{N-1} \mathbf{f}_s^a[i] = 1, \mathbf{f}_s^a \geq \mathbf{0}\}$$

to obtain the uncertainty set:

$$\mathcal{F}_s^a = \{\mathbf{f}_s^a \in \mathbb{R}^N \mid \mathbf{f}_s^a \geq \mathbf{0}, \sum_{i=0}^{N-1} \mathbf{f}_s^a[i] = 1, \| E_s^a \left(\mathbf{f}_s^a - \mathbf{h}_s^a\right) \|_2 \leq 1, E_s^a \succ 0\} \tag{2.7}$$

with $E_s^a = (\mathcal{K}_s^a)^{-1} \times diag\left((h_{ss_0}^a)^{-0.5}, \cdots, (h_{ss_N}^a)^{-0.5}\right) \succ 0$ positive definite. An example of $\mathcal{F}_s^a$ for a two-dimensional case ($N = 2$) is graphically represented in Figure 2.4(b).

We can now define:

Definition 2.13. ***Ellipsoidal-MDP.*** *An Ellipsoidal-MDP is a Convex-MDP where every uncertainty set $\mathcal{F}_s^a \in \mathcal{F}$ is expressed using Set (2.7).*

In Figure 2.5, we show a simple Ellipsoidal-MDP $\mathcal{M}_E$ as an example of a Convex-MDP with non-linear convex uncertainty sets.

2.1.3.4 Entropy Model

The entropy model of uncertainty can be viewed as a variation of the likelihood model. In the likelihood setting we bound the divergence from an empirically extracted distribution, whereas in the entropy setting we bound the divergence from a reference analytical distribution q [125]. We will thus consider sets:

$$\mathcal{F}_s^a = \left\{\mathbf{f}_s^a \in \mathbb{R}^N \mid \mathbf{f}_s^a \geq \mathbf{0}, \sum_{i=0}^{N-1} \mathbf{f}_s^a[i] = 1, \sum_{s'} f_{ss'}^a \log\left(\frac{f_{ss'}^a}{q_{ss'}^a}\right) \leq \beta_s^a\right\} \tag{2.8}$$

where again β_s^a represents the uncertainty level. Intuitively, the larger (in absolute value) the parameter β_s^a, the higher the uncertainty in estimating the transition probabilities.

We define:

Definition 2.14. ***Entropy-MDP.*** *An Entropy-MDP is a Convex-MDP where every uncertainty set $\mathcal{F}_s^a \in \mathcal{F}$ is expressed using Set (2.8).*

2.1.3.5 Multiple Models of Uncertainty Within the Same Convex-MDP

We conclude this section with an important consideration. While so far we have only defined Convex-MDPs for which all uncertainty sets $\mathcal{F}_s^a \in \mathcal{F}$ were expressed with only one uncertainty set (e.g., Interval-MDPs), the algorithms proposed in this dissertation are in fact valid also for Convex-MDPs in which the uncertainty sets $\mathcal{F}_s^a \in \mathcal{F}$ are expressed using different models of uncertainty from one another. Intuitively, this is due to the fact that the presented techniques rely on results of convex duality theory, which are not specific to a single model of uncertainty, but instead apply to a broad class of convex functions, as it will be explained in Chapter 4 and Chapter 6. This feature greatly increases the modeling expressivity, since the designer can use different models to express uncertainty in the estimation of transition probabilities in different parts of the system. For example, in the study of the performance of a random back-off scheme in a wireless protocol, the interval model can be used to express uncertainty in estimating the expected back-off time, while the likelihood model, more suitable to represent transition probabilities extracted from experimental data, can be used to express uncertainty in estimating the quality of the wireless channel.

These considerations are summarized in the following remark.

Remark 2.2. *Each set $\mathcal{F}_s^a \in \mathcal{F}$ within the same Convex-MDP can be expressed with a different uncertainty model to represent different sources of uncertainty, as long as each uncertainty model satisfies Assumption 2.4.*

2.1.4 Resolution of Non-Determinism and Uncertainty

In order to analyze *quantitative* properties of the developed models, we need to generate a probability space over the set of finite and infinite paths $\Pi = \Pi_{fin} \cup \Pi_{inf}$ of the Convex-MDP $\mathcal{M}_{\mathcal{C}}$ [168]. However, a probability space can only be constructed once non-determinism and uncertainty have been resolved. Intuitively, this means that it is necessary to generate *procedures* (or *functions*) capable of selecting, at each step of the execution of the $\mathcal{M}_{\mathcal{C}}$, an action $a \in \mathcal{A}(s)$ and an observed transition probability distribution $\mathbf{f}_s^a \in \mathcal{F}_s^a$ for each state $s \in S$, so that a transition to the following state $s' \in S$ can be then executed stochastically.

In this section, we formally introduce the terminology used to refer to these procedures, following the notation developed in the verification and optimal control literature.

2.1.4.1 Adversaries and Strategies

Depending on whether we are considering the verification or the optimal control problem, we will refer to each possible resolution of non-determinism as an *adversary* or as a *strategy*, respectively. Strategies are sometimes also referred to as *policies*. While there is no difference between adversaries and strategies from a mathematical perspective, we maintain the difference in terminology introduced in the literature to reflect the difference in semantics between the two concepts, as it will be explained in detail in Section 2.2.1. To ease the reader in differentiating between adver-

saries and strategies, we will use the symbol α to refer to adversaries and the symbol σ to refer to strategies.

Both adversaries and strategies are functions that choose an action $a \in \mathcal{A}(s)$ in each state $s \in S$ of $\mathcal{M}_{\mathcal{C}}$. We differentiate among four classes of adversaries/strategies, depending on how this selection is made.

1. **History-dependent (H).** An adversary/strategy is history-dependent if the choice of a is taken based on the sequence of previously visited states.
2. **Markov (M).** An adversary/strategy is Markov if the choice of a is taken based only on the last visited state.
3. **Randomized (R).** An adversary/strategy is randomized if the choice of a is taken stochastically among the available actions in $\mathcal{A}(s)$.
4. **Deterministic (D).** An adversary/strategy is deterministic if the choice of a is taken deterministically.

We now give the formal definition of History-dependent Randomized (HR) adversaries and strategies.

Definition 2.15. ***Adversary.*** *A History-dependent Randomized (HR) adversary for $\mathcal{M}_{\mathcal{C}}$ is a function $\alpha = \Pi_{fin} \times A \rightarrow [0,1]$, which associates to each finite path $\pi \in \Pi_{fin}$ and action $a \in A$ of $\mathcal{M}_{\mathcal{C}}$ a probability for a to be chosen in the last visited state $last(\pi)$ of π, such that $\sum_{\mathcal{A}(last(\pi))} \alpha(\pi, a) = 1$, and $a \in \mathcal{A}(last(\pi))$ if $\alpha(\pi, a) > 0$.*

We call Adv the set of all adversaries α of $\mathcal{M}_{\mathcal{C}}$.

Definition 2.16. ***Strategy.*** *A History-dependent Randomized (HR) strategy for $\mathcal{M}_{\mathcal{C}}$ is a function $\sigma = \Pi_{fin} \times A \rightarrow [0,1]$, which associates to each finite path $\pi \in \Pi_{fin}$ and action $a \in A$ of $\mathcal{M}_{\mathcal{C}}$ a probability for a to be chosen in the last visited state $last(\pi)$ of π, such that $\sum_{\mathcal{A}(last(\pi))} \sigma(\pi, a) = 1$, and $a \in \mathcal{A}(last(\pi))$ if $\sigma(\pi, a) > 0$.*

We call Σ the set of all strategies σ of $\mathcal{M}_{\mathcal{C}}$.

The other classes of adversaries and strategies represent a special case of the HR ones, so their definition can be derived from Definitions 2.15 and 2.16. In particular, an adversary α (strategy σ) is *Markov* (M) if it depends only on $last(\pi)$. Also, α (σ) is *Deterministic* (D) if $\alpha(\pi, a) = 1$ ($\sigma(\pi, a) = 1$) for some $a \in \mathcal{A}(last(\pi))$.

After fixing an adversary α or a strategy σ, all the non-determinism in $\mathcal{M}_{\mathcal{C}}$ is resolved. In particular, $\mathcal{M}_{\mathcal{C}}$ is reduced to an *induced* Convex Markov Chain (Convex-MC). Since in the rest of the dissertation we will only consider deterministic adversaries (strategies), we only give the definition of Induced Convex-MC for deterministic α (σ).

Definition 2.17. ***Induced Convex-MC.*** *For a Convex-MDP $\mathcal{M}_{\mathcal{C}} = (S, S_0, A, \Omega, \mathcal{F}, \mathcal{A}, \mathcal{X}, L)$ and a deterministic adversary α (strategy σ), the induced Convex-MC is $\mathcal{M}_{\mathcal{C}}^{\alpha(\sigma)} = (\Pi_{fin}, S_0, \Omega, \mathcal{F}, \mathcal{X}', L')$ where:*

- *the set of states of* $\mathcal{M}_{\mathcal{C}}^{\alpha(\sigma)}$ *is the set* Π_{fin} *of finite paths of* $\mathcal{M}_{\mathcal{C}}$*;*
- *the sets of initial states* S_0*, atomic prepositions* Ω *and uncertainty sets* $\mathcal{F}$ *remain unchanged;*
- *For any* $\pi \in \Pi_{fin}$*, function* $\mathcal{X}$ *becomes:*

$$\mathcal{X}'(\pi, a) = \begin{cases} \mathcal{X}(last(\pi), a) & \text{if } \alpha(\pi, a) = 1 \ (\sigma(\pi, a) = 1) \\ \emptyset & \text{otherwise.} \end{cases}$$

- *the labeling function* L *becomes* $L \rightarrow L'(\pi) = L(last(\pi)),\ \forall \pi \in \Pi_{fin}$*.*

In general, the induced Convex-MC $\mathcal{M}_{\mathcal{C}}^{\alpha(\sigma)}$ has a (countably) infinite number of states. However, in the case of Markov deterministic (MD) adversaries (strategies), its state space is isomorphic to S and $\mathcal{M}_{\mathcal{C}}$ can be reduced to an $|S|$-state Convex-MC. In particular, we obtain the induced Convex-MC $\mathcal{M}_{\mathcal{C}}^{\alpha(\sigma)} = (S, S_0, \Omega, \mathcal{F}, \mathcal{X}', L')$. Intuitively, the only convex set $\mathcal{F}_s^a \in \mathcal{F}$ of transition PDs available at each state $s \in S$ is the one corresponding to the action $a \in \mathcal{A}(s)$ such that $\alpha(s, a) = 1$ ($\sigma(s, a) = 1$).

2.1.4.2 Nature

We call a *nature* each possible resolution of uncertainty in the Convex-MDP $\mathcal{M}_{\mathcal{C}}$, i.e., a nature chooses a transition probability distribution $\mathbf{f}_s^a \in \mathcal{F}_s^a$ for each state and action of $\mathcal{M}_{\mathcal{C}}$.

Analogously to adversaries and strategies, also a nature can be categorized as history-dependent (H), Markov (M), randomized (R) and deterministic (D). We report the formal definition of an HR nature in the following.

Definition 2.18. ***Nature.*** *Given action* $a \in A$*, a history-dependent randomized (HR) nature is the function* $\eta^a : \Pi_{fin} \times \mathcal{F}_{last(\pi)}^a \rightarrow [0, 1]$*, which associates to each finite path* $\pi \in \Pi_{fin}$ *and next-state transition probability distribution* $\mathbf{f}_{last(\pi)}^a \in \mathcal{F}_{last(\pi)}^a$ *on the states* S *of* $\mathcal{M}_{\mathcal{C}}$ *a probability for* $\mathbf{f}_{last(\pi)}^a$ *to be chosen in the last visited state* $last(\pi)$ *of* π*, such that* $\int_{\mathcal{F}_{last(\pi)}^a} \eta^a(\pi, \mathbf{f}_{last(\pi)}^a) = 1$*, and* $\mathbf{f}_{last(\pi)}^a \in \mathcal{F}_{last(\pi)}^a$ *if* $\eta^a(\pi, \mathbf{f}_{last(\pi)}^a) > 0$*.*

We call Nat the set of all natures η^a of $\mathcal{M}_{\mathcal{C}}$.

We derive the definitions also for the other categories of nature as follows. A nature η^a is Markov (M) if it depends only on $last(\pi)$. Further, η^a is deterministic (D) if $\eta^a(\pi, f_{last(\pi)}^a) = 1$ for some $\mathbf{f}_{last(\pi)}^a \in \mathcal{F}_{last(\pi)}^a$.

After fixing an adversary α or a strategy σ and a nature η^a, all the non-determinism and uncertainty in the Convex-MDP $\mathcal{M}_{\mathcal{C}}$ is resolved. In particular, $\mathcal{M}_{\mathcal{C}}$ is reduced to an *induced* Discrete Time Markov Chain (DTMC). In the following, we give the definition of Induced DTMC for deterministic α (σ) and deterministic η^a.

Definition 2.19. ***Induced DTMC.*** *For a Convex-MDP* $\mathcal{M}_{\mathcal{C}} = (S, S_0, A, \Omega, \mathcal{F}, \mathcal{A}, \mathcal{X}, L)$*, a deterministic adversary* α *(strategy* σ*) and a deterministic nature* η^a*, the induced DTMC is* $\mathcal{M}_{\mathcal{C}}^{\alpha(\sigma),\eta^a} = (\Pi_{fin}, S_0, \Omega, Steps, L')$ *where:*

- *the set of states of* $\mathcal{M}_{\mathcal{C}}^{\alpha(\sigma),\eta^a}$ *is the set* Π_{fin} *of finite paths of* $\mathcal{M}_{\mathcal{C}}$*;*
- *the sets of initial states* S_0 *and atomic prepositions* Ω *remain unchanged;*
- *For any* $\pi, \pi' \in \Pi_{fin}$*, we define function* $Steps : \Pi_{fin} \times \Pi_{fin} \rightarrow [0,1]$ *to compute the probability of transitioning between two states of* $\mathcal{M}_{\mathcal{C}}^{\alpha(\sigma),\eta^a}$*. Formally:*

$$Steps(\pi,\pi') = \begin{cases} \mathbf{f}_{s_i}^a[j] & \text{if } \alpha(\pi,a) = 1 \ (\sigma(\pi,a) = 1),\ \eta^a(\pi,\mathbf{f}_{s_i}^a) = 1, \\ & last(\pi) = s_i,\ last(\pi') = s_j \\ 0 & \text{otherwise.} \end{cases}$$

  By construction, $\sum_{\pi'} Steps(\pi,\pi') = 1,\ \forall \pi \in \Pi_{fin}$*;*
- *the labeling function* L *becomes* $L \rightarrow L'(\pi) = L(last(\pi)),\ \forall \pi \in \Pi_{fin}$*.*

Figure 2.6 shows (part of) one of the Induced-DTMCs from the Convex-MDP of Figure 2.3.

Again, in the case of Markov deterministic (MD) adversaries (strategies) and natures, the state space of the induced DTMC is isomorphic to S and $\mathcal{M}_{\mathcal{C}}$ can be reduced to an $|S|$-state DTMC.

There exists a one-to-one mapping between the infinite paths of the DTMC $\mathcal{M}_{\mathcal{C}}^{\alpha(\sigma),\eta^a}$ and the infinite paths of Convex-MDP $\mathcal{M}_{\mathcal{C}}$ when operating according to adversary α (strategy σ) and nature η^a. This means that the DTMC yields, for an initial state $s \in S_0$, a probability space $Prob_s^{\alpha(\sigma),\eta^a}$ over these infinite paths. In the next section, we will show how it is possible to use the probability space $Prob_s^{\alpha(\sigma),\eta^a}$ to analyze quantitative properties of the Convex-MDP $\mathcal{M}_{\mathcal{C}}$.

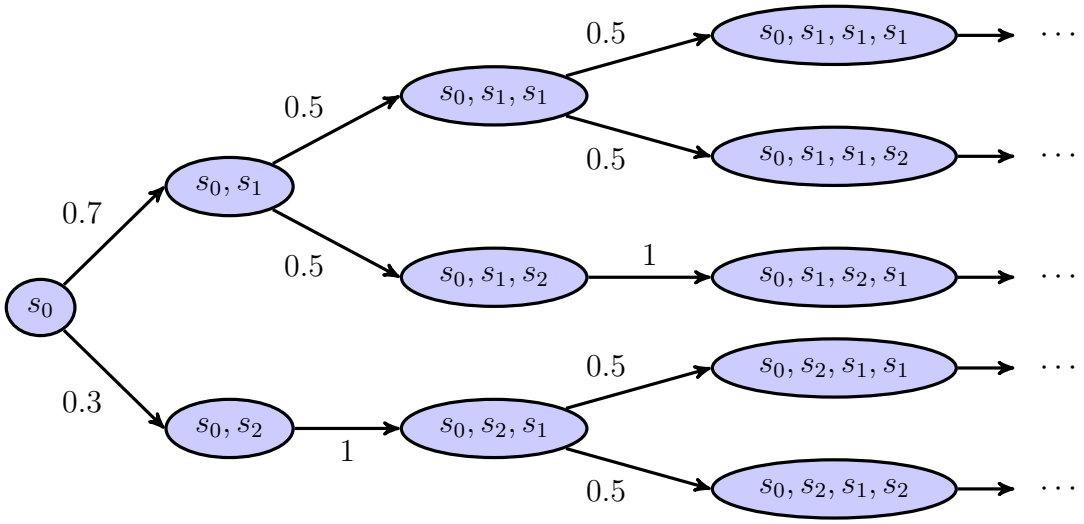


Figure 2.6: Example of one of the Induced DTMCs from the Convex-MDP in Figure 2.3. We assumed a deterministic adversary (strategy) that selects action a in state s_0 ($\alpha(\sigma)(s_0,a) = 1$), and a deterministic nature that selects the PD $\mathbf{f}_{s_0}^a = [0, 0.7, 0.3, 0]$ in s_0, when action a is selected ($\eta^a(s_0,\mathbf{f}_{s_0}^a) = 1$).

2.2 Probabilistic Computation Tree Logic (PCTL)

In order to formally verify and optimally control properties of Convex-MDPs, we need an appropriate "language" to unambiguously specify such properties. We use Probabilistic Computation Tree Logic (PCTL), a probabilistic logic derived from CTL which includes a probabilistic operator $\boldsymbol{P}$ [72] and a reward operator $\boldsymbol{R}$ [60]. The syntax of this logic is defined as follows:

$$\begin{aligned}
\phi &::= True \mid \omega \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \boldsymbol{P}_{\bowtie p}[\psi] \mid \boldsymbol{R}^{r}_{\bowtie v}[\rho] && \text{state formulas} \\
\psi &::= \boldsymbol{\mathcal{X}}\ \phi \mid \phi_1\ \boldsymbol{\mathcal{U}}^{\leq k}\ \phi_2 \mid \phi_1\ \boldsymbol{\mathcal{U}}\ \phi_2 && \text{path formulas} \\
\rho &::= \boldsymbol{\mathcal{I}}^{=k} \mid \boldsymbol{\mathcal{C}}^{\leq k} \mid \boldsymbol{\mathcal{C}}\ \phi && \text{rewards}
\end{aligned}$$

where $\omega \in \Omega$ is an atomic proposition, $\bowtie \in \{\leq, <, \geq, >\}$, $p \in [0,1]$, $v \in \mathbb{R}_{\geq 0}$, $k \in \mathbb{N}$ and r is a reward structure for a Convex-MDP as introduced in Definition 2.9.

Path formulas ψ use the temporal operators *Next* ($\boldsymbol{\mathcal{X}}$), *Bounded Until* ($\boldsymbol{\mathcal{U}}^{\leq k}$) and *Unbounded Until* ($\boldsymbol{\mathcal{U}}$). Intuitively, $\boldsymbol{\mathcal{X}}\ [\phi]$ is true if ϕ is satisfied in the next state; $\phi_1\ \boldsymbol{\mathcal{U}}^{\leq k}\ \phi_2$ is true if ϕ_2 is satisfied within k time-steps and ϕ_1 holds up until that point; and $\phi_1\ \boldsymbol{\mathcal{U}}\ \phi_2$ is true if ϕ_2 is satisfied at some point in the future and ϕ_1 holds up until then. These formulas are evaluated over paths and only allowed as parameters to the $\boldsymbol{P}_{\bowtie p}[\psi]$ operator.

Reward formulas use the temporal operators *Instantaneous Reward* ($\boldsymbol{\mathcal{I}}^{=k}$), *Bounded Cumulative Reward* ($\boldsymbol{\mathcal{C}}^{\leq k}$) and *Cumulative Reward* ($\boldsymbol{\mathcal{C}}$). Intuitively, the *Instantaneous reward* operator computes the reward of the state entered after k steps of execution of the Convex-MDP; the *Bounded Cumulative Reward* operator computes the reward accumulated during k steps of execution; the *Cumulative Reward* operator computes the reward accumulated before reaching a state satisfying ϕ. These formulas are evaluated over reward structures and only allowed as parameters to the $\boldsymbol{R}_{\bowtie v}[\rho]$ operator.

Intuitively, the $\boldsymbol{P}$ and $\boldsymbol{R}$ operators check whether the satisfaction probability of some path formula (reward) is always above or below the numerical threshold p (v). Further, we will allow the *quantitative* versions of the $\boldsymbol{P}$ and $\boldsymbol{R}$ operators, which instead simply compute the maximum/minimum probability (reward) of the formula itself. To express such operators, we will use the syntax $\boldsymbol{P}^{min}[\psi]$/$\boldsymbol{P}^{max}[\psi]$ ($\boldsymbol{R}^{min}[\rho]$/$\boldsymbol{R}^{max}[\rho]$).

Remark 2.3. *The* quantitative *version of the $\boldsymbol{P}$ and $\boldsymbol{R}$ operators cannot be nested within a PCTL formula.*

We can now determine the *size* of a PCTL formula as follows.

Size of a PCTL formula We define the size $\mathcal{Q}$ of a PCTL formula as the number of Boolean connectives and probabilistic and reward operators not containing a temporal operator with a finite time horizon (*Next*, *Unbounded Until* and *Cumulative Reward*), plus the number of probabilistic and reward operators containing a temporal operator with a finite time horizon (*Bounded Until*, *Instantaneous Reward* and *Bounded Cumulative Reward*) times the the maximum time horizon k_{max} in the formula. Formally:

$$\mathcal{Q} = \#OP_{No\ Bound} + \#OP_{Bounded} \times k_{max}$$

2.2.1 PCTL Semantics

The interpretation or *semantics* of the logic PCTL varies depending on whether we aim to model check a property of the Convex-MDP or to optimally control the model to maximize its performance. Before defining the two corresponding semantics of the logic, we (informally) define the two problems that will be analyzed in the rest of the dissertation. The formal definitions of these problems will be given in Chapter 4 and Chapter 6, respectively.

- **Formal Verification.** In the verification settings, we aim to verify whether the Convex-MDP $\mathcal{M}_{\mathcal{C}}$ satisfies a given PCTL property ϕ under all resolutions of non-determinism and uncertainty. As a consequence, we interpret adversaries and natures as playing *together* against the model, with the goal of making $\mathcal{M}_{\mathcal{C}}$ fail property ϕ. The model-checking procedure will aim to identify the worst-case adversary and nature and verify that $\mathcal{M}_{\mathcal{C}}$ still satisfies ϕ also under these worst-case conditions.

- **Optimal Control.** In the control settings, we aim to synthesize a strategy for the Convex-MDP $\mathcal{M}_{\mathcal{C}}$ such that the model performance gets optimized, while having $\mathcal{M}_{\mathcal{C}}$ satisfy a given PCTL property ϕ under all resolutions of uncertainty. From a game-theory perspective, we interpret strategies and natures as playing a game *against* one another, where strategies aim to maximize system performance while natures aim to minimize such performance to have $\mathcal{M}_{\mathcal{C}}$ fail property ϕ.

Intuitively, in both settings, we interpret nature as representing errors and inaccuracies in the modeling process or unexpected (e.g., faulty) behaviors of the system. Consequently, we aim to consider the most adversarial effect of such modeling shortcomings. In the verification settings, we further assume that the system has already been designed so that its behavior is already determined. We thus aim to check whether such a system does indeed satisfy a given property also under its worst-case behavior. In the control settings, on the other hand, the system has not been fully designed yet, and we aim to set its behavior by means of a *strategy* such that its performance gets optimized.

In this dissertation, we will not consider the problem of **Parameter Synthesis**, which is the dual of the formal verification one, i.e., strategies and natures play together to maximize system performance. This problem arises when the designer has the capability of setting the value of some parameter of the system which varies continuously, e.g., the component concentration in a chemical process, and he or she is interested in selecting the parameter value that maximizes a given system performance[1]. While also this problem is of high practical interest, the development of a scalable procedure to solve it has so far been elusive to the verification community. Further analysis on the matter is left as a promising future direction.

In the following, we define the semantics of PCTL for the verification and the control settings.

[1] In the context of the Convex-MDP modeling formalism, these parameters are captured in terms of state-transition probabilities.

2.2.1.1 PCTL Semantics for the Verification Problem

In the verification settings, we aim to verify whether the Convex-MDP $\mathcal{M}_{\mathcal{C}}$ satisfies a given PCTL property ϕ under all resolutions of non-determinism and uncertainty. This can be done by generating the induced DTMCs $\mathcal{M}_{\mathcal{C}}^{\alpha,\eta^a}$ for all adversaries $\alpha \in Adv$ and natures $\eta^a \in Nat$, and checking whether all such induced DTMCs satisfy the PCTL property ϕ (*universal quantification*).

Considering a run of $\mathcal{M}_C$ starting from state $s \in S$, we define:

$$P_s(\alpha, \eta^a)[\psi] \triangleq Prob_s^{\alpha,\eta^a}\left(\{\pi \in \Pi_s^{\alpha,\eta^a} \mid \pi \models \psi\}\right) \tag{2.9}$$

the probability of taking a path $\pi \in \Pi_s$ that satisfies ψ under adversary α and nature η^a. In Equation (2.9), $Prob_s^{\alpha,\eta^a}$ is the probability space generated by the induced DTMC $\mathcal{M}_{\mathcal{C}}^{\alpha,\eta^a}$. $P_s^{max}[\psi]$ ($P_s^{min}[\psi]$) denote the maximum (minimum) probability $P_s(\alpha, \eta^a)[\psi]$ across all adversaries $\alpha \in Adv$ and natures $\eta^a \in Nat$, and the vectors $\mathbf{P}^{max}[\psi] \in \mathbb{R}^N$ ($\mathbf{P}^{min}[\psi] \in \mathbb{R}^N$) collect these probabilities $\forall s \in S$. If α and η^a are Markov deterministic (MD) in state s, we write $P_s(a, \mathbf{f}_s^a)$, where a and $\mathbf{f}_s^a$ are the action and resolution of uncertainty that are deterministically chosen at each execution step by α and η^a.

For a Convex-MDP $\mathcal{M}_{\mathcal{C}}$, and a property ϕ, we will write $\mathcal{M}_{\mathcal{C}}, s \models_{Adv,Nat} \phi$ to denote that, when starting from a state $s \in S$, $\mathcal{M}_{\mathcal{C}}$ satisfies ϕ for any $\alpha \in Adv$ and $\eta^a \in Nat$. We will then collect all states $s \in S$ satisfying ϕ in the satisfiability set:

$$Sat(\phi) = \{s \in S \mid \mathcal{M}_{\mathcal{C}}, s \models_{Adv,Nat} \phi\} \tag{2.10}$$

To compute properties involving the reward operator $\boldsymbol{R}$, we further define three random vari-

Table 2.1: PCTL semantics for Convex-MDPs in the verification settings

s	$\models$	True	
s	$\models$	ω	iff $\omega \in L(s)$
s	$\models$	$\neg\phi$	iff $s \not\models \phi$
s	$\models$	$\phi_1 \wedge \phi_2$	iff $s \models \phi_1 \wedge s \models \phi_2$
s	$\models$	$\boldsymbol{P}_{\lhd p}[\psi]$	iff $P_s^{max}(\{\pi \in \Pi_s \mid \pi \models \psi\}) \lhd p$
s	$\models$	$\boldsymbol{P}_{\rhd p}[\psi]$	iff $P_s^{min}(\{\pi \in \Pi_s \mid \pi \models \psi\}) \rhd p$
π	$\models$	$\boldsymbol{\mathcal{X}}\ \phi$	iff $\pi[1] \models \phi$
π	$\models$	$\phi_1\ \boldsymbol{\mathcal{U}}^{\leq \boldsymbol{k}}\ \phi_2$	iff $\exists i \leq k \mid \pi[i] \models \phi_2 \wedge \forall j < i\ \pi[j] \models \phi_1$
π	$\models$	$\phi_1\ \boldsymbol{\mathcal{U}}\ \phi_2$	iff $\exists k \geq 0 \mid \pi \models \phi_1\ \boldsymbol{\mathcal{U}}^{\leq \boldsymbol{k}}\ \phi_2$
s	$\models$	$\boldsymbol{R}_{\lhd v}[\rho]$	iff $\mathbb{E}_s^{max}(\rho) \lhd v$
s	$\models$	$\boldsymbol{R}_{\rhd v}[\rho]$	iff $\mathbb{E}_s^{min}(\rho) \rhd v$

ables over the sets of paths starting from state $s \in S$, i.e., Π_s:

$$\boldsymbol{\mathcal{I}}^{=\boldsymbol{k}} : \Pi_s \to \mathbb{R}_{\geq 0} \quad \text{with} \quad \boldsymbol{\mathcal{I}}^{=\boldsymbol{k}}(\pi) \triangleq r_s(\pi(k)) \tag{2.11}$$

$$\boldsymbol{\mathcal{C}}^{\leq \boldsymbol{k}} : \Pi_s \to \mathbb{R}_{\geq 0} \quad \text{with} \quad \boldsymbol{\mathcal{C}}^{\leq \boldsymbol{k}}(\pi) \triangleq rew_r(\pi, k) \tag{2.12}$$

$$\boldsymbol{\mathcal{C}}\,\phi : \Pi_s \to \mathbb{R}_{\geq 0} \quad \text{with} \quad \boldsymbol{\mathcal{C}}(\pi) \triangleq \begin{cases} \infty & \text{if } \pi[i] \notin Sat(\phi),\ \forall i \in \mathbb{N} \\ rew_r(\pi, k^{Sat(\phi)}) & \text{otherwise} \end{cases} \tag{2.13}$$

where $rew_r(\pi, k)$ is the path reward as introduced in Definition 2.10, $Sat(\phi)$ is the set of states satisfying property ϕ, and $k^{Sat(\phi)}$ is the minimum integer number of steps in path π to reach a state in $Sat(\phi)$ starting from s, i.e., $k^{Sat(\phi)} = min\{k \in \mathbb{N} \mid \pi[k] \in Sat(\phi)\}$.

For any $\rho \in \{\boldsymbol{\mathcal{I}}^{=\boldsymbol{k}}, \boldsymbol{\mathcal{C}}^{\leq \boldsymbol{k}}, \boldsymbol{\mathcal{C}}\}$, $\mathbb{E}_s(\alpha, \eta^a)[\rho]$ denotes the *expected value* of the random variable ρ when the execution of the Convex-MDP is controlled by adversary α and nature η^a. Moreover, $\mathbb{E}_s^{max}[\rho]$ ($\mathbb{E}_s^{min}[\rho]$) denotes the maximum (minimum) expected value of ρ, $\mathbb{E}_s(\alpha, \eta^a)[\rho]$, across all adversaries $\alpha \in Adv$ and natures $\eta^a \in Nat$, and the vectors $\mathbb{E}^{max}[\rho] \in \mathbb{R}^N$ ($\mathbb{E}^{min}[\rho] \in \mathbb{R}^N$) collect the expected values of the reward $\forall s \in S$.

The semantics of the logic is reported in Table 2.1, where we write $\models$ instead of $\mathcal{M}_{\mathcal{C}} \models_{Adv,Nat}$ for simplicity.

2.2.1.2 PCTL Semantics for the Control Problem

In the control settings, we aim to synthesize a strategy σ under which the Convex-MDP $\mathcal{M}_{\mathcal{C}}$ maximizes some given performance while at the same time satisfying a PCTL specification ϕ under all resolutions of uncertainty. We can thus map the synthesis task to a constrained optimization problem, where we aim to optimize the system performance constrained to specification ϕ.

As a quantitative measure to express the system performance, we will use the *total expected reward*, defined as follows.

Table 2.2: PCTL semantics for for Convex-MDPs in the control settings

s	$\models$	True	
s	$\models$	ω	iff $\omega \in L(s)$
s	$\models$	$\neg\phi$	iff $s \not\models \phi$
s	$\models$	$\phi_1 \wedge \phi_2$	iff $s \models \phi_1 \wedge s \models \phi_2$
s	$\models$	$\boldsymbol{P}_{\lhd p}[\psi]$	iff $P_s^{\sigma,max}(\{\pi \in \Pi_s \mid \pi \models \psi\}) \lhd p$
s	$\models$	$\boldsymbol{P}_{\rhd p}[\psi]$	iff $P_s^{\sigma,min}(\{\pi \in \Pi_s \mid \pi \models \psi\}) \rhd p$
π	$\models$	$\boldsymbol{\mathcal{X}}\,\phi$	iff $\pi[1] \models \phi$
π	$\models$	$\phi_1\,\boldsymbol{\mathcal{U}}^{\leq \boldsymbol{k}}\,\phi_2$	iff $\exists i \leq k \mid \pi[i] \models \phi_2 \wedge \forall j < i\ \pi[j] \models \phi_1$
π	$\models$	$\phi_1\,\boldsymbol{\mathcal{U}}\,\phi_2$	iff $\exists k \geq 0 \mid \pi \models \phi_1\,\boldsymbol{\mathcal{U}}^{\leq \boldsymbol{k}}\,\phi_2$
s	$\models$	$\boldsymbol{R}_{\lhd v}[\rho]$	iff $\mathbb{E}_s^{\sigma,max}(\rho) \lhd v$
s	$\models$	$\boldsymbol{R}_{\rhd v}[\rho]$	iff $\mathbb{E}_s^{\sigma,min}(\rho) \rhd v$

Definition 2.20. ***Total Expected Reward.*** *Given a Convex-MDP $\mathcal{M}_{\mathcal{C}}$, the total expected reward for state $s \in S$ under strategy $\sigma \in \Sigma$ is defined as:*

$$\mathbb{W}_s^{\sigma} := \min_{\eta^a \in Nat} \mathbb{E}^{\sigma,\eta^a}(rew_r(\pi, k))) \tag{2.14}$$

where we minimize across the action range $\eta^a \in Nat$ of the adversarial nature the expected reward over all paths $\pi \in \Pi_s$ with horizon k starting from s and visited under strategy σ. In the following, we will also use the symbol $\mathbb{W}_{S_0}^{\sigma}$ to represent the sum *of the expected rewards $\mathbb{W}_s^{\sigma}$ over all the initial states $s \in S_0$.*

We will only consider Convex-MDPs such that $\mathbb{W}_s^{\sigma}$ exists and it is finite $\forall s \in S, \forall \sigma \in \Sigma$. These include finite and infinite-horizon Convex-MDPs ($k \in \mathbb{N} \cup \{+\infty\}$) with zero-reward absorbing states. If the Convex-MDP contains states $s \in S$ such that the path reward $\mathbb{W}_s^{\sigma} = +\infty$ for some path $\pi \in \Pi_s$ under strategy σ, we will preprocess the Convex-MDP underlying graph to disconnect those states, and then compute the path reward under strategy σ for the remaining states on the new graph. Further, we note that it is possible to apply the algorithms proposed in this dissertation also when interpreting rewards as quantities to minimize (e.g., costs). An extension to other classes of Convex-MDPs, e.g., with both positive and negative rewards, is possible in some specific cases, but not considered in this dissertation. For more details, the interested reader is referred to the work by Puterman [142].

Given a strategy $\sigma \in \Sigma$, we can then verify whether $\mathcal{M}_{\mathcal{C}}$ satisfies specification ϕ by generating the induced DTMCs $\mathcal{M}_{\mathcal{C}}^{\sigma,\eta^a}$ for all natures $\eta^a \in Nat$, and checking whether all such induced DTMCs satisfy the PCTL property ϕ. Analogously to the verification settings, we define:

$$P_s(\sigma, \eta^a)[\psi] \stackrel{def}{=} Prob_s^{\sigma,\eta^a}\left(\{\pi \in \Pi_s^{\sigma,\eta^a} \mid \pi \models \psi\}\right) \tag{2.15}$$

the probability of taking a path $\pi \in \Pi_s$ that satisfies ψ under strategy σ and nature η^a. In Equation (2.15), $Prob_s^{\sigma,\eta^a}$ is the probability space generated by the induced DTMC $\mathcal{M}_{\mathcal{C}}^{\sigma,\eta^a}$. $P_s^{\sigma,max}[\psi]$ ($P_s^{\sigma,min}[\psi]$) denote the maximum (minimum) probability $P_s(\sigma, \eta^a)[\psi]$ across all natures $\eta^a \in Nat$, for a fixed strategy σ, and the vectors $\mathbf{P}^{\sigma,max}[\psi] \in \mathbb{R}^N$ ($\mathbf{P}^{\sigma,min}[\psi] \in \mathbb{R}^N$) collect these probabilities $\forall s \in S$.

For a Convex-MDP $\mathcal{M}_{\mathcal{C}}$, strategy σ, and property ϕ, we will write $\mathcal{M}_{\mathcal{C}}, \sigma, s \models_{Nat} \phi$ to denote that, when starting from a state $s \in S$, and operating under σ, $\mathcal{M}_{\mathcal{C}}$ satisfies property ϕ for any $\eta^a \in Nat$. We will then collect all states $s \in S$ satisfying ϕ under strategy σ in the satisfiability set:

$$Sat^{\sigma}(\phi) = \{s \in S \mid \mathcal{M}_{\mathcal{C}}, \sigma, s \models_{Nat} \phi\} \tag{2.16}$$

The three temporal operators $\boldsymbol{\mathcal{I}^{=k}}, \boldsymbol{\mathcal{C}^{\leq k}}, \boldsymbol{\mathcal{C}}$ appearing in reward properties are defined similarly to the ones introduced in Equation (2.11)-(2.13) for the PCTL semantics for the verification problem, and will not be redefined. For any $\rho \in \{\boldsymbol{\mathcal{I}^{=k}}, \boldsymbol{\mathcal{C}^{\leq k}}, \boldsymbol{\mathcal{C}}\}$, $\mathbb{E}_s^{\sigma,max}[\rho]$ ($\mathbb{E}_s^{\sigma,min}[\rho]$) denotes the maximum (minimum) *expected value* of the random variable ρ under strategy σ across all natures $\eta^a \in Nat$.

The semantics of the logic is reported in Table 2.2, where we write $s \models$ instead of $\mathcal{M}_{\mathcal{C}}, \sigma, s \models_{Nat}$ for simplicity.

We now collect all strategies $\sigma \in \Sigma$ under which all the initial states $s \in S_0$ of $\mathcal{M}_{\mathcal{C}}$ satisfy the PCTL specification ϕ in the set:

$$\Sigma_\phi = \{\sigma \in \Sigma \mid S_0 \subseteq Sat^\sigma(\phi)\} \tag{2.17}$$

The goal of the strategy-synthesis algorithm will be to determine the strategy $\sigma^* \in \Sigma_\phi$ that maximizes $\mathbb{W}^\sigma_{S_0}$, i.e., $\forall \sigma \in \Sigma_\phi, \mathbb{W}^\sigma_{S_0} \leq \mathbb{W}^{\sigma^*}_{S_0}$.

2.2.2 Expressing System Properties in PCTL

The relevance of a formal logic lies in its capability of expressing a wide variety of properties of the system under analysis, i.e., in its *expressivity*. In this section, we show how to express system properties using PCTL. In particular, we consider properties expressed in the *qualitative* logic CTL*, a logic widely used in the verification community, and re-write these properties using the PCTL syntax to give guidelines to the readers in how to translate commonly-used properties into PCTL. The material in this section is partially taken from work by Hansson and Jonsson [72].

Remark 2.4. *The logic PCTL allows to compute the* quantitative *version of all the properties described in this section, i.e., the satisfaction probability does not need to be either 0 or 1, but it is allowed to be any real number p in the interval $0 \leq p \leq 1$.*

We start by defining the shortcut temporal operator *Weak Until* $\boldsymbol{\mathcal{U}_W}$:

$$\boldsymbol{P}_{\geq p}[\phi_1 \, \boldsymbol{\mathcal{U}_W} \, \phi_2] \equiv \neg \boldsymbol{P}_{>1-p}[\neg\phi_2 \, \boldsymbol{\mathcal{U}} \, \neg(\phi_1 \vee \phi_2)]$$
$$\boldsymbol{P}_{> p}[\phi_1 \, \boldsymbol{\mathcal{U}_W} \, \phi_2] \equiv \neg \boldsymbol{P}_{\geq 1-p}[\neg\phi_2 \, \boldsymbol{\mathcal{U}} \, \neg(\phi_1 \vee \phi_2)]$$

Intuitively, the *Weak Until* operator is a variance of the (strong) *Unbounded Until* operator ($\boldsymbol{\mathcal{U}}$), which does not require ϕ_2 to be ever satisfied, as long as ϕ_1 is always satisfied starting from the initial states $s \in S_0$.

We report in Table 2.3 a list of commonly used CTL* properties rewritten using the PCTL syntax. In other words, the properties in the two columns of the table are *equivalent* ($\equiv$), i.e., they have the same satisfying states. In Table 2.3, $0 \leq p \leq 1$ is the satisfaction probability and $k \in \mathbb{N}$ is a natural number. Intuitively, the formula $\mathbf{A}\ \psi$ means that all execution paths satisfy property ψ;

Table 2.3: Conversion table between equivalent CTL* and PCTL properties

CTL*	PCTL
$\mathbf{A}\ \psi$	$\boldsymbol{P}_{\geq 1}[\psi]$
$\mathbf{E}\ \psi$	$\boldsymbol{P}_{> 0}[\psi]$
$\mathbf{A}\ \boldsymbol{G}\ \phi$	$\boldsymbol{P}_{\geq 1}[\phi \, \boldsymbol{\mathcal{U}_W} \, \neg True]$
$\mathbf{A}\ \boldsymbol{F}\ \phi$	$\boldsymbol{P}_{\geq 1}[True \, \boldsymbol{\mathcal{U}} \, \phi]$
$\mathbf{E}\ \boldsymbol{G}\ \phi$	$\boldsymbol{P}_{> 0}[\phi \, \boldsymbol{\mathcal{U}_W} \, \neg True]$
$\mathbf{E}\ \boldsymbol{F}\ \phi$	$\boldsymbol{P}_{> 0}[True \, \boldsymbol{\mathcal{U}} \, \phi]$

$\mathbf{E}\,\psi$ that there exists a path satisfying ψ with non-zero probability; $\mathbf{A}\,\boldsymbol{G}\,\phi$ means that ϕ is always satisfied in all states that can be reached with non-zero probability; $\mathbf{A}\,\boldsymbol{F}\,\phi$ means that a state where ϕ is satisfied will be eventually reached with probability equal to 1; $\mathbf{E}\,\boldsymbol{G}\,\phi$ means that there is a non-zero probability for globally satisfied, and; $\mathbf{E}\,\boldsymbol{F}\,\phi$ means that there exists a state where ϕ holds, which can be reached with non-zero probability.

We can now define the two following shortcut operators:

$$\begin{aligned}
\boldsymbol{P}_{\geq p}[\boldsymbol{F}\ \phi] &\equiv \boldsymbol{P}_{\geq p}\,[True\,\boldsymbol{\mathcal{U}}\ \phi] \\
\boldsymbol{P}_{\geq p}[\boldsymbol{G}\ \phi] &\equiv \boldsymbol{P}_{\geq p}\,[\phi\,\boldsymbol{\mathcal{U}_W}\ \neg True] \\
\boldsymbol{P}_{\geq p}[\boldsymbol{G}\ \phi] &\equiv \boldsymbol{P}_{<1-p}\,[\boldsymbol{F}\ \neg\phi]
\end{aligned}$$

Intuitively, $\boldsymbol{P}_{\geq p}[\boldsymbol{F}\ \phi]$ means that ϕ eventually holds with a probability of at least p, and $\boldsymbol{P}_{\geq p}[\boldsymbol{G}\ \phi]$ means that ϕ holds continuously with a probability of at least p.

Finally, we define the *Lead-to* operator $\rightsquigarrow$, introduced originally by Owicki and Lamport [128]:

$$\boldsymbol{P}_{\geq p}[\phi_1 \rightsquigarrow \phi_2] \equiv \mathbf{A}\,\boldsymbol{G}\ (\phi_1 \rightarrow \boldsymbol{P}_{\geq p}\boldsymbol{F}\ \phi_2)$$

Intuitively, this operator enforces the system to eventually satisfy property ϕ_2 whenever ϕ_1 becomes true, with probability not lower than p.

2.2.3 Soundness and Completeness

In the rest of the dissertation, we will compare the runtime of the proposed verification and strategy-synthesis algorithms to other results reported in the literature. We thus need metrics to assess the correctness of the different approaches in order to fairly compare their runtime performance. In this dissertation, we use *soundness* and *completeness* as necessary conditions to determine the correctness of an algorithm.

Definition 2.21. *An algorithm is correct only if it is sound and complete.*

Intuitively, an algorithm is sound if any reported solution is indeed a solution of the analyzed problem. Further, an algorithm is complete if it always generates a solution for the analyzed problem if such a solution exists (dually, an algorithm is complete if no solution to the analyzed problem exists if the algorithm reports so).

We now define soundness and completeness specifically for the verification and optimal control problems analyzed in this dissertation.

2.2.3.1 Soundness and Completeness for Model-Checking Algorithms

The verification of a PCTL state formula ϕ can be viewed as a decision problem. The model-checking algorithm V needs to determine whether a state $s \in S_0$ is (or is not) contained in the set $Sat(\phi)$ defined in Equation (2.10). We can thus define the following properties for V:

Definition 2.22. ***Soundness of a model-checking algorithm.*** *A model-checking algorithm V for the verification of PCTL properties of Convex-MDPs is sound if a state $s \in S$ in $\mathcal{M}_\mathcal{C}$ does satisfy PCTL property ϕ if the algorithm reports so. Formally:*

$$s \in Sat_V(\phi) \Rightarrow s \in Sat(\phi)$$

where $Sat_V(\phi)$ is the computed satisfiability set, while $Sat(\phi)$ is the actual satisfiability set.

Definition 2.23. ***Completeness of a model-checking algorithm.*** *A model-checking algorithm V for the verification of PCTL properties of Convex-MDPs is complete if a state $s \in S$ in $\mathcal{M}_\mathcal{C}$ does not satisfy PCTL property ϕ if the algorithm reports so. Formally:*

$$s \notin Sat_V(\phi) \Rightarrow s \notin Sat(\phi)$$

where $Sat_V(\phi)$ is the computed satisfiability set, while $Sat(\phi)$ is the actual satisfiability set.

2.2.3.2 Soundness and Completeness for Strategy-Synthesis Algorithms

The strategy-synthesis problem from a PCTL specification ϕ can be viewed as a constrained optimization problem. The strategy-synthesis algorithm SS needs to determine the strategy $\sigma^* \in \Sigma_\phi$ that maximizes $\mathbb{W}^\sigma_{S_0}$. We can thus define the following properties for SS:

Definition 2.24. ***Soundness of a strategy-synthesis algorithm.*** *Given a Convex-MDP $\mathcal{M}_\mathcal{C}$, a strategy-synthesis algorithm SS from a PCTL specification ϕ is sound if the synthesized strategy σ^* does maximize $\mathbb{W}^\sigma_{S_0}$ among all strategies $\sigma \in \Sigma_\phi$, if the algorithm reports so. Formally:*

$$\sigma^*_{SS} = \underset{\sigma \in \Sigma_\phi}{argmax}\ \mathbb{W}^\sigma_{S_0} \Rightarrow \sigma^* = \underset{\sigma \in \Sigma_\phi}{argmax}\ \mathbb{W}^\sigma_{S_0}$$

where σ^*_{SS} is the strategy synthesized by SS, while σ^* is the actual strategy that maximizes $\mathbb{W}^\sigma_{S_0}$.

Definition 2.25. ***Completeness of a strategy-synthesis algorithm.*** *Given a Convex-MDP $\mathcal{M}_\mathcal{C}$, a strategy-synthesis algorithm SS from a PCTL specification ϕ is complete if the set of strategy $\sigma \in \Sigma_\phi$ is empty, if the algorithm reports so. In other words, algorithm SS is complete if it correctly reports that the constrained optimization problem is unfeasible. Formally:*

$$\Sigma_{\phi,SS} = \emptyset \Rightarrow \Sigma_\phi = \emptyset$$

where $\Sigma_{\phi,SS}$ is the set of strategies that satisfy specification ϕ computed by SS, while Σ_ϕ is the actual set of strategies that satisfy specification ϕ.

We will show in Chapter 4 and Chapter 6 that algorithms for the verification and optimal control of probabilistic (reward) formulas $\phi = \boldsymbol{P}_{\bowtie p}[\psi]$ ($\phi = \boldsymbol{R}_{\bowtie v}[\rho]$) in the presence of uncertainties require to solve convex optimization problems over the set $\mathbb{R}$ of the real numbers. Optima of these problems can be arbitrary real numbers, so, in general, they can be computed only to within a desired accuracy ϵ_d. We consider an algorithm to be sound and complete if the error in determining the satisfaction probabilities of ϕ is bounded by such a parameter ϵ_d, since the returned result will still be accurate enough in most practical applications.

Chapter 3

Related Work

In this chapter, we give an overview of related work presented in the literature both in the area of formal verification and of optimal control. We focus only on work related to the theoretical and algorithmic contributions of this dissertation, while we defer the presentation of the work related to the two analyzed case studies (the verification of the behavior of a human driver and the synthesis of energy pricing strategies in smart grids) to the corresponding chapters. We follow the structure of the dissertation in presenting the material of this chapter. We first discuss several modeling formalisms used to capture the behavior of stochastic systems. Second, we review formal logics introduced to express properties of these systems. Third, both for the verification and optimal control problem, we review a number of proposed algorithmic techniques and analyze their theoretical complexity. This study clarifies the contributions in theoretical complexity described in this dissertation. We conclude by presenting several model-checking tools proposed in the literature and by analyzing their capabilities.

3.1 Probabilistic Modeling Frameworks

In this section, we report modeling formalisms and formal logics introduced in the literature to model and express properties of stochastic systems. We will compare each of the reported results with the framework presented in Chapter 2 in terms of expressivity, i.e., how accurately and extensively they can represent system dynamics and properties.

Remark 3.1. *Overall, we will see that there is a clear trade-off between modeling expressivity and theoretical complexity of the model-checking and control problems. The most expressive models and formal logics can indeed capture many details of the analyzed systems and express intricate properties to check the correctness of their behavior. On the other hand, the resulting model-checking and optimal control problems are often intractable in the most general forms, and only approximated algorithms or algorithms operating on fragments of the logics exist to solve them.*

3.1.1 Modeling Formalisms

In this section, we review several modeling formalisms that have been used to capture the behavior of stochastic systems. In particular, we will focus on highlighting the differences in terms of modeling expressivity with respect to the Convex-MDP formalism introduced in this dissertation, in order to guide the reader in choosing the most appropriate modeling style for his or her specific application.

3.1.1.1 Discrete-Time Probabilistic Models

Stochastic models like Discrete-Time Markov Chains (DTMCs) [46] and Markov Decision Processes (MDPs) [26] have been widely used in the last decades to formally represent systems that exhibit random or probabilistic behaviors. In these models, the system execution is represented as a sequence of discretized steps among a finite set of states, each capturing a snapshot of the system dynamics. Transitions are performed randomly according to a pre-characterized probability distribution function. Moreover, both models satisfy the Markov property, i.e., the behavior of the future of the process only depends upon the current state and not on the rest of the execution history. In DTMCs, the system dynamics are purely stochastic and the system transitions between states autonomously at each execution step. On the other hand, in MDPs multiple possible *actions* are available at each state, and each action is associated to a different state-transition probability distribution. A controller or adversary can choose which action to take at each execution step to drive the system execution. We notice that DTMCs can be interpreted as a special case of MDPs, in which only one action is available at each state.

Convex-MDPs generalize MDPs by allowing transitions to be expressed in terms of convex sets of probability distributions, to capture uncertainties in the modeling process. A simplified formalism allowing only closed-interval sets to represent uncertainties has been first introduced by Kozine and Utkin in [93] as Interval-valued Discrete-Time Markov Chains (IDTMCs). Two semantic interpretations have been proposed for these models [150]: Uncertain Markov Chains (UMCs) and Interval Markov Decision Processes (IMDPs)[1]. A UMC is interpreted as a family of (possibly uncountably many) DTMCs, where each member of the family is a DTMC whose transition probabilities lie within the interval range given in the UMC. The transition probabilities for each state are fixed at the beginning of the execution and they remain the same across the whole execution. In IMDPs, instead, the uncertainty is resolved through non-determinism. Each time a state is visited, a transition distribution within the interval constraints is picked by the adversarial nature, and a probabilistic step is taken accordingly. Thus, IMDPs allow modeling a non-deterministic choice made from a set of (possibly uncountably many) choices. In this dissertation, we focused on this second semantic interpretation, as stated in Assumption 2.3.

[1] IMDPs are not to be confused with Interval-MDPs as introduced in Section 2.1.3.1. Interval-MDPs allow modeling non-determinism in state-transition probabilities, while IMDPs are DTMCs with closed-interval uncertainty sets. In fact, Interval-MDPs get reduced to IMDPs only after all non-determinism has been resolved by fixing a strategy or adversary for the Interval-MDP model.

The IDTMC formalism was then extended to consider also non-linear convex uncertainty models by Nilim and El Ghaoui [125] in the context of robust control. These modeling formalism have since then been used in applications in which it was important to capture uncertainties in the modeling process, due for example to modeling errors, non-modeled dynamics or inaccuracies in the estimation of transition probabilities. Such applications include the modeling of biological reactions [11], weather forecast [125], financial portfolio dynamics [23], and election-vote systems [63], just to name a few.

3.1.1.2 Continuous-Time Probabilistic Models

In the modeling formalisms presented so far, execution time evolves in fixed discrete steps. The dynamics of continuous time systems can thus only be approximated by setting the duration of the discretization step to be appropriately short, in order to capture all the desired system dynamics. While this discretization technique can indeed be effective for a broad category of systems (for example, the ones analyzed in this dissertation), other systems might require very short discretization steps and the representation of a long execution time in order to achieve the desired accuracy, thus making the discretization of their execution prohibitive from a computational perspective. This is the case for example for models of complex chemical reactions. Formalisms capable of capturing the concept of continuous (*real-valued*) time are thus desirable to model such systems.

Continuous-Time Markov Chains (CTMCs) have been proposed to address this problem [159]. In CTMCs, the time spent in each state is not fixed, and it instead takes non-negative real values according to a predefined exponential distribution. State transitions are then executed stochastically as in DTMCs, and the execution is represented as sequence of visited states. CTMCs have indeed received wide adoption both to model chemical reactions [8] and I/O queues in computational systems [3].

Probabilistic Timed Automata (PTA) have been proposed by Segala [149] to combine discrete probabilistic choices, real time and non-determinism. PTA are automata equipped with a finite set of real-valued clocks. In a PTA, a transition between *locations* is enabled when a guard condition, possibly including checks on the values of the clocks, gets satisfied. Transitions are instantaneous, but time elapses when the automata is within a location. When a transition is enabled, the next visited location is chosen stochastically according to a predefined probability distribution.

We note that both CTMCs and PTA can be transformed into MDPs by discretizing their execution into a finite number of steps [152, 100], so that the techniques for model checking and optimal control presented in this dissertation can be applied. In fact, we will show in Section 4.3.2.2 an example of such a transformation applied to a PTA model. However, the transformation usually comes at the cost of a substantial increase in the number of states in the model, in particular if the chosen discretization step is short with respect to the execution window to be analyzed. Verification and control techniques specialized in the analysis of these systems do exist and can sometimes achieve better performance in terms of accuracy of the results and runtime. On the other hand, these models have been overall less studied in the literature, so they do not benefit from the richness of theoretical and algorithmic results already developed for MDPs. The interested reader is referred to the work by Hartmanns et al. [74] for more details.

3.1.1.3 Partially-Observable Markov Decision Processes

In the modeling formalisms presented above, it was assumed that a full knowledge of the system to be modeled was available both at the time of model creation and during the execution of the model itself. Partially-Observable MDPs (POMDPs) [120] have been introduced to model systems for which such a full knowledge is not available and the uncertainty in estimating the system state is instead central to the modeling problem. These models have been used for a wide range of applications including human-machine interfaces, quality-control and machine replacement, robot path-planning and automated health-care assistance [120].

POMDPs model a decision process in which it is assumed that the system dynamics are determined by an underlying MDP, but the instantaneous state cannot be directly observed. When an action is taken and a transition to a new state is made, it is not possible to determine in which state the model has transitioned, but instead the model returns an *observation*, which gives a "hint" of what the new state is. Based on this observation, it is possible to maintain a probability distribution, called *belief state*, over the set of possible states. The belief state can then be used to estimate the probability of being in a specific state of the underlying MDP at each execution step. Instead of a sequence of visited states, a POMDP thus visits a sequence of belief states, based on the history of observations and on the observation probabilities specified for each state of the underlying MDP.

POMDPs are extremely expressive models, but their high expressivity has hindered the development of effective algorithms to analyze and control properties of these systems. In fact, most verification and control problems applied to POMDPs are *undecidable* or *intractable*, and only approximate algorithms have been developed. We report here only results about the synthesis of optimal strategies to control the execution of POMDPs[2]. In MDPs, the strategy-synthesis problem amounts to find an optimal mapping, i.e., a strategy, from states to actions, to determine which action to take in each state. In POMDPs, instead, the control problem aims to find a mapping from probability distributions over states, i.e., the belief states, to actions. Under the assumption that for each belief state there are a finite number of actions and a finite number of observations, there are only a finite number of possible next belief states, each corresponding to a pair (action, observation). The POMDP can thus be reduced to a continuous-space MDP, i.e., an MDP with uncountably infinite states. In this continuous-state MDP, each state represents a belief state of the original POMDP, hence the need for uncountably infinite states. On the other hand, the number of available transitions at each state of the continuous-space MDP is finite, and transition probabilities are easily derived from the transitions and observation probabilities of the POMDP. Adaptations of the classical strategy-synthesis algorithm for MDPs based on the Bellman recursion have then been applied to synthesize control strategies for the continuous-space MDPs [158, 85].

[2]We note that these algorithms simply focus on the maximization of some reward function of the POMDP without enforcing the model to satisfy any predefined specification during its execution, as we instead do for Convex-MDPs in the results presented in Chapter 6.

3.1.2 Formal Logics

In this dissertation, we use Probabilistic Computation Tree Logic as the formal logic to express properties of Convex-MDPs. We use this logic because it allows expressing *quantitative* properties of the analyzed Convex-MDP models, which are extremely useful in characterizing the performance of probabilistic systems. For example, we can express the property *"The probability that the mechanical arm hits a human operator while lifting a car is less than* 10^{-9}*"*, and then verify whether the property holds or not. However, PCTL is not the only formal logic used to express properties of probabilistic systems. In this section we review several alternative formal logics introduced in the literature, and compare them with PCTL in terms of expressivity. The list of analyzed formal logics is not meant to be comprehensive and it just reports the logics that are most used in the literature.

3.1.2.1 Qualitative Logics

Qualitative logics allow one to express only properties that need to hold almost certainly or almost never, i.e., with probability equal to 1 or 0, respectively. The most widely used qualitative logics are the Computation Tree Logic Star (CTL*) [54] and its two sub-logics Linear Temporal Logic (LTL) [137] and Computation Tree Logic (CTL) [40]. The syntax of CTL* is defined as follows:

$$\begin{aligned} \phi &::= True \mid \omega \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \mathbf{A}\,[\psi] \mid \mathbf{E}\,[\psi] && \text{state formulas} \\ \psi &::= \phi \mid \neg\psi \mid \psi_1 \wedge \psi_2 \mid \boldsymbol{\mathcal{X}}\,\psi \mid \boldsymbol{F}\,\psi \mid \boldsymbol{G}\,\psi \mid \psi_1\,\boldsymbol{\mathcal{U}}\,\psi_2 && \text{path formulas} \end{aligned}$$

where $\omega \in \Omega$ is an atomic proposition. This logic defines two path quantifiers. The universal quantifier **A** enforces the path formula ψ to hold across all execution paths; the existential quantifier **E** requires at least one path to satisfy ψ. The temporal operators are defined as follows. *Next* ($\boldsymbol{\mathcal{X}}$) is satisfied if ψ holds in the next state; *Globally* ($\boldsymbol{G}$) if ψ holds in all states; *Eventually* ($\boldsymbol{F}$) if it is possible to reach a state satisfying ψ; and *Unbounded Until* ($\boldsymbol{\mathcal{U}}$) if it is possible to reach a state satisfying ψ_2 while only visiting states satisfying ψ_1.

These temporal logics differ depending on how they handle branching in the underlying computation tree representing the system execution. In LTL, operators are provided for describing events along a single computation path. The logic thus disallows path quantifiers. CTL, instead, restricts the set of formulas allowed by CTL* by enforcing every temporal operator in the formula to be preceded by a path quantifier. Finally, CTL* combines both branching-time and linear-time operators. In fact, there exist properties that can be expressed in LTL and not in CTL and vice versa, while the more expressive CTL* combines all the properties that can be formulated using the two sub-logics.

CTL* and its sublogics allow to express *safety* (informally, *something negative will never happen*), *liveness* (informally, *something positive will eventually happen*) and *fairness* (informally, *something will happen infinitely often*) properties of a system. For example, using LTL, it is possible to express the liveness property *"Globally, if a request occurs, then it will be eventually acknowledged"*, using the syntax $\phi = \boldsymbol{G}\ (req \rightarrow \boldsymbol{F}\ ack)$.

Overall, qualitative logics are widely used for their simplicity and expressivity. These logics have been extensively studied in the last forty years and they are very well understood both by the research community and by the users of verification tools. A wealth of theoretical and algorithmic results have been developed to analyze properties of probabilistic systems (e.g., DTMCs, CTMCs, MDPs) and a number of efficient model-checking tools have been implemented, as it will be described in the following. On the other hand, these logics cannot express *quantitative* properties, a fundamental limitation in the analysis of systems that cannot be proven error-free under all circumstances, but whose behavior is still acceptable as long as the probability of failure is sufficiently low. Noticeably, any system including a physical component (e.g., a plant) belongs to this category. Such a physical component can be modeled using one of the formalism introduced in Section 3.1.1 and its probability of failure can be annotated within the model in terms of a transition probability to a failure state. A *quantitative* logic can then be used to query the probability for the system to transition to the failure state. Hence explained the importance of quantitative logics, which are introduced next.

3.1.2.2 Quantitative Logics

Quantitative logics allow to express properties that need to be satisfied with probability larger or smaller than a given real-valued threshold $0 \leq p \leq 1$.

Probabilistic-LTL (PLTL) is an extension of LTL, which maintains the full expressivity of LTL while adding the possibility to express also quantitative properties [15]. We will see in Section 3.2.1 that the high expressivity of this logic comes at the price of also higher complexity of the model-checking problem.

In this dissertation, we use Probabilistic-CTL (PCTL) as formal logic to represent properties of Convex-MDPs, as introduced in Section 2.2. PCTL is a logic derived from CTL, which disables the path-quantifier operators and instead introduces a probabilistic operator $\boldsymbol{P}$ [72]. We have shown in Section 2.2.2 that PCTL is a highly expressive logic, capable of specifying a wide range of system properties. On the other hand, PCTL does not allow arbitrarily nested path formulas, i.e., two path operators can only be connected using a state operator. Therefore, PCTL cannot express arbitrary liveness and fairness properties.

To overcome this shortcoming of the PCTL logic, alternative logics, such as ω-PCTL and Probabilistic Branching Time Logic (PBTL), have thus been developed [38, 19].

The logic ω-PCTL allows the specification of Büchi conditions and it can express any ω-regular condition. The syntax of the logic is defined as:

$$
\begin{aligned}
\phi &::= True \mid a \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \boldsymbol{P}_{\bowtie p}[\psi] && \text{state formulas} \\
\psi &::= \boldsymbol{\mathcal{X}}\ \phi \mid \phi_1\ \boldsymbol{\mathcal{U}}^{\leq \boldsymbol{k}}\ \phi_2 \mid \phi_1\ \boldsymbol{\mathcal{U}}\ \phi_2 \mid \psi^{\omega} && \text{path formulas} \\
\psi &::= \textbf{Büchi}(\phi) \mid \textbf{coBüchi}(\phi) \mid \psi_1^{\omega} \wedge \psi_2^{\omega} \mid \psi_1^{\omega} \vee \psi_2^{\omega} && \text{infinitary path formulas}
\end{aligned}
$$

where a is an atomic proposition. The canonical Rabin and Street conditions can be expressed as a conjunction and disjunction of Büchi and coBüchi conditions. As a consequence, ω-PCTL can express Rabin and Street conditions, and, in turns, any ω-regular property. We refer the reader

to the article by Thomas [163] for more details on this derivation. The logic ω-PCTL thus offers advantages in terms of expressivity in the specification of properties of stochastic systems. As in the case of PLTL, though, the higher expressivity comes at the cost of an increase of the complexity of the model-checking problem.

Certain liveness properties cannot be satisfied by concurrent systems unless some *fairness* assumptions on the behavior of the adversary is enforced. The logic PBTL was introduced to allow expressing such fairness assumptions on specifications expressed in PCTL [19]. In particular, although not as expressive as ω-PCTL, the logic PBTL can formulate properties enforcing both fairness or strong fairness conditions on the system behavior.

Finally, we introduce the Continuous Stochastic Logic (CSL), a logic specifically developed to express properties of CTMCs [14]. The syntax of the logic is defined as follows.

$$\phi ::= True \mid \omega \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \boldsymbol{P}_{\bowtie p}[\psi] \qquad \text{state formulas}$$
$$\psi ::= \phi_1 \, \boldsymbol{\mathcal{U}}_{[t_1,t_2]} \phi_2 \, \boldsymbol{\mathcal{U}}_{[t_3,t_4]} \cdots \qquad \text{path formulas}$$

where $0 \leq p \leq 1$ and $t_i \in \mathbb{R}$ is a real valued number expressing a time instant in an appropriate unit of measure (e.g., seconds). CSL is a quantitative logic, since it allows a probabilistic operator $\boldsymbol{P}$, analogous to the one defined for PCTL. Moreover, path formulas allow to express the property that a system will remain in a state satisfying ϕ_1 before making a transition within t_2 time units to a state satisfying ϕ_2, and to concatenate the syntax expressing these transitions for an arbitrary number of times.

3.2 Verification Algorithms

In this section, we review several formal techniques that have been proposed in the literature to verify properties of probabilistic systems. In particular, we will focus on results applying to MDPs, unless differently specified.

3.2.1 Model Checking

The *model-checking* problem can be formalized as a decision problem. Given a structure M and a property ϕ, the model-checking problem answers whether M satisfies property ϕ, written as $M \models \phi$. If M does not satisfy the property, i.e., $M \not\models \phi$, the model-checking algorithm produces a counterexample. This formal verification technique is a rigorous mathematical approach in proving the correctness of a system based on an exhaustive state-space exploration [41]. In the following, we give details about the complexity class of the model-checking problem for properties expressed using different kinds of formal logics, and briefly report the most successful algorithmic approaches used to solve the model-checking problem.

3.2.1.1 Model Checking Qualitative Properties

The problem of model checking a CTL formula on an MDP is in the class P and efficient model-checking algorithms and tools have been implemented [40]. Briefly, these algorithms rely on

the reachability analysis on the states of the underlying state-transition graph of the model. A property is satisfied if the system reaches at least one of the states satisfying the CTL formula almost certainly or almost never starting from any initial state.

On the other hand, for properties expressed with the full LTL logic, the model-checking problem for MDPs is known to be polynomial in the size of the model but double-exponential in the size of the LTL formula. In general, the model-checking algorithm for LTL is performed in two steps. First, the system model gets composed with the Büchi automaton representation of the property to be verified. This step is responsible for the exponential explosion of the model state space. Secondly, the LTL model-checking problem is reduced to checking whether an *accepting* state in the composition of the system with the Büchi automaton is visited infinitely often. Noticeably, as a consequence of the high worst-case time complexity for the general case, researchers [136] have focused instead on properties expressed with a subset of LTL, known as GR(1), for which polynomial-time model-checking algorithms exist.

3.2.1.2 Model Checking Quantitative Properties

The model-checking problem for PLTL properties of MDPs follows the same steps of the analogous problem for LTL formulas. A probability space is defined over the product of the original model to be verified and the Büchi automaton representing the property to be verified. The probability of satisfying the LTL specification is then equal to the probability of reaching a satisfying state in the product model. Given the similarity of the solution method, it is easy to prove that the model-checking problem is double exponential in the size of the formula and polynomial in the size of the system, as for the model checking of LTL formulas.

The model checking of PLTL formulas has been investigated also for the model of Interval Markov Chains (IMCs). In that setting, the complexity increases and the problem has been proven to be in EXPSPACE, and PSPACE-hard [22].

The model checking of PCTL properties for MDPs has been known to be in P for a long time [72]. The theoretical complexity of the problem of model checking PCTL specifications for IDTMCs was instead addressed by Sen et al. [150]. Since the two semantic interpretations of IDTMCs, as introduced in Section 3.1.1.1, yield different model-checking results, the complexity characterization was given separately for each interpretation. For the IMDP semantics, the problem was shown to be at most in PSPACE [150], and the result was later improved to co-NP by Chatterjee et al. [38]. These results rely on the construction of an equivalent MDP that encodes all behaviors of the IMDP. For each state in the new MDP, the available state transitions can be mapped to the Basic Feasible Solutions (BFSs) of the set of inequalities specifying the transition probabilities of the original IMDP. Since in the worst case the number of BFSs is exponential in the number of states of the IMDP, the equivalent MDP can have size exponential in the size of the original IMDP. The lower bound on the theoretical complexity of the problem was set by the same authors to be P.

In this dissertation, specifically in Chapter 4, we improve the previously best-known complexity result of co-NP [38] to P, for the fragment of PCTL in which we disallow all operators with a

Table 3.1: Algorithmic Complexity of PCTL Model Checking

Model	W/o $\boldsymbol{\mathcal{U}^{\leq k}}$, $\boldsymbol{\mathcal{I}^{=k}}$, $\boldsymbol{\mathcal{C}^{\leq k}}$	W/ $\boldsymbol{\mathcal{U}^{\leq k}}$, $\boldsymbol{\mathcal{I}^{=k}}$, $\boldsymbol{\mathcal{C}^{\leq k}}$
DTMC [72]	$O(poly(\mathcal{R}) \times \mathcal{Q})$	$O(poly(\mathcal{R}) \times \mathcal{Q} \times log(k_{max}))$
IMDP [38]	$O(exp(\mathcal{R}) \times \mathcal{Q})$	$O(exp(\mathcal{R}) \times \mathcal{Q} \times log(k_{max}))$
Convex-MDP [ours]	$O(poly(\mathcal{R}) \times \mathcal{Q})$	$O(poly(\mathcal{R}) \times \mathcal{Q} \times k_{max})$

finite time horizon (i.e., the *Bounded Until* ($\boldsymbol{\mathcal{U}^{\leq k}}$), *Instantaneous Reward* ($\boldsymbol{\mathcal{I}^{=k}}$) and *Bounded Cumulative Reward* ($\boldsymbol{\mathcal{C}^{\leq k}}$) operators). We also characterize the complexity of our algorithm for the full PCTL syntax based on the size $\mathcal{R}$ of the Interval-MDP model, the size $\mathcal{Q}$ of the PCTL formula, and the maximum bound k_{max} of the operators with a finite time horizon. Our algorithm runs in $O(poly(\mathcal{R}) \times \mathcal{Q} \times k_{max})$ time, which is pseudo-polynomial in k_{max} (i.e., polynomial if k_{max} is counted in its unary representation and exponential if k_{max} is counted in its binary representation). Moreover, the same complexity results hold also when non-linear (convex) models of uncertainty are used to express uncertainty in the state-transition probabilities, i.e., for the broader class of Convex-MDPs.

In the following, we give an intuitive explanation of why algorithms for the PCTL model checking of MDPs run in time polynomial in k_{max} counted in its binary representation, while the same result does not apply when considering Convex-MDPs. In particular, we restrict our attention on the Bounded Until operator, but the same reasoning applies also for the Instantaneous Reward and Bounded Cumulative Reward operators. The difference stems from the computation of the set $Sat\left(\boldsymbol{P}_{\bowtie p}\left[\phi_1\, \boldsymbol{\mathcal{U}^{\leq k}}\, \phi_2\right]\right)$. For MDPs, this computation involves raising the transition matrices $F^a, \forall a \in A$ to the $(k_{max})^{th}$ power, to model the evolution of the system in k_{max} steps. Matrix exponentiation is an operation that takes $O(log(k_{max}))$ steps. Matrix exponentiation can also be performed in the model-checking algorithm proposed by Sen et al. [150], on the transition matrix of the exponentially-larger MDP that encodes all behaviors of the original IMDP. As a consequence, the results on theoretical complexity for IMDPs [38, 150] can be extended to the full PCTL syntax (although the operators with a finite time horizon were not explicitly considered in those works). In our implementation of the model-checking algorithm, instead, we cannot do matrix exponentiation, because the transition matrix $F^a \in \mathcal{F}^a$ selected by the nature might change at each step, if the optimal nature is history-dependent. The unrolling of the execution history of the Convex-MDP thus needs to be done one step at a time. While the possibility of performing matrix exponentiation allows to derive a lower *theoretical* complexity result for MDPs, such a computation is seldom used in *practical* implementations of the model-checking algorithms, because the transition matrix of an MDP is often sparse and repeatedly squaring it would cause fill-ins [60]. Moreover, we note that both $\mathcal{Q}$ and k_{max} are typically small in practical applications [11, 101, 103], so the dominant factor for runtime is the size of the model $\mathcal{R}$. Hence, the increase in complexity results does not hinder the applicability of our approach to real-world case studies. We summarize the results in theoretical complexity in Table 3.1.

The model checking of ω-PCTL properties of DTMCs is decidable in P. On the other hand, the problem is only decidable in PSPACE (co-NP) when applied to the UMC (IMDP) semantic

interpretation of IDTMCs [38]. As a consequence, no efficient model-checking algorithms are known for IDTMCs and ω-PCTL has not found wide applicability for such models.

Also the model checking of PBTL properties of DTMCs and MDPs is decidable in P. On the other hand, the problem of model checking PBTL properties on Convex-MDPs has not been studied yet and it represents an interesting direction for future work.

Finally, the problem of model checking time bounded properties of CTMCs was proven to be decidable in P by Baier et al. [17]. This result was obtained via a reduction to the problem of computing transient state probabilities for CTMCs, which can be performed efficiently using techniques like *uniformisation*.

3.2.2 Statistical Model Checking

Statistical model checking has appeared as an alternative approach in the landscape of formal methods to address the problem of analyzing properties of stochastic systems [174]. Instead of exhaustively exploring all system behaviors, in statistical model checking the stochastic system under analysis is *simulated* for finitely many runs. Hypothesis testing is then used to infer whether the simulated samples provide a statistical evidence for the satisfaction or violation of the specification.

This technique has been applied to the estimation of the satisfaction probability of properties expressed in PCTL, among other logics, for a wide variety of modeling formalisms. In fact, the main advantage of statistical model-checking techniques is their capabilities of processing any stochastic system that can be simulated, without imposing any further constraint on the modeling approach. Moreover, statistical model checking is arguably able to process systems of large size, since their behavior does not need to be exhaustively explored but just simulated, possibly in parallel across multiple machines. While the accuracy of the estimated satisfaction probability does depend on the number of simulations, approximations of such probability are incrementally produced by the algorithm, so the designer can obtain intermediate results of the model-checking problem, which could already provide the required insight on system functionality.

On the other hand, statistical model checking suffers from several drawbacks. From a theoretical perspective, statistical model checking is not an exact method, and the computed satisfaction probabilities are just approximations of the real ones. This means that this technique can be only used to prove the *robust* satisfaction of a quantitative property, i.e., the actual probability of satisfying a given specification needs to be bounded away from the threshold to which it is compared. More specifically, given a structure M and a property ϕ, we call H_0 the hypothesis that $M \models \phi$, and H_1 the alternative hypothesis, i.e., that $M \not\models \phi$. The probability of accepting H_1 (H_0) given that H_0 (H_1) holds can only be bound to be at most $\alpha > 0$ ($\beta > 0$), but never set to be null, unless in the limit of infinite samples. The error bounds α and β are supplied as parameters to the model-checking procedure, and their value sets a trade-off between the accuracy in the result and the runtime of the verification algorithm. Furthermore, in terms of modeling expressivity, statistical model checking can only be used to analyze fully-stochastic systems, i.e., non-determinism cannot be resolved. The reason of such limitation lies in the fact that this verification technique

does not have any mechanism to resolve non-determinism, i.e., statistical model checking cannot determine what is the worst-case adversary for the probabilistic model.

Finally, we note that statistical model checking could be, in principle, applied also to the verification of stochastic models with uncertain transition probabilities, i.e., Uncertain Markov Chains[3]. In particular, the state-transition distribution could be randomly drawn from the uncertainty set at each execution step of the simulation, according to a uniform distribution across the uncertainty set. We note that such an approach would result in an estimation of the *expected* or *average* resolution of uncertainty within the uncertainty set. On the other hand, the model-checking algorithm presented in Chapter 4 is capable of computing the *worst-case* resolution of uncertainty within a convex uncertainty set. Both approaches can indeed provide useful insight to the designer, so they should both be applied to the analysis of system properties.

3.2.3 Approximate Probabilistic Bisimulation

We conclude the overview of formal methods for the verification of probabilistic systems by introducing the concept of probabilistic bisimulation.

Model-checking techniques are known to suffer from the state explosion problem, i.e., the number of states required to represent the system and the properties to be verified rapidly becomes too large to be efficiently processed. To overcome this problem, abstractions of the original systems can be created to reduce its state-space. On the other hand, it is fundamental to guarantee that such an abstraction preserves the same behavior of the system, at least in the context of the property to be verified. Bisimulation is an equivalence relation between the state-spaces of the original and abstracted systems often used in the context of formal verification. It should be noted, though, that the computation of an exact abstraction, i.e., one which is guaranteed to maintain the same system behavior, might be a task even more computationally intensive than the model-checking one performed on the original model. Approximate Bisimulations have thus been introduced to reduce the computational requirement of creating a system abstraction, while formally bounding the maximum introduced error in representing the dynamics of the original system.

These techniques have recently been applied to the problem of guaranteeing robustness to PCTL model checking by creating Approximate Probabilistic Bisimulations (APBs) of Markov Chains [48]. In particular, the existence of an APB to a Labeled Markov Chain (LMC) with precision ϵ is proven to imply the preservation of ϵ-robust PCTL formulas in the LMC. As a consequence, the APB can be used in place of the original system as long as an error bounded by the constant ϵ in the satisfaction probabilities can be tolerated.

This work differs from ours because we aim to check PCTL formulas of MDPs whose transition probabilities are affected by uncertainties due to estimation errors or imperfect information about the environment. In other words, these uncertainties have a clear "physical" meaning and they are not to be traced back to mathematical approximations generated in the process of abstraction of the system behavior. Further, our goal is different in that we aim to expose the effect of uncertainties

[3]We use the term Uncertain Markov Chains to refer to Markov chains with arbitrary uncertainty sets of state-transition probability distributions. Since in statistical model checking the system is simulated, we do not need to require the uncertainty set to be convex.

(small perturbations) in the model behavior on the satisfaction of the property that we are verifying, to guide the designer to appropriately model the system dynamics.

Finally, the concept of probabilistic bisimulations has recently been applied to Convex-MDP models [75]. The authors propose algorithms to compute the bisimulation quotients in time polynomial in the size of the model and exponential in the uncertain branching of the property to be verified. Experimental results show that a substantial state space reduction can be achieved by applying this approach, so that the algorithms developed in this dissertation can then be run more effectively.

3.2.4 Model-Checking Tools

In parallel to the formalization of the aforementioned theoretical results, the verification community has also worked on the development of tools for the model checking of probabilistic systems. In the following we briefly introduce the most relevant ones.

At the time of writing, the PRISM Model Checker appears to be at the frontier of the available model-checking tools [99]. The tool has been used to analyze a multitude of applications, from communication protocols and biological pathways to security problems. PRISM is capable of analyzing several different probabilistic models (e.g., discrete and continuous-time Markov Chains (DTMCs/CTMCs), Markov Decision Processes (MDPs), Probabilistic Timed Automata (PTA)) and properties expressed in a variety of specification languages (e.g., PCTL, LTL, CSL). Along the years, the tool has been used to verify and analyze properties of a heterogeneous set of systems, ranging from biochemical reactions and power-management units to security protocols and many others.

INFAMY [70] is a tool for the model checking of CSL formulas on infinite-state CTMCs. In particular, INFAMY is capable of handling the time-bounded subclass of the logic CSL. While conventional model checkers explore the given model exhaustively, a costly operation due to state explosion, or even an impossible one if the model is infinite, INFAMY only explores the model up to a finite depth, with the depth bound being computed on-the-fly. In particular, the computation of the depth bound is configurable to adapt to the characteristics of the specific class of the model under analysis and to the desired accuracy in the computed results. INFAMY has been applied to the study of protein-synthesis procedures and of several queuing protocols.

The Markov Reward Model Checker (MRMC) [86] supports PCTL and CSL (and the extensions of these logics to include rewards) model checking for several probabilistic models, including MDPs, DTMCs and CTMCs. MRMC supports reward-bounded reachability probabilities (a property is satisfied if the accumulated reward to reach the satisfying states is below/above a given threshold), property-driven minimization of the size of the system bisimulation, and precise on-the-fly steady-state detection.

The Erlangen-Twente Markov Chain Checker (ETMCC) [79] is a software tool which supports the automatic checking of properties given as CSL or aCSL [80] formulas. It supports models expressed with the formalism of finite CTMCs, labelled with atomic propositions and/or transition names. The tool has been used to verify the correct functionality of cyclic server polling systems and to detect software failures in a multiprocessor mainframe.

VESTA [151] is a tool to perform statistical analysis of probabilistic systems. In particular, it supports statistical model checking and statistical evaluation of expected values of temporal expressions. VESTA supports the verification of PCTL and CSL properties of both DTMCs and CTMCs. Furthermore, VESTA supports the statistical computation of expected values of expressions written in a query language called Quantitative Temporal Expressions (QUATEX). The tool has been used to verify system performance in several case studies, including the analysis of Denial-of-Service (DoS) security attacks, in which it is important to estimate the success probability of such attacks.

Finally, Ymer [173] is a tool for the model checking of PCTL and CSL properties of infinite-space DTMCs and CTMCs. It implements statistical model-checking techniques in order to be able to (approximately) solve also models with infinite state space. In particular, Ymer uses distributed discrete-event simulation to rapidly generate multiple execution traces and acceptance sampling to determine convergence of the model-checking algorithm.

Remark 3.2. *The development of a model-checking algorithm for PCTL properties of Convex-MDP is a novel contribution of this dissertation, since this formalism is not currently supported by any available tool. A Python implementation of the proposed algorithms is available [1]. Moreover, at the time of writing, the author is working on the integration of the proposed algorithms into the PRISM Model Checker distribution. The interested reader is invited to refer to the PRISM website [2] for more details about the date of the public release or to contact the author of this dissertation directly.*

3.3 Control Algorithms

The formalism of MDPs has originally been introduced by the work of Bellman [21] and Howard [81] in the context of the synthesis of control strategies for stochastic systems with the goal to maximize (minimize) a given reward (cost) function. We refer to this problem as an *unconstrained optimization* of the behavior of the MDP, since no additional constraint on the execution of the MDP is enforced.

More recently, MDPs have also become popular in the context of the synthesis of control strategies to satisfy a given specification expressed using a formal logic [16, 56, 103, 114, 148]. Indeed, formal logics are able to express more complex constraints to guide the execution of the decision process, a required feature in several applications (e.g., robot path-planning). We refer to this problem as a *feasibility* problem, since the synthesis algorithm aims to determine a strategy which satisfies the given specification, without requiring any specific system property to be optimized.

In Chapter 6, we will present the first sound and complete algorithm to synthesize strategies of Convex-MDPs capable of optimizing a given cost function and enforcing the model to obey to a specification expressed using the full PCTL syntax. Our algorithm can thus be seen as a *constrained optimization* and it combines the features of both approaches described above. Moreover, our algorithm can be applied also to the more general formalism of Convex-MDPs.

In this section, we review previous work in the areas of synthesis of unconstrained optimal strategies and of synthesis of control strategies from specifications in a formal logic.

3.3.1 Synthesis of Control Strategies for Unconstrained Reward Maximization

In this section, we will only consider the problem of reward maximization, but all the reported results are valid also for the dual problem of cost minimization. Several classes of cost functions have been considered in the literature to maximize the reward of the execution of an MDP [142]. In particular, we report:

- **Total Expected Reward.** By total reward, it is in generally meant the sum of the state and action rewards associated to an execution path of the MDP. The total expected reward criterion thus aims to find the control strategy for the MDP under analysis that maximizes the expected value of the sum of the state and action rewards across all the execution paths allowed by the control strategy.
- **Discounted Total Expected Reward.** The discounted total expected reward is a variant of the total expected reward that introduces *discounts*, i.e., weights, in the value of the state and action rewards, depending on how far away the states (actions) are visited (taken) in the system execution. In other words, if we consider a discount factor $0 \leq \lambda \leq 1$, the reward of a state (action) visited (taken) after k steps gets *discounted* by a factor λ^k. This approach has the advantage of weighting less the rewards that are farther away in the execution history, thus modeling the fact that events more remote in the future are less likely to be predicted, so they should be considered less in the computation of the instantaneous control strategy.
- **Average Reward.** The average reward criterion aims to maximize the average reward (and not the sum across an execution path) of the states (actions) visited (taken) during an execution path.

The problem of synthesizing optimal control strategies for MDPs for all the classes of cost functions introduced above can be solved in polynomial time for *finite-horizon* MDPs and, with mild assumptions on the utilized reward structure and on the structure of the MDP underlying graph, also for *infinite-horizon* MDPs [142]. These results descend from the fact that Markov deterministic strategies are optimal for the synthesis problem. Efficient algorithms based on dynamic programming have then been developed [142]. They rely on some variation of the famous Bellman recursion [21], presented in the following for the case of the synthesis of Markov deterministic strategies to optimize the total expected reward cost function.

$$V(s) = r_s(s) + \max_{a \in \mathcal{A}(s)} \left(r_a(s,a) + \sum_{s' \in S} f^a_{ss'} \times V(s') \right)$$

In the formula above, r is a reward structure for the MDP, $V(s)$ is a variable whose value represents the total expected reward for state $s \in S$, and $f^a_{ss'}$ is the probability of transitioning to state s' when starting from s and selecting action a. Intuitively, the maximization problem iterates across all the actions available at each state, and it selects the action which results in the highest expected reward.

The computation of such a recursion begins by assigning a total expected reward value $V(s_N)$ to the terminal states of the MDP and then recurses backwards in the execution path of the MDP by computing at each iteration the total expected reward of the states visited one step earlier, until all the initial states are reached.

The problem of synthesizing control strategies to maximize the total expected reward and the discounted expected reward for Convex-MDPs has been studied by Nilim and El Ghaoui [125]. Also for Convex-MDPs, the synthesis problem can be solved in time polynomial in the size of the model by applying a clever modification of the Bellman recursion, which computes at each recursion step the worst-case resolution of the uncertainty in the state-transition probability distribution within the convex uncertainty set. In particular, the authors solve the following "max-min" problem:

$$V(s) = r_s(s) + \max_{a \in \mathcal{A}(s)} \min_{\mathbf{f}_s^a \in \mathcal{F}_s^a} \left(r_a(s,a) + \sum_{s' \in S} f_{ss'}^a \times V(s') \right)$$

where we remind that $\mathcal{F}_s^a$ is the convex uncertainty set associated to state $s \in S$ and action $a \in \mathcal{A}(s)$. The main contribution in the work by Nilim and El Ghaoui [125] is the development of polynomial-time optimization routines based on duality theory to solve the inner optimization problem.

3.3.2 Synthesis of Control Strategies from Specifications in a Formal Logic

The problem of strategy synthesis for MDPs from PCTL specifications was first studied by Baier et al. [16]. Control strategies for MDPs are divided into four categories depending on: 1) whether the transition is chosen deterministically (D) or randomly (R); 2) the choice does (does not) depend on the sequence of previously visited states (Markov (M) and history-dependent (H)). It was proven that the four types of strategies form a strict hierarchy:

$$\text{MD} \prec \text{MR} \prec \text{HD} \prec \text{HR}$$

where we use $A \prec B$ to indicate that class of strategies A is strictly less powerful than class of strategies B, i.e., there exist MDPs $\mathcal{M}_{\mathcal{C}}$ and PCTL formulas ϕ such that for no strategy in A $\mathcal{M}_{\mathcal{C}}$ can satisfy ϕ, whereas $\mathcal{M}_{\mathcal{C}}$ can satisfy ϕ for some strategy in B. Moreover, the problem of determining whether it exists an MD/MR (HD/HR) strategy that meets all specifications was found to be NP-complete (elementary).

Kučera et al. [96] showed how to synthesize MR controllers that are robust to linear perturbations via a reduction to a formula in the first-order logic of reals. This work is the closest to the results presented in Chapter 6, albeit we consider also non-linear convex models of uncertainties. Further, the monolithic formulation of the synthesis problem proposed in that work might cause an explosion of the algorithmic runtime. In fact, to the best of our knowledge, the algorithm has not been applied to any real-world case study.

The algorithm for the synthesis of control strategies presented in Chapter 6 is the first sound and complete algorithm capable of processing formulas expressed in the full PCTL syntax. Other

algorithms to solve simplified versions of the synthesis problem have previously been presented in the literature. Lahijanian et al. [103] and Kwiatkowska and Parker [102] adapted routines for the model checking of PCTL properties of MDPs to the strategy-synthesis problem. These algorithms run in time polynomial in the model size, but they are not complete [103] or can handle properties with only one quantitative operator [102].

The synthesis of multi-strategies for MDPs is studied by Draeger et al. [50]. A multi-strategy is a function that associates not only one action $a \in \mathcal{A}(s)$ to each state of the model, but the whole subset of actions $A_{sat} \in \mathcal{A}(s)$ available at each state that guarantee the satisfiability of the given PCTL specification. The advantage of this approach is that the controller can then arbitrarily select any strategy among those allowed by the synthesized multi-strategy, according, for example, to additional constraints or to a cost function not specified (because unknown) at the time of synthesis. The proposed monolithic formulation of the multi-strategy synthesis problem can, on the other hand, handle only a subset of PCTL properties.

A large body of research has been developed to consider the synthesis problem within the context of game theory. So far, we have considered a synthesis problem in which all model states are either controllable (the controller can select the action to take in that state) or purely stochastic (there is only one action available at the state), a problem referred to in game-theory as a $1\frac{1}{2}$-player game. In the more general version of the problem, some of the states in which non-determinism is present are not controllable. These states are referred to as *environment*, and the choice of the action to take in such states is set by an *adversary*. Solving a $2\frac{1}{2}$-player game amounts to find a strategy for the controller such that the given specification is satisfied for any adversary operating on the environment states. Within the context of $2\frac{1}{2}$-player games, researchers have studied the theoretical complexity and proposed synthesis algorithms both for qualitative [6, 36, 44, 43, 59] and quantitative [7, 31, 37] specifications expressed in a variety of formal logics.

In particular, Brazdil et al. [31] studied two-player games with winning objectives expressed in PCTL. Also our formulation can be interpreted as a game, where the controller plays against nature. On the other hand, following the Convex-MDP semantics defined in Assumption 2.3, we give nature the power of selecting a different strategy at each execution step, while the work presented in [31] aims to analyze the more complex problem of finding the single optimal strategy for nature. Our formulation is useful to model time-varying processes (e.g., the power generated by renewable sources of energy), and an optimal strategy for the controller can be synthesized algorithmically, as it will be shown in Chapter 6. The existence of an optimal strategy for the controller and nature in the formulation of [31] is instead undecidable in general, and it becomes decidable only for a fragment of the PCTL logic.

Finally, Wolff et al. [172] studied the problem of synthesizing an optimal control for Convex-MDPs satisfying a specification expressed in the fragment of the PLTL logic which allows to use only one probabilistic operator. The proposed technique first converts the LTL specification to a Rabin automaton, whose size is worst-case doubly exponential in the size of the LTL formula. It then composes the automaton with the MDP, and it finally runs the robust dynamic programming procedure proposed by Nilim and El Ghaoui [125] to solve for the optimal control policy.

Chapter 4

Probabilistic Model-Checking with Uncertainties

In this chapter, we present our contributions regarding the formal verification of PCTL properties of Convex-MDPs. The chapter is divided in three sections. In the first section, we formally define the problem of PCTL model checking for Convex-MDPs and prove the main results about the theoretical complexity of this problem. In the second section, we give details about the routines for the verification of the temporal operators defined in the PCTL syntax. In particular, we prove the soundness and completeness of each routine and derive its algorithmic complexity. Finally, in the third section, we overview the software implementation of the previously described routines and apply the model-checking algorithm to the analysis of three case studies. We use this experimental evaluation both to evaluate the impact of modeling uncertainties on model-checking results and to assess the runtime performance of the proposed algorithm on problems of increasing size.

4.1 Theoretical Complexity of PCTL model checking for Convex-MDPs

Our proof of the results about the theoretical complexity of PCTL model checking for Convex-MDPs is constructive, i.e., we develop a sound and complete algorithm to solve the model-checking problem, characterize its algorithmic complexity and use this result to set an upper-bound on the theoretical complexity of the problem. Since we will show that the computed upper-bound coincides with the previously known lower-bound P, as described in Section 3.2.1.2, we conclude that our results characterize exactly the theoretical complexity of the problem.

4.1.1 Problem Definition and Algorithm Overview

We begin by formally define the problem of PCTL model checking for Convex-MDPs.

Definition 4.1. ***PCTL Model Checking for Convex-MDPs.*** ***Given*** *a Convex-MDP $\mathcal{M}_{\mathcal{C}} = (S, S_0, A, \Omega, \mathcal{F}, \mathcal{A}, \mathcal{X}, L)$ of size $\mathcal{R}$ and a PCTL formula ϕ of size $\mathcal{Q}$ over a set of atomic propositions Ω,* ***verify*** *whether $\mathcal{M}_{\mathcal{C}}$ models ϕ, i.e., $\mathcal{M}_{\mathcal{C}} \models \phi$, by computing the set $Sat(\phi) \in S$ of states satisfying ϕ over the uncertainty sets $\mathcal{F}_s^a \in \mathcal{F}$, and checking if the set of initial states S_0 is contained in $Sat(\phi) \in S$. Formally:*

$$\mathcal{M}_{\mathcal{C}} \models \phi \quad \Leftrightarrow \quad S_0 \subseteq Sat(\phi)$$

We now overview the algorithmic procedure used to model check PCTL properties of Convex-MDPs. The overview will clarify what are the routines that need to be developed to solve the model-checking problem.

As in the verification of CTL [72], the algorithm traverses bottom-up the parse tree for ϕ, recursively computing the set $Sat(\phi')$ of states satisfying each sub-formula ϕ'. At the end of the traversal, the algorithm computes the set $Sat(\phi)$ of states satisfying ϕ and it determines if $\mathcal{M}_{\mathcal{C}} \models \phi$ by checking if $S_0 \subseteq Sat(\phi)$. For the *qualitative* PCTL operators, the satisfying states are computed as follows:

$$Sat\left(True\right) = S \tag{4.1}$$
$$Sat(\omega) = \{s \in S \mid \omega \in L(s)\} \tag{4.2}$$
$$Sat(\neg\phi) = S \setminus Sat(\phi) \tag{4.3}$$
$$Sat(\phi_1 \wedge \phi_2) = Sat(\phi_1) \cap Sat(\phi_2) \tag{4.4}$$

For the *quantitative* PCTL operators, the algorithm needs to compute:

$$Sat\left(\boldsymbol{P}_{\lhd p}\left[\psi\right]\right) = \{s \in S \mid P_s^{max}(\psi) \lhd p\} \tag{4.5}$$
$$Sat\left(\boldsymbol{P}_{\rhd p}\left[\psi\right]\right) = \left\{s \in S \mid P_s^{min}(\psi) \rhd p\right\} \tag{4.6}$$
$$Sat\left(\boldsymbol{R}_{\lhd v}\left[\rho\right]\right) = \{s \in S \mid \mathbb{E}_s^{max}(\rho) \lhd v\} \tag{4.7}$$
$$Sat\left(\boldsymbol{R}_{\rhd v}\left[\rho\right]\right) = \left\{s \in S \mid \mathbb{E}_s^{min}(\rho) \rhd v\right\} \tag{4.8}$$

for each of the temporal operators defined in Section 2.2, i.e., the *Next* ($\boldsymbol{\mathcal{X}}$), *Bounded Until* ($\boldsymbol{\mathcal{U}}^{\leq k}$), *Unbounded Until* ($\boldsymbol{\mathcal{U}}$), *Instantaneous Reward* ($\boldsymbol{\mathcal{I}}^{=k}$), *Bounded Cumulative Reward* ($\boldsymbol{\mathcal{C}}^{\leq k}$) and *Cumulative Reward* ($\boldsymbol{\mathcal{C}}$) operators.

We will thus need to develop routines to compute each of the sets defined above. Further, we require a routine to be sound and complete, as defined in Section 2.2.3.1, to be suitable to correctly solve the model-checking problem.

For the *qualitative* fragment of the PCTL logic, we can leverage results already presented in the literature [5, 60]. It is also easy to prove that these routines are sound and complete, because they are based on reachability analysis over the finite number of states of $\mathcal{M}_{\mathcal{C}}$ [60]. We will thus assume these routines to be available in the rest of the dissertation.

In this chapter, we will instead focus on developing sound and complete routines to compute Sets (4.5) - (4.8) for Convex-MDPs and we will characterize their algorithmic complexity to set an upper bound on the theoretical complexity of the model-checking problem.

4.1.2 Optimal Adversaries and Natures

According to the semantics of the PCTL logic, as defined in Section 2.2.1.1, a state $s \in S$ verifies (models) a property ϕ if and only if the state satisfies ϕ for all adversaries $\alpha \in Adv$ and natures $\eta^a \in Nat$. Nevertheless, in order to guarantee the termination of the verification algorithm, we cannot enumerate all adversaries and natures and verify ϕ for each. In fact, there would be uncountably infinite natures to be verified, given that we defined the uncertainty sets in the continuous space $\mathbb{R}^N$. We instead need techniques to quickly determine the optimal (in the worst-case sense) adversary and nature and just verify that the property is satisfied when such an optimal adversary and nature control the execution of the Convex-MDP, to then infer that the property holds (does not hold) for all adversaries and natures.

In this section, we determine the optimal adversary and nature for each temporal operator of the PCTL logic. Such a classification will then guide the development of the corresponding model-checking routines.

We begin by analyzing the temporal operators not containing any finite bound on the number of steps of execution of the Convex-MDP, i.e., the *Next* ($\boldsymbol{\mathcal{X}}$), *Unbounded Until* ($\boldsymbol{\mathcal{U}}$) and *Cumulative Reward* ($\boldsymbol{\mathcal{C}}$) operators. Starting from a result presented by Puterman about MDPs [142], we prove that there always exist optimal Markov deterministic (MD) adversaries and natures to verify these properties. Intuitively, this means that we will need to determine only one optimal adversary and nature, which will be chosen deterministically. Moreover, the Markov property means that we need to consider only one step of the execution of the Convex-MDP, so the optimal adversary will simply be one of the actions $a \in \mathcal{A}(s)$ available at state $s \in S$ and the optimal nature will be a point $\mathbf{f}_s^a \in \mathcal{F}_s^a$ in the convex set $\mathcal{F}_s^a$ associated to action a at state s. Formally, we can compute Sets (4.5) - (4.8) by solving for each state $s \in S$ the optimization problems on the left in the following equations and by comparing (on the right) the computed optimal value with the satisfaction thresholds p for the $\boldsymbol{P}$ operator and r for the $\boldsymbol{R}$ operator:

$$P_s^{max}[\psi] = \max_{a \in \mathcal{A}(s)} \max_{\mathbf{f}_s^a \in \mathcal{F}_s^a} P_s(a, \mathbf{f}_s^a)[\psi] \quad \Rightarrow \quad P_s^{max}[\psi] \overset{?}{\leq} p \tag{4.9}$$

$$P_s^{min}[\psi] = \min_{a \in \mathcal{A}(s)} \min_{\mathbf{f}_s^a \in \mathcal{F}_s^a} P_s(a, \mathbf{f}_s^a)[\psi] \quad \Rightarrow \quad P_s^{min}[\psi] \overset{?}{\geq} p \tag{4.10}$$

$$\mathbb{E}_s^{max}[\rho] = \max_{a \in \mathcal{A}(s)} \max_{\mathbf{f}_s^a \in \mathcal{F}_s^a} \mathbb{E}_s(a, \mathbf{f}_s^a)[\rho] \quad \Rightarrow \quad \mathbb{E}_s^{max}[\rho] \overset{?}{\leq} v \tag{4.11}$$

$$\mathbb{E}_s^{min}[\rho] = \min_{a \in \mathcal{A}(s)} \min_{\mathbf{f}_s^a \in \mathcal{F}_s^a} \mathbb{E}_s(a, \mathbf{f}_s^a)[\rho] \quad \Rightarrow \quad \mathbb{E}_s^{min}[\rho] \overset{?}{\geq} v \tag{4.12}$$

We prove the optimality of Markov deterministic adversaries and natures in the following proposition.

Proposition 4.1. *Given a Convex-MDP $\mathcal{M}_{\mathcal{C}}$ and a target state $s_t \in S$, there always exist deterministic and memoryless adversaries and natures for $\mathcal{M}_{\mathcal{C}}$ that achieve the maximum/minimum probability/reward of reaching s_t, if the following conditions hold:*

- *A is finite;*
- *the inner optimization in Problems (4.9) – (4.12) always attains its optimum $\nu_s^*(a)$ over the sets $\mathcal{F}_s^a, \forall s \in S, \forall a \in \mathcal{A}(s)$, i.e., there exists a finite feasible $\mathbf{f}_s^a \in \mathcal{F}_s^a$ such that $P_s(a, \mathbf{f}_s^a)[\psi] = \nu_s^*(a)$ ($\mathbb{E}_s(a, \mathbf{f}_s^a)[\rho] = \nu_s^*(a)$).*

Sketch of proof. The proof is divided into two parts. First, we use Banach fixed-point theorem [142] to prove the existence of an adversary and a nature that achieve the maximum (minimum) probabilities of reaching s_t. Second, we prove that at least one such adversary and nature is memoryless and deterministic. The proof extends the one in Puterman [142], Theorem $6.2.10$. We need to prove that Problems (4.9) – (4.12) always attain the maximum (minimum) over the feasibility sets $\mathcal{F}_s^a$, i.e., $\forall s \in S, \forall a \in \mathcal{A}(s), \exists \mathbf{f}_s^a \in \mathcal{F}_s^a : ||\mathbf{f}_s^a||_2 < \infty, P_s(a, \mathbf{f}_s^a)[\psi] = \nu_s^*(a)$ ($\mathbb{E}_s(a, \mathbf{f}_s^a)[\psi] = \nu_s^*(a)$). This is indeed true for all the convex sets $\mathcal{F}_s^a$ considered in this dissertation. The interval and ellipsoidal models result in *compact* sets $\mathcal{F}_s^a$ on which Weierstrass theorem holds. For the likelihood and entropy model we use the notion of *consistency*, which guarantees the existence and uniqueness of a point in $\mathcal{F}_s^a$ where the optimum is attained. □

The verification of temporal operators containing a finite bound on the number of steps of execution of the Convex-MDP, i.e., the *Bounded Until* $\left(\boldsymbol{\mathcal{U}}^{\leq \boldsymbol{k}} \right)$, *Instantaneous Reward* $\left(\boldsymbol{\mathcal{I}}^{=\boldsymbol{k}} \right)$ and *Bounded Cumulative Reward* $\left(\boldsymbol{\mathcal{C}}^{\leq \boldsymbol{k}} \right)$ operators, requires instead history-dependent deterministic (HD) adversaries and natures in the general case. As an example, we verify the property $\phi = \boldsymbol{R}_{\leq 1}[\boldsymbol{\mathcal{I}}^{=\mathbf{3}}]$ for the Convex-MDP shown in Figure 4.1. In order to do so, we need to compute $\mathbb{E}_{s_0}^{max}[\boldsymbol{\mathcal{I}}^{=\mathbf{3}}]$. It is easy to see that the optimal adversary takes the self-loop b twice and then selects action a, which yields expected instantaneous reward of 2. No Markov adversary can achieve such reward value. Intuitively, the adversary may need to *wait* in some states until the time comes to take a step towards states in which the reward is large.

As a consequence, the verification of temporal operators containing a finite time horizon will require the unrolling of k steps of the execution of the Convex-MDP. For each step, following a reasoning similar to the one in Proposition 4.1, it is possible to show that deterministic adversaries

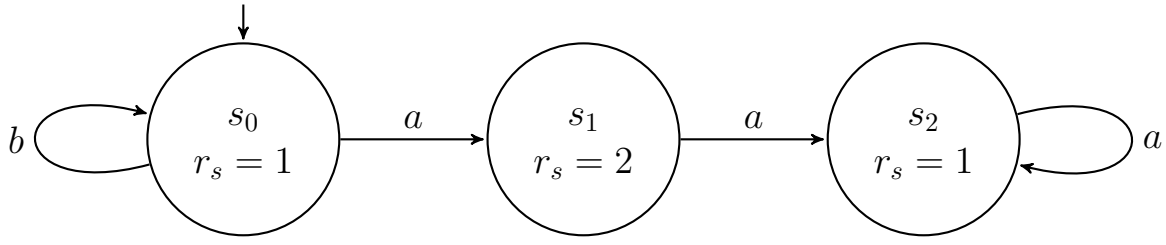


Figure 4.1: Example of a Convex-MDP for which only history-dependent adversaries are optimal.

and natures are optimal. In conclusion, we will need to solve Problems (4.9) – (4.12) k times for each state and then collect the results of the last iteration to compute Sets (4.5) - (4.8).

Remark 4.1. *The careful reader might find counter-intuitive that the model checking of the* Until *operator simply requires the computation of a Markov adversary and nature, while a history-dependent adversary and nature are required for the* Bounded Until *operator, given that the former can be interpreted as the limit version of the latter for $k \to +\infty$. The explanation comes from realizing that the* Until *operator characterizes a* steady-state *property of the Convex-MDP, i.e., a regime in which the adversary and nature always choose the same action and transition probability distribution at each step of the execution. As a consequence, only the steady-state action and transition probability distribution need to be determined to correctly compute the satisfaction probabilities. On the other hand, the* Bounded Until *operator characterizes a* transient *property, i.e., a regime in which the adversary and nature might choose a different action or probability distribution in consecutive steps, depending on the dynamics of the Convex-MDP. Consequently, each step of the execution of the model needs to be characterized to correctly compute the satisfaction probabilities.*

The same reasoning applies also to the Cumulative Reward *and* Bounded Cumulative Reward *operators.*

4.1.3 New Results in Complexity

Using the results of Section 4.1.2, we developed model-checking routines to solve Problems (4.9) – (4.12) for each temporal operator allowed in the PCTL syntax. These routines encode the transitions of $\mathcal{M}_{\mathcal{C}}$ into convex programs and solve them. From the returned solution, it is then possible to determine the *quantitative* satisfaction probabilities $P_s^{max/min}[\psi]$ ($\mathbb{E}_s^{max/min}[\rho]$) $\forall s \in S$, which get compared to the threshold p (v) to compute the sets $Sat\left(\boldsymbol{P}_{\bowtie p}\left[\psi\right]\right)$ ($Sat\left(\boldsymbol{R}_{\bowtie v}\left[\rho\right]\right)$).

To prove the results on algorithmic complexity for the developed model-checking routines, we use the following key result from convex theory [124].

Proposition 4.2. *Given the convex program:*

$$\begin{aligned} \min_{\mathbf{x}} \; & f_0(\mathbf{x}) & \\ \text{s.t. } & f_i(\mathbf{x}) \leq 0 & i = 1, \cdots, m \\ & h_j(\mathbf{x}) = 0 & j = 1, \cdots, p \end{aligned}$$

with $\mathbf{x} \in \mathbb{R}^n$, $f_i, i = 1, \cdots, m$ convex functions and $h_j, j = 1, \cdots, p$ affine functions, the optimum ν^* can be found to within $\pm\epsilon_d$ in time complexity that is polynomial in the size of the problem (n, m, p) and $\log(1/\epsilon_d)$.

We are now ready to state the main contribution of this chapter:

Theorem 4.1. ***Complexity of PCTL model checking for Convex-MDPs.***

1. *The problem of model checking a PCTL formula ϕ without any temporal operator with a finite time horizon (i.e., only using the $\boldsymbol{\mathcal{X}}$, $\boldsymbol{\mathcal{U}}$, $\boldsymbol{\mathcal{C}}$ operators) on a Convex-MDP $\mathcal{M}_{\mathcal{C}}$ of size $\mathcal{R}$ is in P.*

2. *A formula ϕ' containing temporal operators with a finite time horizon can be model checked with time complexity $O\left(poly(\mathcal{R}) \times pseudo{-}poly(\mathcal{Q})\right)$, i.e., polynomial in the size of $\mathcal{M}_{\mathcal{C}}$ and pseudo-polynomial in the maximum time horizon k_{max} of the $\boldsymbol{\mathcal{U}^{\leq k}}$, $\boldsymbol{\mathcal{I}^{=k}}$, $\boldsymbol{\mathcal{C}^{\leq k}}$ operators.*

Sketch of proof. For the first part, we start by reminding from Chapter 3 that the lower bound on the theoretical complexity of the problem of PCTL model checking for Convex-MDPs is P. We are thus left to show that the upper-bound coincides with the lower-bound to establish the results on theoretical complexity.

The proof is constructive. Our verification algorithm parses ϕ in time linear in the number of operators of ϕ [72], computing the satisfiability set of each operator. For the non-probabilistic operators, the satisfiability sets can be computed in time polynomial in $\mathcal{R}$ using set operations, i.e., set inclusion, complementation and intersection.

For the probabilistic and reward operators, we leverage Proposition 4.2 and prove that the proposed verification routines:

1. solve a number of convex problems polynomial in $\mathcal{R}$;

2. generate these convex programs in time polynomial in $\mathcal{R}$.

The correctness and time-complexity for formulas involving the *Next* ($\boldsymbol{\mathcal{X}}$), *Unbounded Until* ($\boldsymbol{\mathcal{U}}$) and *Cumulative Reward* ($\boldsymbol{\mathcal{C}}$) operators are formalized in Lemma 4.1, 4.3 and 4.7 (Section 4.2.1, 4.2.3 and 4.2.6). It thus follows that the overall algorithm runs in time polynomial in $\mathcal{R}$ and in the size of ϕ.

For the operators with a finite time horizon, we will need to unroll up to k_{max} steps of the execution of the Convex-MDP and for each step solve the convex Problems (4.9) – (4.12), which can be done in time polynomial in $\mathcal{R}$. This explains the additional term in the formula of the algorithmic complexity. We notice that the term pseudo-polynomial refers to the fact that the algorithmic complexity is polynomial (exponential) in k_{max} if k_{max} is represented in unary (binary) format in computer memory. The detailed results about the *Bounded Until* ($\boldsymbol{\mathcal{U}^{\leq k}}$), *Instantaneous Reward* ($\boldsymbol{\mathcal{I}^{=k}}$) and *Bounded Cumulative Reward* ($\boldsymbol{\mathcal{C}^{\leq k}}$) operators are formalized in Lemma 4.2, 4.5 and 4.6 (Section 4.2.2, 4.2.4 and 4.2.5). □

Referring to the material presented in Section 4.1.2, we can interpret the results in Theorem 4.1 as stating that the PCTL formulas that admit optimal Markov deterministic adversaries and natures can be model checked in time polynomial in the size of the formula, while PCTL formulas that require history-dependent deterministic adversaries and natures need additional algorithmic complexity to unroll the execution of the analyzed Convex-MDP. For both classes of formulas, the model-checking algorithm is instead polynomial in the size $\mathcal{R}$ of the Convex-MDP.

4.2 Model-Checking Routines

In this section, we give details about the routines used to model check the *quantitative* operators allowed in the PCTL syntax, i.e., the probability $\boldsymbol{P}$ and reward $\boldsymbol{R}$ operators. We instead refer the reader to the work by Forejt et al. [60] for details about the model-checking routines of the *qualitative* fragment of the logic, which rely on reachability analysis of the Convex-MDP underlying graph and whose results do not change after introducing uncertainties in the model.

For brevity, we only consider properties in the form $\phi = \boldsymbol{P}_{\lhd p}[\psi]$ ($\phi = \boldsymbol{R}_{\lhd v}[\rho]$), but the results can trivially be extended to $\phi = P_{\rhd p}[\psi]$ ($\phi = \boldsymbol{R}_{\rhd v}[\rho]$) by replacing the optimization operator "max" ("min") with "min" ("max") in the optimization problems derived in the following.

4.2.1 *Next* Operator

The *Next* operator $\boldsymbol{\mathcal{X}}\,[\phi_1]$ computes the states from which it is possible to reach a state satisfying ϕ_1 in one execution step. For example, if the Convex-MDP models an assembly line assigning a state to each station of the line, the *Next* operator might be used to verify that all states are capable of reaching a state further along in the assembly line within one execution step, to avoid backlogging.

Formally, properties for this operator are expressed as:

$$\phi = \boldsymbol{P}_{\lhd p}[\boldsymbol{\mathcal{X}}\ \phi_1]$$

The pseudocode of the model-checking routine for the *Next* operator is shown in Algorithm 4.1. First, the set $S^{yes} = Sat(\phi_1)$ of all states satisfying ϕ_1 is computed (line 3). Second (line $4-6$), for each state $s \in S$, the algorithm computes the satisfaction probabilities as defined in Equation (4.9) considering only the immediate transitions from each state, i.e., by solving the problem:

$$P_s^{max}[\boldsymbol{\mathcal{X}}\ \phi_1] = \max_{a \in \mathcal{A}(s)} \max_{\mathbf{f}_s^a \in \mathcal{F}_s^a} \sum_{s' \in S^{yes}} f_{ss'}^a \tag{4.13}$$

Algorithm 4.1: Next Operator.

1: **Input** $\mathcal{M}_{\mathcal{C}} = (S, S_0, A, \Omega, \mathcal{A}, \mathcal{F}, L)$, $\phi = \boldsymbol{P}_{\lhd p}[\boldsymbol{\mathcal{X}}\ \phi_1]$
2: **Output** The set $y = Sat(\phi)$
3: Compute $S^{yes} = Sat(\phi_1)$
4: **for all** $s \in S$ **do**
5: $\quad P_s^{max}[\mathcal{X}\phi_1] = \max_{a \in \mathcal{A}(s)} \max_{\mathbf{f}_s^a \in \mathcal{F}_s^a} \sum_{s' \in S^{yes}} f_{ss'}^a$
6: **end for**
7: $y = \{s \in S \mid P_s^{max} \lhd p\}$

In Problem (4.13), the inner optimization problem is a convex program since $\mathcal{F}_s^a$ is one of the convex sets introduced in Section 2.1.3. Finally, the computed probabilities are compared to the threshold p to select the states that satisfy ϕ (line 7).

Remark 4.2. *The sets $\mathcal{F}_s^a$ can be expressed with models of uncertainty different from one another $\forall s \in S, \forall a \in \mathcal{A}(s)$, since each instance of the inner optimization problem in Problem 4.13 gets solved independently from the others.*

Lemma 4.1. ***Model Checking of the* Next *Operator.*** *The routine to verify the* Next *operator is sound, complete and guaranteed to terminate with algorithmic complexity that is polynomial in the size $\mathcal{R}$ of $\mathcal{M}_\mathcal{C}$.*

Proof. From Problem (4.13) we see that there is one "inner" convex program for each state $s \in S$ and action $a \in \mathcal{A}(s)$, for a total of at most MN problems. Each problem has at most N unknowns, representing the probability of transitioning from state s to state s' for $s' \in S^{yes}$. Moreover, it has $N + 1$ constraints to guarantee that the solution lies in the probability simplex, and D_s^a constraints to enforce the solution to be within the uncertainty set $\mathcal{F}_s^a$. According to the definition in Section 2.1, the total number of unknowns and constraints for each convex program is thus linear in $\mathcal{R}$. Using Proposition 4.2, each inner problem is solved in time polynomial in $\mathcal{R}$. Soundness and completeness also follow directly from Proposition 4.2, which states that the optimum of Problem (4.13) can be found to within the desired accuracy $\pm\epsilon_d$ in time linear in $\log(1/\epsilon_d)$. □

We verify $\phi = \boldsymbol{P}_{\leq 0.4}[\boldsymbol{\mathcal{X}}\ \omega]$ for the Convex-MDP shown in Figure 2.3. Trivially, $S^{yes} = \{s_2\}$. Setting up the inner problem for state s_0 and action a, we get:

$$P_{s_0}^{a,max} = \max_{f_{01}, f_{02}} f_{02}$$
$$\text{s.t. } 0.6 \leq f_{01} \leq 0.8;\ 0.2 \leq f_{02} \leq 0.5;\ f_{01} + f_{02} = 1$$

with solution $P_{s_0}^{a,max}[\mathcal{X}\omega] = 0.4$. Repeating $\forall a \in A, \forall s \in S$, we get $\mathbf{P}^{max}[\boldsymbol{\mathcal{X}}\omega] = [0.4, 0.5, 0, 0.6]$, so $Sat(\phi) = \{s_0, s_2\}$.

4.2.2 *Bounded Until* Operator

The *Bounded Until* operator $\phi_1\ \boldsymbol{\mathcal{U}}^{\leq \boldsymbol{k}}\ \phi_2$ computes the states from which it is possible to reach a state satisfying ϕ_2 within k execution steps while visiting only states satisfying ϕ_1. For example, if the Convex-MDP models a hand-shaking protocol, the *Bounded Until* operator might be used to verify that the "acknowledgment" packet gets sent within 3 steps of execution, while visiting only states in which the correctness of the received packet is checked (e.g., parity check, packet origin, etc.).

Formally, properties for this operator are expressed as:

$$\phi = \boldsymbol{P}_{\lhd p}[\phi_1\ \boldsymbol{\mathcal{U}}^{\leq \boldsymbol{k}}\ \phi_2]$$

Algorithm 4.2: Bounded Until Operator.

1: **Input** $\mathcal{M}_{\mathcal{C}} = (S, S_0, A, \Omega, \mathcal{A}, \mathcal{F}, L)$, $\phi = \boldsymbol{P}_{\triangleleft p}[\phi_1 \, \boldsymbol{\mathcal{U}}^{\leq \boldsymbol{k}} \, \phi_2]$
2: **Output** The set $y = Sat(\phi)$
3: Compute S^{yes}, S^{no} and $S^{?}$
4: **for** i = 1 : k **do**
5: **for all** $s \in S^{?}$ **do**
6: $x_s^i = \max_{a \in \mathcal{A}(s)} \max_{\mathbf{f}_s^a \in \mathcal{F}_s^a} \sum_{s'} f_{ss'}^a x_{s'}^{i-1}$
7: **end for**
8: **end for**
9: $\mathbf{P}^{max}[\phi_1 \, \boldsymbol{\mathcal{U}}^{\leq \boldsymbol{k}} \, \phi_2] = \mathbf{x}^k$
10: $y = \{s \in S \mid P_s^{max} \triangleleft p\}$

The pseudocode of the model-checking routine for the *Bounded Until* operator is shown in Algorithm 4.2. First, the sets:

$$
\begin{aligned}
S^{yes} &\stackrel{def}{=} Sat(\phi_2) \\
S^{no} &\stackrel{def}{=} S \setminus (Sat(\phi_1) \cup Sat(\phi_2)) \\
S^{?} &= S \setminus (S^{no} \cup S^{yes})
\end{aligned}
$$

are precomputed using classical reachability routines over the Convex-MDP underlying graph [60] (line 3). As explained in Section 4.1.2, Markov adversaries are not sufficient to maximize the satisfaction probability $\mathbf{P}^{max}[\phi_1 \, \boldsymbol{\mathcal{U}}^{\leq \boldsymbol{k}} \, \phi_2]$, so we need to unroll the execution of the Convex-MDP for k steps. This is done at lines $4-9$, where the maximum satisfaction probabilities $\mathbf{P}^{max}[\phi_1 \, \boldsymbol{\mathcal{U}}^{\leq \boldsymbol{k}} \, \phi_2] = \mathbf{x}^k = G^k(\mathbf{0})$ to satisfy ϕ are computed for all states $s \in S$ applying k times mapping G defined as:

$$
\mathbf{x}^i = G^i(\mathbf{x}^{i-1}) = \begin{cases} 0; \; 1; & \forall s \in S^{no}; \; \forall s \in S^{yes}; \\ 0; & \forall s \in S^{?} \wedge i = 0; \\ \max\limits_{a \in \mathcal{A}(s)} \max\limits_{\mathbf{f}_s^a \in \mathcal{F}_s^a} \sum\limits_{s'} f_{ss'}^a x_{s'}^{i-1} & \forall s \in S^{?} \wedge i > 0 \end{cases} \tag{4.14}
$$

and $\mathbf{x}^{-1} = \mathbf{0} \in \mathbb{R}^N$. Finally, the computed probabilities are compared to the threshold p to select the states that satisfy ϕ (line 10).

Remark 4.3. *The sets $\mathcal{F}_s^a$ can be expressed with models of uncertainty different from one another $\forall s \in S, \forall a \in \mathcal{A}(s)$, since each optimization problem in Mapping (4.14) gets solved independently from the others.*

Lemma 4.2. ***Model Checking of the*** **Bounded Until** ***Operator.*** *The routine to verify the* Bounded Until *operator is sound, complete and guaranteed to terminate with algorithmic complexity that is polynomial in the size $\mathcal{R}$ of $\mathcal{M}_C$ and pseudo-polynomial in the time horizon k of $\boldsymbol{\mathcal{U}}^{\leq \boldsymbol{k}}$.*

Proof. For the first part, the proof is similar to the one for the *Next* Operator. The sets S^{yes}, S^{no} and $S^{?}$ are precomputed in time polynomial in $\mathcal{R}$ using conventional reachability routines over the Convex-MDP underlying graph [60]. To apply Mapping (4.14), we need to solve $O(MQ)$ "inner" convex programs with $Q = |S^{?}|$, i.e., one $\forall s \in S^{?}, \forall a \in \mathcal{A}(s)$. Each problem has at most N unknowns, $N + 1 + D_s^a$ constraints, and it is solved in time polynomial in $\mathcal{R}$. Further, the pseudo-polynomial complexity in k comes from applying Mapping (4.14) k times.

We now study how the numerical errors intrinsic to the solution of a convex program propagate across the k iterations of the algorithm. By Proposition 4.2, we can state that each inner problem can be solved with accuracy $\pm\epsilon_{inn}$ in time linear in $\log(1/\epsilon_{inn})$. On the other hand, we are left to prove the soundness and completeness of the overall solution, since the ϵ_{inn}-approximations in computing $\mathbf{x}^i, \forall i$ get propagated at each iteration, and might in principle result in a larger error at the end of the procedure. We call ϵ_s^i the error accumulated at step i for state s, $x_s^i = x_{s,id}^i + \epsilon_s^i$, where $x_{s,id}^i$ is the solution with no error propagation, and ϵ_s^k the error in the final solution. Also, $\mathbf{f}_s^{a,i} \in \mathcal{F}_s^a$ is the optimal solution of the inner problem at step i. We solve this difficulty by noting that the optimal value of the the inner problem is computed by multiplying vector $\mathbf{x}^i$ by $\mathbf{f}_s^{a,i} \in \mathcal{F}_s^a$, with $\mathbf{1}^T \mathbf{f}_s^a = 1, \forall \mathbf{f}_s^a \in \mathcal{F}_s^a, \forall a \in \mathcal{A}(s)$. At the first, second and i^{th} iteration:

$$
\begin{aligned}
x_s^1 &= x_{s,id}^1 + \epsilon_s^1 = \mathbf{f}_s^{a,1}\mathbf{x}^0 + \epsilon_{inn} \\
x_s^2 &= \mathbf{f}_s^{a,2}\mathbf{x}^1 + \epsilon_{inn} = \mathbf{f}_s^{a,2}\left(F^{a,1}\mathbf{x}^0 + \epsilon_{inn}\mathbf{1}\right) + \epsilon_{inn} = \mathbf{f}_s^{a,2}F^{a,1}\mathbf{x}^0 + 2\epsilon_{inn} \\
x_s^i &= \mathbf{f}_s^{a,i}\mathbf{x}^{i-1} + \epsilon_{inn} = \mathbf{f}_s^{a,i}\left(F^{a,i-1}\mathbf{x}^{i-1} + (i-1)\epsilon_{inn}\mathbf{1}\right) + \epsilon_{inn} = \mathbf{f}_s^{a,i}F^{a,i-1}\ldots F^{a,1}\mathbf{x}^0 + i\epsilon_{inn}
\end{aligned}
$$

so ϵ_s^i increases linearly with i. The desired accuracy ϵ_d of the final solution can thus be guaranteed by solving each inner problem with accuracy $\epsilon_{inn} = \epsilon_d/k$. □

We verify property $\phi = \boldsymbol{P}_{\leq 0.6}[\vartheta\, \boldsymbol{\mathcal{U}}^{\leq \mathbf{1}}\omega]$ for the Convex-MDP shown in Figure 2.3. First, we compute the sets $S^{yes} = \{s_2\}$, $S^{no} = \{s_1\}$ and $S^{?} = \{s_0, s_3\}$. By applying Mapping (4.14) once, we obtain $\mathbf{P}^{max}[\vartheta\, \boldsymbol{\mathcal{U}}^{\leq \mathbf{1}}\omega] = [0.4, 0, 1, 0.6]$ and $Sat(\phi) = \{s_0, s_1, s_3\}$.

4.2.3 *Unbounded Until* Operator

The *Unbounded Until* operator $\phi_1\, \boldsymbol{\mathcal{U}}\, \phi_2$ computes the states from which it is possible to eventually reach a state satisfying ϕ_2 while visiting only states satisfying ϕ_1. For example, if the Convex-MDP models an electric power system with a backup unit, the *Unbounded Until* operator might be used to verify that all the states in which the main power generator fails can be reached only by visiting states in which the backup unit is operational, so that it can then be possible to switch to the backup to maintain functionality of the overall system.

Formally, properties for this operator are expressed as:

$$\phi = \boldsymbol{P}_{\lhd p}[\phi_1\, \boldsymbol{\mathcal{U}}\, \phi_2]$$

Algorithm 4.3: Unbounded Until Operator.

1: **Input** $\mathcal{M}_{\mathcal{C}} = (S, S_0, A, \Omega, \mathcal{A}, \mathcal{F}, L)$, $\phi = \boldsymbol{P}_{\lhd p}[\phi_1 \, \boldsymbol{\mathcal{U}} \, \phi_2]$
2: **Output** The set $y = Sat(\phi)$
3: Compute S^{yes}, S^{no} and $S^{?}$
4: $\mathbf{P}^{max}[\phi_1 \, \boldsymbol{\mathcal{U}} \, \phi_2] =$ ComputeProbabilities()
5: $y = \{s \in S \mid P_s^{max} \lhd p\}$

The pseudocode of the model-checking routine for the *Unbounded Until* operator is shown in Algorithm 4.3. First, the sets

$$
\begin{aligned}
S^{yes} &\stackrel{def}{=} Sat(\phi_2) \\
S^{no} &\stackrel{def}{=} S \setminus (Sat(\phi_1) \cup Sat(\phi_2)) \\
S^{?} &= S \setminus (S^{no} \cup S^{yes})
\end{aligned}
$$

are precomputed using classical reachability routines over the Convex-MDP underlying graph [60] (line 3). Second, the maximum probability to satisfy ϕ, as defined in Equation (4.9), is computed for all states $s \in S$ by calling the procedure "ComputeProbabilities()" (line 4). Finally, the computed satisfaction probabilities are compared to the threshold p to determine the satisfiability set $Sat(\phi)$ (line 5).

In the rest of the section, we propose two implementations of the procedure "ComputeProbabilities()", both returning the exact solution modulo rounding errors due to the machine finite resolution. The first implementation is based on Convex Programming (CP), while the second is based on Value Iteration (VI). We report both implementations since they differ in applicability and runtime performance. The CP procedure has algorithmic complexity provably polynomial in $\mathcal{R}$, but it can only be applied to Convex-MDPs that satisfy Assumption 2.4. On the other hand, the VI procedure can be applied to any convex model of uncertainty, but its algorithmic complexity depends not only on the problem size but also on the problem data. In general, the user is invited to use both procedures for the model checking of the problem at hand and select the one that performs better for the specific scenario of interest.

4.2.3.1 Convex Programming Procedure (CP)

We start from the classical Linear Programming (LP) formulation to solve the problem without the presence of uncertainty [60]:

$$\begin{aligned} &\min_{\mathbf{x}} \mathbf{x^T 1} \\ &\text{s.t. } x_s = 0; && \forall s \in S^{no}; \\ &\quad x_s = 1; && \forall s \in S^{yes}; \\ &\quad x_s \geq \mathbf{x^T f_s^a} && \forall s \in S^{?}, \forall a \in \mathcal{A}(s) \end{aligned} \tag{4.15}$$

where $\mathbf{P}^{max}[\phi_1 \, \boldsymbol{\mathcal{U}} \, \phi_2] = \mathbf{x}^*$ is computed solving only one LP. Intuitively, Problem (4.15) selects the optimal (in the worst-case sense) adversary by selecting the highest upper bound on the values of the satisfaction probabilities x_s across the actions $a \in \mathcal{A}(s)$ available at each state $s \in S^?$. Problem (4.15) has N unknowns and $N - Q + MQ$ constraints, where $Q = |S^?| = O(N)$, so its size is polynomial in $\mathcal{R}$.

Since the *Unbounded Until* operator admits optimal Markov deterministic adversaries and natures, as proved in Proposition 4.1, we can rewrite Problem (4.15) when uncertainties are added to the model as:

$$\begin{aligned} &\min_{\mathbf{x}} \mathbf{x^T 1} \\ &\text{s.t. } x_s = 0; && \forall s \in S^{no}; \\ &\quad x_s = 1; && \forall s \in S^{yes}; \\ &\quad x_s \geq \max_{\mathbf{f_s^a} \in \mathcal{F}_s^a} (\mathbf{x^T f_s^a}) && \forall s \in S^{?}, \forall a \in \mathcal{A}(s) \end{aligned} \tag{4.16}$$

i.e., we further maximize the lower bound on x_s across the action range of the adversarial nature. The decision variable of the inner problem is $\mathbf{f}_s^a$ and its optimal value $\nu^*(\mathbf{x})$ is parametrized in the outer problem decision variable $\mathbf{x}$. Problem (4.16) can be written in convex form for an arbitrary uncertainty model by replacing the last constraint with a set of constraints, one for each point in $\mathcal{F}_s^a$. However, this approach results in infinite constraints if the set $\mathcal{F}_s^a$ contains infinitely many points, as in the cases considered in this work, making the problem unsolvable. We solve this difficulty using duality, which allows to rewrite Problem (4.16) with a number of additional constraints only polynomial in the size $\mathcal{R}$ of the Convex-MDP. For each state $s \in S^?$ and action $a \in \mathcal{A}(s)$, we replace the primal inner problem in the outer Problem (4.16) with its dual:

$$\nu_s^a(\mathbf{x}) = \max_{\mathbf{f}_s^a \in \mathcal{F}_s^a} \mathbf{x}^T \mathbf{f}_s^a \qquad \Rightarrow \qquad d_s^a(\mathbf{x}) = \min_{\boldsymbol{\lambda}_s^a \in \mathcal{D}_s^a} g(\boldsymbol{\lambda}_s^a, \mathbf{x}) \tag{4.17}$$

where $\boldsymbol{\lambda}_s^a$ is the (vector) Lagrange multiplier and $\mathcal{D}_s^a$ is the feasibility set of the dual problem. In the dual problem, the decision variable is $\boldsymbol{\lambda}_s^a$ and its optimal value $d_s^a(\mathbf{x})$ is again parametrized in the outer problem decision variable $\mathbf{x}$. The dual function $g(\boldsymbol{\lambda}_s^a, \mathbf{x})$ and the set $\mathcal{D}_s^a$ are convex by construction in $\boldsymbol{\lambda}_s^a$ for arbitrary uncertainty models, so the dual problem is convex. Further, since

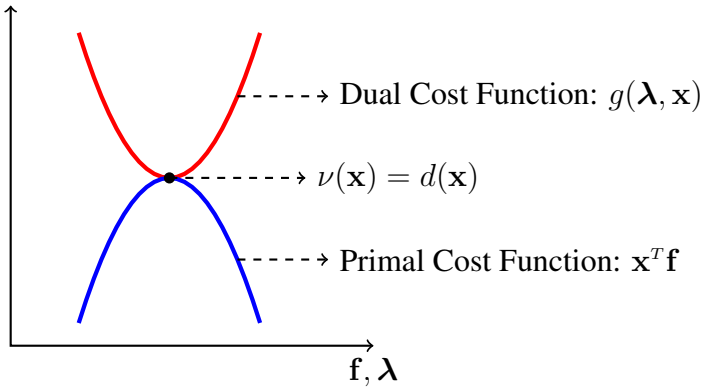


Figure 4.2: Graphical representation of the primal-dual transformation introduced in Equation 4.17. The optimal values of the primal $\nu(\mathbf{x})$ and dual $d(\mathbf{x})$ problems coincide because strong duality holds. Moreover, the dual cost function $g(\boldsymbol{\lambda}, \mathbf{x})$ always overestimates the primal optimal value $\nu(\mathbf{x})$. We dropped the superscript a and subscript s for clarity.

also the primal problem is convex, strong duality holds, i.e., $\nu_s^a = d_s^a$, $\forall\ \mathbf{x} \in \mathbb{R}^N$, because the primal problem satisfies Slater's condition [30] for any non-trivial uncertainty set $\mathcal{F}_s^a$.

As shown in Figure 4.2, the cost function $g(\boldsymbol{\lambda}_s^a, \mathbf{x})$ of the dual problem always overestimates the primal optimal value $\nu_s^a(\mathbf{x})$ for any value of the dual variable $\boldsymbol{\lambda}_s^a$. When substituting the primal problems with the dual problems in Problem (4.16), we can thus drop the inner optimization operators because the outer optimization operator will nevertheless aim to find the least overestimates, i.e., the dual optimal values $d_s^a, \forall s \in S, a \in \mathcal{A}(s)$, to minimize its own cost function. We get the CP formulation:

$$
\begin{aligned}
&\min_{\mathbf{x}} \mathbf{x}^\mathbf{T}\mathbf{1} && && \min_{\mathbf{x},\boldsymbol{\lambda}} \mathbf{x}^\mathbf{T}\mathbf{1} && \\
&\text{s.t. } x_s = 0; && && \text{s.t. } x_s = 0; && \forall s \in S^{no}; && \text{(4.18a)}\\
&\quad x_s = 1; && \Rightarrow && \quad x_s = 1; && \forall s \in S^{yes}; && \text{(4.18b)}\\
&\quad x_s \geq \min_{\boldsymbol{\lambda}_s^a \in \mathcal{D}_s^a} g\left(\boldsymbol{\lambda}_s^a, \mathbf{x}\right) && && \quad x_s \geq g\left(\boldsymbol{\lambda}_s^a, \mathbf{x}\right); && \forall s \in S^?, \forall a \in \mathcal{A}(s); && \text{(4.18c)}\\
& && && \boldsymbol{\lambda}_s^a \in \mathcal{D}_s^a && \forall s \in S^?, \forall a \in \mathcal{A}(s) && \text{(4.18d)}
\end{aligned}
$$

The decision variables of Problem (4.18) are both $\mathbf{x}$ and $\boldsymbol{\lambda}_s^a$, so the CP formulation is convex only if the dual function $g(\boldsymbol{\lambda}_s^a, \mathbf{x})$ is jointly convex in $\boldsymbol{\lambda}_s^a$ and $\mathbf{x}$. While this condition cannot be guaranteed for arbitrary uncertainty models, we prove constructively that it holds for the ones considered in

the dissertation. For example, for the interval model, Problem (4.18) reads:

$$
\begin{aligned}
&\min_{\mathbf{x},\boldsymbol{\lambda}_s^{\boldsymbol{a}}} \mathbf{x^T 1} \\
&\text{s.t. } x_s = 0; && \forall s \in S^{no}; \\
&\quad x_s = 1; && \forall s \in S^{yes}; \\
&\quad x_s \geq \lambda_{1,s}^a - (\underline{\mathbf{f}}_a^s)^T \boldsymbol{\lambda}_{\mathbf{2},\boldsymbol{s}}^{\boldsymbol{a}} + (\overline{\mathbf{f}}_a^s)^T \boldsymbol{\lambda}_{\mathbf{3},\boldsymbol{s}}^{\boldsymbol{a}}; && \forall s \in S^?, \forall a \in \mathcal{A}(s); \\
&\quad \mathbf{x} + \boldsymbol{\lambda}_{\mathbf{2},\boldsymbol{s}}^{\boldsymbol{a}} - \boldsymbol{\lambda}_{\mathbf{3},\boldsymbol{s}}^{\boldsymbol{a}} - \lambda_{1,s}^a \mathbf{1} = \mathbf{0}; && \forall s \in S^?, \forall a \in \mathcal{A}(s); \\
&\quad \boldsymbol{\lambda}_{\mathbf{2},\boldsymbol{s}}^{\boldsymbol{a}} \geq \mathbf{0},\ \boldsymbol{\lambda}_{\mathbf{3},\boldsymbol{s}}^{\boldsymbol{a}} \geq \mathbf{0} && \forall s \in S^?, \forall a \in \mathcal{A}(s)
\end{aligned}
\tag{4.19}
$$

with $\boldsymbol{\lambda}_{\boldsymbol{s}}^{\boldsymbol{a}} = [\lambda_{1,s}^a, \boldsymbol{\lambda}_{\mathbf{2},\boldsymbol{s}}^{\boldsymbol{a}}, \boldsymbol{\lambda}_{\mathbf{3},\boldsymbol{s}}^{\boldsymbol{a}}]$. Problem (4.19) is an LP, so trivially jointly convex in $\mathbf{x}$ and $\boldsymbol{\lambda}_s^a$.

Analogously, Problem (4.18) for the ellipsoidal model becomes:

$$
\begin{aligned}
&\min_{x_s,\boldsymbol{\lambda}_s^{\boldsymbol{a}}} \mathbf{x_s^T 1} \\
&\text{s.t. } x_s = 0 && \forall s \in S^{no} \\
&\quad x_s = 1 && \forall s \in S^{yes} \\
&\quad x_s \geq \lambda_{1,s}^a + \lambda_{2,s}^a + \mathbf{h_s^{aT}} E_s^a \lambda_{\mathbf{3,s}}^{\mathbf{a}} && \forall s \in S^?, \forall a \in A \\
&\quad \|\boldsymbol{\lambda}_{\mathbf{3},\boldsymbol{s}}^{\boldsymbol{a}}\|_2 \leq \lambda_{2,s}^a && \forall s \in S^?, \forall a \in A \\
&\quad \mathbf{x} - \lambda_{1,s}^a \mathbf{1} - E_s^{aT} \boldsymbol{\lambda}_{\mathbf{3},\boldsymbol{s}}^{\boldsymbol{a}} = \mathbf{0} && \forall s \in S^?, \forall a \in A
\end{aligned}
\tag{4.20}
$$

which is a Second-Order Cone Program (SOCP), so again trivially jointly-convex in $\mathbf{x}$ and $\boldsymbol{\lambda}_s^a$.

For the likelihood model, Problem (4.18) reads:

$$
\begin{aligned}
&\min_{x_s,\boldsymbol{\lambda}_s^{\boldsymbol{a}}} \mathbf{x_s^T 1} \\
&\text{s.t. } x_s = 0; && \forall s \in S^{no}; && \text{(4.21a)} \\
&\quad x_s = 1; && \forall s \in S^{yes}; && \text{(4.21b)} \\
&\quad x_s \geq \lambda_{1,s}^a - (1+\beta_s^a)\lambda_{2,s}^a + \lambda_{2,s}^a \textstyle\sum_{s'} h_{ss'}^a \log\left(\frac{\lambda_{2,s}^a h_{ss'}^a}{\lambda_{1,s}^a - x_{s'}}\right); && \forall s \in S^?, \forall a \in \mathcal{A}(s); && \text{(4.21c)} \\
&\quad \lambda_{1,s}^a \geq \max_{s' \in S} x_{s'}; && \forall s \in S^?, \forall a \in \mathcal{A}(s) && \text{(4.21d)} \\
&\quad \lambda_{2,s}^a \geq 0 && \forall s \in S^?, \forall a \in \mathcal{A}(s) && \text{(4.21e)}
\end{aligned}
$$

We prove its joint convexity in $\mathbf{x}$ and $\boldsymbol{\lambda}_s^a$ as follows. The cost function and Constraints (4.21a), (4.21b), (4.21d) and (4.21e) are trivially convex. Constraint (4.21c) is generated by a primal-dual transformation, so, according to convex theory, it is convex in the dual variables $\boldsymbol{\lambda}_s^a$ by construction. Moreover, convex theory also guarantees that the affine subtraction of $\mathbf{x}$ from $\boldsymbol{\lambda}_s^a$ preserves convexity under the condition enforced by Constraint (4.21d), so we conclude that Problem (4.21) is convex.

Finally, for the entropy model, Problem (4.18) can be written as:

$$\begin{aligned}
&\min_{x_s, \boldsymbol{\lambda}_s^a} \ \mathbf{x_s^T 1} \\
&\text{s.t. } x_s = 0; && \forall s \in S^{no}; && \text{(4.22a)}\\
&\quad x_s = 1; && \forall s \in S^{yes}; && \text{(4.22b)}\\
&\quad x_s \geq \lambda_{1,s}^a \log\left(\textstyle\sum_{s'} q_{ss'} \exp\left(\frac{x_{s'}}{\lambda_{1,s}^a}\right)\right) + \beta_s \lambda_{1,s}^a; && \forall s \in S^?, \forall a \in \mathcal{A}(s); && \text{(4.22c)}\\
&\quad \boldsymbol{\lambda_s^a} \geq \mathbf{0} && \forall s \in S^?, \forall a \in \mathcal{A}(s) && \text{(4.22d)}
\end{aligned}$$

We prove its joint convexity in $\mathbf{x}$ and $\boldsymbol{\lambda}_s^a$ as follows. The cost function and Constraints (4.22a), (4.22b) and (4.22d) are trivially convex. Constraint (4.22c) is generated by a primal-dual transformation, so, according to convex theory, it is convex in the dual variables $\boldsymbol{\lambda}_s^a$ by construction. Moreover, we prove that it is also jointly convex in $\mathbf{x}$ by induction on the number NS of next states $s' \in S$ for state s. As a base case, $NS = 1$, and we can rewrite the constraint as:

$$x_s \geq \lambda_{1,s}^a \log\left(q_{ss'}\right) + x_{s'} + \beta_s \lambda_{1,s}^a$$

which is affine in $\boldsymbol{\lambda}_s^a$ and $\mathbf{x}$, so trivially jointly convex. We now assume that the constraint is jointly convex for $NS = n$, and prove that it is jointly convex also for $NS = n + 1$. This result immediately follows from observing that increasing NS simply introduces one more term in the summation, so if the constraint is jointly convex for $NS = n$ it must remain jointly convex also for $NS = n + 1$, since an affine addition preserves convexity according to convex theory. We conclude that Problem (4.22) is convex.

For general Convex-MDPs, we repeat here Assumption 2.4:

Assumption 4.1. ***Joint-Convexity.*** *Given a Convex-MDP $\mathcal{M}_\mathcal{C}$, for all convex uncertainty sets $\mathcal{F}_s^a \in \mathcal{F}$, the dual function $g(\boldsymbol{\lambda}_s^a, \mathbf{x})$ in Equation (4.17) is jointly-convex in both $\boldsymbol{\lambda}_s^a$ and $\mathbf{x}$.*

According to Proposition 4.2, Problem (4.18) can thus be solved in polynomial time. Also for this formulation, $\mathbf{P}^{max}[\phi_1 \ \boldsymbol{\mathcal{U}} \ \phi_2] = \mathbf{x}^*$, so all the satisfaction probabilities can be computed by solving only one convex problem.

Remark 4.4. *We note that we can combine models of uncertainty different from one another within a single CP formulation, since each instance of the dual problem in Equation (4.17) is independent from the others. Moreover, according to Assumption 2.1, all uncertainty sets $\mathcal{F}_s^a \in \mathcal{F}, \ \forall s \in S, \forall a \in \mathcal{A}(s)$ are independent from one another, so the convexity of the CP formulation is preserved. As an example, if both the interval and ellipsoidal models are used, the overall CP formulation is an SOCP.*

We can now summarize the results of this section in the following Lemma.

Lemma 4.3. ***Model Checking of the*** **Unbounded Until** ***Operator Using the CP Procedure.*** *The CP procedure to verify the* Unbounded Until *operator is sound, complete and guaranteed to terminate with algorithmic complexity polynomial in the size $\mathcal{R}$ of $\mathcal{M}_\mathcal{C}$, if $\mathcal{M}_\mathcal{C}$ satisfies Assumption 4.1.*

Proof. The CP procedure solves only one convex program to compute the satisfaction probabilities $\mathbf{P}^{max}[\phi_1 \ \boldsymbol{\mathcal{U}} \ \phi_2] = \mathbf{x}^*$ of all states $s \in S$. This convex program gets generated in time polynomial in $\mathcal{R}$ as follows. We formulate Constraints (4.18c) and (4.18d) for all $s \in S^?$ and $a \in \mathcal{A}(s)$, i.e., $O(MQ)$ constraints, where $Q = |S^?| = O(N)$. They are derived from MQ primal-dual transformations as in Equation (4.17). Each primal inner problem has N unknowns, $N+1$ constraints to represent the probability simplex and D_s^a constraints to represent the uncertainty set $\mathcal{F}_s^a$. From duality theory, the corresponding dual inner problem has $N+1+D_s^a$ unknowns and $2N+1+D_s^a$ constraints. Overall, Problem (4.18) has $O\left((N+1+D)MQ\right)$ more unknowns and $O\left((2N+1+D)MQ\right)$ more constraints of Problem (4.15), so its size is still polynomial in $\mathcal{R}$. Further, if $\mathcal{M}_\mathcal{C}$ satisfies Assumption 4.1, Problem (4.18) is convex. Using Lemma 4.2, we then conclude that it can be solved in time polynomial in $\mathcal{R}$. Finally, when strong duality holds for the transformation in Equation (4.17), soundness and completeness of the final solution are preserved because the dual and primal optimal value of each inner problem are equivalent. □

We verify $\phi = \boldsymbol{P}_{\geq 0.4}[\ \vartheta \ \boldsymbol{\mathcal{U}} \ \omega\ ]$ on the Convex-MDP shown in Figure 2.3. Problem (4.18) written with the data of the model has 19 variables and 11 constraints (attached in Appendix A). The solution reads: $\mathbf{P}^{min}[\ \vartheta \ \boldsymbol{\mathcal{U}} \ \omega\ ] = [0.2, 0, 1, 0.46]$, and, in conclusion, $Sat(\phi) = \{s_2, s_3\}$.

4.2.3.2 Value Iteration Procedure (VI)

In this section, we present an alternative procedure for the model checking of the *Unbounded Until* operator based on Value Iteration (VI). While the CP procedure formulates a single convex program to compute the satisfaction probabilities all at once, the VI procedure brakes the overall problem into multiple subproblems and iterates through them until it reaches the solution. Intuitively, the procedure runs the iteration introduced for the *Bounded Until* operator (Mapping (4.14)) until the desired level of accuracy ϵ_d in the estimation of the satisfaction probabilities $\mathbf{P}^{max}[\phi_1 \ \boldsymbol{\mathcal{U}} \ \phi_2]$ is achieved. In order to prove the soundness and correctness of the procedure, we will first show that the procedure converges to the exact solution in the limit of infinite iterations. We will then discuss exiting conditions to achieve the desired accuracy ϵ_d in the solution and how to handle error propagation across iterations.

To prove the soundness and completeness of the procedure, we will use the following results:

Definition 4.2. ***Contraction.*** *Let (B, d) be a metric space and $g : B \to B$ a function. Function g is a* contraction *if there is a real number θ such that $0 \leq \theta < 1$ and*

$$d\left(g(u), g(v)\right) \leq \theta d(u, v) \quad \forall u, v \in B$$

Proposition 4.3. ***Contraction Mapping.*** *Let (B, d) be a complete metric space and $g : B \to B$ a contraction. Then, for $x \in B$ there exists a unique point $x^* \in B$ such that:*

$$g(x^*) = x^* \qquad \textit{and} \qquad \lim_{k \to +\infty} g^k(x) = x^*$$

In particular, we use the same mapping $g = G$ defined in Mapping (4.14) and repeated here for convenience:

$$\mathbf{x}^i = G^i(\mathbf{x}^{i-1}) = \begin{cases} 0;\ 1; & \forall s \in S^{no};\ \forall s \in S^{yes}; \\ 0; & \forall s \in S^{?} \wedge i = 0; \\ \max\limits_{a \in \mathcal{A}(s)} \max\limits_{\mathbf{f}_s^a \in \mathcal{F}_s^a} \sum\limits_{s'} f_{ss'}^a x_{s'}^{i-1} & \forall s \in S^{?} \wedge i > 0 \end{cases} \tag{4.23}$$

with the sets S^{no}, S^{yes} and $S^{?}$ defined in Section 4.2.3.

To evaluate how distant from the exact solution the estimate at the current iteration is, we use the infinity norm $\| \ . \ \|_\infty$ of a vector $v \in \mathbb{R}^N$.

Definition 4.3. ***Infinity Norm.*** *Given a vector $v \in \mathbb{R}^N$, the infinity norm of v is defined as:*

$$\| \ . \ \|_\infty = max\left(\mid v_1 \mid, \cdots, \mid v_N \mid\right)$$

We can now prove that:

Proposition 4.4. *Mapping G is a contraction over the metric space $(\mathbb{R}^N, \| \ . \ \|_\infty)$.*

Sketch of proof. The proof is based on results presented in [15, 24, 172] adapted to account for uncertainties in the model. The full derivation and the constructive steps to compute θ from the Convex-MDP data can be found in Appendix B. The value of θ is derived in Equation (B.4). □

We now state the main results of this section:

Lemma 4.4. ***Model Checking of the*** **Unbounded Until** ***Operator Using the VI Procedure.*** *The VI procedure to verify the* Unbounded Until *operator is sound and complete, i.e.,*

$$\mathbf{P}^{max}[\phi_1 \ \boldsymbol{\mathcal{U}} \ \phi_2] = \lim_{k \to +\infty} G^k(\mathbf{x}) \tag{4.24}$$

Proof. To prove soundness and completeness, we prove that the fixed point x^* computed by repeatedly applying Mapping (4.23) converges to an optimal adversary and nature. By contradiction, if x^* was not optimal, the optimization problems in Mapping (4.23) would find a new optimal point, with no need to keep track of the iteration history because Proposition 4.1 states that there exists a Markov optimal adversary and nature. However, this would contradict Lemma 4.4, which states that Mapping (4.23) is a contraction. We conclude that x^* is optimal. □

Remark 4.5. *The VI routine can handle arbitrary models of uncertainty, in particular also ones that do not satisfy Assumption 4.1, because the inner convex optimization problem in Mapping (4.23) computes the optimal adversarial natures at iteration i, $\mathbf{f}_s^{a,i}$, using the values of the satisfaction probabilities $\mathbf{x}^{i-1}$ computed in the previous iteration. As a consequence, we do not need to require joint-convexity in $\mathbf{f}_s^a$ and $\mathbf{x}$ to solve the optimization problem.*

The higher flexibility in handling diverse models of uncertainty comes at a cost. In fact, we will prove next that the required number of iterations K to compute the satisfaction probabilities $\mathbf{P}^{max}[\phi_1 \,\boldsymbol{\mathcal{U}}\, \phi_2] = \mathbf{x}^K$ within the desired accuracy $\pm\epsilon_d$ depends not only on the problem size $\mathcal{R}$ but also on the problem data, i.e., the numerical values of the estimated transition probabilities.

We start by noting that the result in Proposition 4.4 allows us to bound the error in the estimation of $\mathbf{P}^{max}[\phi_1 \,\boldsymbol{\mathcal{U}}\, \phi_2]$ at the end of the i^{th} iteration by:

$$\epsilon_i = \|\mathbf{P}^{max}[\phi_1 \,\boldsymbol{\mathcal{U}}\, \phi_2] - \mathbf{x}^i\|_\infty \leq \epsilon_0 \frac{\theta^i}{1-\theta}$$

where ϵ_0 is the initial error in the estimation which can be trivially bounded by $\epsilon_0 \leq 1$. We can thus obtain a sufficient condition to guarantee that the accuracy in the solution is bounded by $\pm\epsilon_d$:

$$\epsilon_i \leq \epsilon_d \quad \Rightarrow \quad \mid P_s^{max}[\phi_1 \,\boldsymbol{\mathcal{U}}\, \phi_2] - x^i \mid \leq \epsilon_d, \;\; \forall s \in S^?$$

The resulting bound K on the number of iterations is:

$$\frac{\theta^K}{1-\theta} \leq \epsilon_d \quad \Rightarrow \quad K \geq \frac{log[\epsilon_d(1-\theta)]}{log(\theta)} \tag{4.25}$$

Since the contraction parameter θ depends on the model data, as derived in Appendix B (Equation (B.4)), we conclude that also the number of iterations K required to converge to the desired accuracy ϵ_d varies in general with the model data.

In a practical application, the computation of the contraction parameter θ and the corresponding required number of iterations K is quite sensitive to numerical errors, since the evaluation of Equation (4.25) involves the computation of the ratio of the logarithms of two small numbers. As a consequence, in our software implementation of the VI procedure, we used a different stopping criterion based on relative tolerance. In particular, the procedure stops the iteration of Mapping (4.23) when:

$$\delta_r > \max_{s \in S^?} \left(\frac{|x_s^i - x_s^{i-1}|}{x_s^i} \right) \tag{4.26}$$

The required δ_r to achieve accuracy ϵ_d, i.e., δ_r^*, depends on the Convex-MDP model and is determined by trial-and-error, a common practice in iterative procedures (e.g., the ODE solver in a circuit simulator). To determine δ_r^*, we compute several approximations of $P^{max}[\phi_1 \,\boldsymbol{\mathcal{U}}\, \phi_2]$ while decreasing δ_r by steps of $10\times$. We heuristically stop when no probability $P_s^{max}[\phi_1 \,\boldsymbol{\mathcal{U}}\, \phi_2], \forall s \in S^?$, changes more than ϵ_d after checking Criterion (4.26) for δ_r^* and $\delta_r^*/100$. Finally, errors in solving the inner problems, as introduced in Section 4.2.2, are propagated across iterations. We call ϵ_{inn} the accuracy in solving one iteration of the inner problem. If the VI procedure exits after K iterations and $\epsilon_d < K \times \epsilon_{inn}$, the procedure needs to be run again after decreasing ϵ_{inn} to, approximately, $\epsilon_{inn} < \epsilon_d/K$.

We use the VI routine with $\epsilon_d = 10^{-3}$ to verify again $\phi = P_{\geq 0.4}[\vartheta \,\boldsymbol{\mathcal{U}}\, \omega]$ in the example in Figure 2.3. After 5 iterations, we get $\mathbf{P}^{min}[\vartheta \,\boldsymbol{\mathcal{U}}\, \omega] = [0.2, 0, 1, 0.46]$ and $Sat(\phi) = \{s_2, s_3\}$.

Algorithm 4.4: Instantaneous Reward Operator

1: **Input** $\mathcal{M}_{\mathcal{C}} = (S, S_0, A, \Omega, \mathcal{A}, \mathcal{F}, L)$, $\phi = \boldsymbol{R}_{\lhd v}[\boldsymbol{\mathcal{I}^{=k}}]$
2: **Output** The set $y = Sat(\phi)$
3: Initialize $\mathbf{x}^0 = \mathbf{r_s}$
4: **for** i = 1 : k **do**
5: **for all** $s \in S$ **do**
6: $x_s^i = \max_{a \in \mathcal{A}(s)} \max_{\mathbf{f}_s^a \in \mathcal{F}_s^a} \sum_{s'} f_{ss'}^a x_{s'}^{i-1}$
7: **end for**
8: **end for**
9: $\mathbb{E}^{max}[\boldsymbol{\mathcal{I}^{=k}}] = \mathbf{x}^k$
10: $y = \{s \in S \mid \mathbb{E}_s^{max} \lhd v\}$

4.2.4 *Instantaneous Reward* Operator

The *Instantaneous Reward* operator $\boldsymbol{\mathcal{I}^{=k}}$ computes the expected reward of the state entered after k steps of the execution of the Convex-MDP. For example, if the Convex-MDP models an I/O queue for a wireless node and the reward structure represents the queue size in each state, then the instantaneous reward operator may compute the maximum expected size of the queue after the first $k = 2$ rounds of the algorithm to build the network backbone.

Formally, properties for this operator are expressed as:

$$\phi = \boldsymbol{R}^{\,r}_{\lhd v}[\boldsymbol{\mathcal{I}^{=k}}]$$

The pseudocode of the model-checking routine for the *Instantaneous Reward* operator is shown in Algorithm 4.4. First, we initialize the vector $\mathbf{x}^0 \in \mathbb{R}^N$ with the value of the state reward function r_s evaluated for each state $s \in S$ (line 3). As explained in Section 4.1.2, Markov adversaries are not sufficient to maximize the expected instantaneous reward $\mathbb{E}^{max}[\boldsymbol{\mathcal{I}^{=k}}]$, so we need to unroll the execution of the Convex-MDP for k steps. This is done at lines $4 - 9$ where the maximum instantaneous rewards $\mathbb{E}^{max}[\boldsymbol{\mathcal{I}^{=k}}] = \mathbf{x}^k = J^k(\mathbf{x}^0)$ are computed for all states $s \in S$ applying k times mapping J defined as:

$$\mathbf{x}^i = J(\mathbf{x}^{i-1}) = \max_{a \in \mathcal{A}(s)} \max_{\mathbf{f}_s^a \in \mathcal{F}_s^a} \sum_{s'} f_{ss'}^a x_{s'}^{i-1} \quad \forall s \in S \tag{4.27}$$

Finally, the computed instantaneous rewards are compared to the threshold v (line 10).

Remark 4.6. *The sets $\mathcal{F}_s^a$ can be expressed with models of uncertainty different from one another $\forall s \in S, \forall a \in \mathcal{A}(s)$, since each inner optimization problem in Mapping (4.27) gets solved independently from the others.*

Lemma 4.5. ***Model Checking of the*** **Instantaneuous Reward** ***Operator.*** *The routine to verify the* Instantaneous Reward *operator is sound, complete and guaranteed to terminate with algorithmic complexity that is polynomial in the size $\mathcal{R}$ of $\mathcal{M}_{\mathcal{C}}$ and pseudo-polynomial in the time horizon k of $\boldsymbol{\mathcal{I}^{=k}}$.*

Proof. The proof follows the same reasoning of the one for the *Bounded Until* operator in Section 4.2.2 and it is not repeated for brevity. □

We verify property $\phi = \boldsymbol{R}_{\leq 0.95}[\boldsymbol{\mathcal{I}^{=1}}]$ for the Convex-MDP shown in Figure 2.3. First, we initialize $\mathbf{x}^0 = [0, 1, 1, 0]$. Applying once Mapping (4.27), we get $\mathbb{E}^{max}[\boldsymbol{\mathcal{I}^{=1}}] = [1, 1, 1, 0.9]$ and $Sat(\phi) = \{s_3\}$.

4.2.5 *Bounded Cumulative Reward* Operator

The *Bounded Cumulative Reward* operator $\boldsymbol{\mathcal{C}^{\leq k}}$ computes the expected reward accumulated during k steps of the execution of the Convex-MDP. For example, if the Convex-MDP models a computation system and the reward structure represents the energy spent to execute an instruction, then the Bounded Cumulative Reward operator may compute the total expected energy spent after executing $k = 100$ instructions.

Formally, properties for this operator are expressed as:

$$\phi = \boldsymbol{R}^{\,r}_{\lhd v}[\boldsymbol{\mathcal{C}^{\leq k}}]$$

The pseudocode of the model-checking routine for the *Bounded Cumulative Reward* operator is shown in Algorithm 4.5. First, we initialize the vector $\mathbf{x}^0 = \mathbf{0} \in \mathbb{R}^N$ (line 3). As explained in Section 4.1.2, Markov adversaries are not sufficient to maximize the expected bounded cumulative

Algorithm 4.5: Bounded Cumulative Reward Operator

1: **Input** $\mathcal{M}_{\mathcal{C}} = (S, S_0, A, \Omega, \mathcal{A}, \mathcal{F}, L)$, $\phi = \boldsymbol{R}_{\lhd v}[\boldsymbol{\mathcal{C}^{\leq k}}]$
2: **Output** The set $y = Sat(\phi)$
3: Initialize $\mathbf{x}^0 = \mathbf{0}$
4: **for** i = 1 : k **do**
5: **for all** $s \in S$ **do**
6: $x_s^i = r_s(s) + \max_{a \in \mathcal{A}(s)} \left[r_a(s,a) + \max_{\mathbf{f}_s^a \in \mathcal{F}_s^a} \sum_{s'} f_{ss'}^a x_{s'}^{i-1} \right]$
7: **end for**
8: **end for**
9: $\mathbb{E}^{max}[\boldsymbol{\mathcal{C}^{\leq k}}] = \mathbf{x}^k$
10: $y = \{s \in S \mid \mathbb{E}_s^{max} \lhd v\}$

reward $\mathbb{E}^{max}[\boldsymbol{\mathcal{C}}^{\leq \boldsymbol{k}}]$, so we need to unroll the execution of the Convex-MDP for k steps. This is done at lines $4-9$ where the maximum bounded cumulative rewards $\mathbb{E}^{max}[\boldsymbol{\mathcal{C}}^{\leq \boldsymbol{k}}] = \mathbf{x}^k = L^k(\mathbf{x}^0)$ are computed for all states $s \in S$ applying k times mapping L defined as:

$$\mathbf{x}^i = L(\mathbf{x}^{i-1}) = r_s(s) + \max_{a \in \mathcal{A}(s)} \left[r_a(s,a) + \max_{\mathbf{f}_s^a \in \mathcal{F}_s^a} \sum_{s'} f_{ss'}^a x_{s'}^{i-1} \right] \qquad \forall s \in S \tag{4.28}$$

Finally, the computed bounded cumulative rewards are compared to the threshold v (line 10).

Remark 4.7. *The sets $\mathcal{F}_s^a$ can be expressed with models of uncertainty different from one another $\forall s \in S, \forall a \in \mathcal{A}(s)$, since each inner optimization problem in Mapping (4.28) gets solved independently from the others.*

Lemma 4.6. ***Model Checking of the* Bounded Cumulative Reward *Operator.*** *The routine to verify the* Bounded Cumulative Reward *operator is sound, complete and guaranteed to terminate with algorithmic complexity that is polynomial in the size $\mathcal{R}$ of $\mathcal{M}_{\mathcal{C}}$ and pseudo-polynomial in the time horizon k of $\boldsymbol{\mathcal{C}}^{\leq \boldsymbol{k}}$.*

Proof. The proof follows the same reasoning of the one for the *Bounded Until* operator in Section 4.2.2 and it is not repeated for brevity. □

We verify property $\phi = \boldsymbol{R}_{\leq 3}[\boldsymbol{\mathcal{C}}^{\leq \mathbf{2}}]$ for the Convex-MDP shown in Figure 2.3. First, we initialize $\mathbf{x}^0 = [0,0,0,0]$. After the first (second) iteration of Mapping (4.28), we get $\mathbf{x}^1 = [1,2,1,2]$ ($\mathbf{x}^2 = [2.8, 3.5, 3, 3.6]$). We can thus set $\mathbb{E}^{max}[\boldsymbol{\mathcal{C}}^{\leq \mathbf{2}}] = \mathbf{x}^2$ and $Sat(\phi) = \{s_0, s_2\}$.

4.2.6 *Cumulative Reward* Operator

The *Cumulative Reward* operator $\boldsymbol{\mathcal{C}}\ \phi_1$ computes the expected reward accumulated during the execution of the Convex-MDP before reaching a state satisfying ϕ_1. For example, if the Convex-MDP models an I/O queue for a wireless node and the reward structure represents the number of attempted transmissions, then the cumulative reward operator may compute the expected number of retransmissions before successfully sending a packet across the lossy wireless channel.

Formally, properties for this operator are expressed as:

$$\phi = \boldsymbol{R}^{r}_{\lhd v}[\boldsymbol{\mathcal{C}}\ \phi_1]$$

The pseudocode of the model-checking routine for the *Cumulative Reward* operator is shown in Algorithm 4.6. First, the sets

$$\begin{aligned} S^{yes} &\stackrel{def}{=} Sat(\phi_1) \\ S^{\infty} &\stackrel{def}{=} S \setminus \{s \in S \mid \forall \pi \in \Pi_s,\ \exists j \in \mathbb{N} \text{ s.t. } \pi[j] \in S^{yes}\} \\ S^{?} &\stackrel{def}{=} S \setminus (S^{\infty} \cup S^{yes}) \end{aligned}$$

Algorithm 4.6: Cumulative Reward Operator.

1: **Input** $\mathcal{M}_{\mathcal{C}} = (S, S_0, A, \Omega, \mathcal{A}, \mathcal{F}, L)$, $\phi = \boldsymbol{R}_{\lhd v}[\boldsymbol{\mathcal{C}}\ \phi_1]$
2: **Output** The set $y = Sat(\phi)$
3: Compute S^{yes} and S^{∞}
4: $\mathbb{E}^{max}[\boldsymbol{\mathcal{C}}\ \phi] =$ ComputeRewards()
5: $y = \{s \in S \mid \mathbb{E}^{max}_{s} \lhd v\}$

are precomputed using classical reachability routines over the Convex-MDP underlying graph [60] (line 3). In particular, the set S^{∞} contains the states $s \in S$ starting from which it exists at least one path in the execution of the Convex-MDP which does not visit states satisfying ϕ_1. Since the optimal adversary would select that path to maximize the expected reward, the maximum expected reward for the states $s \in S^{\infty}$ is infinite, i.e., $\mathbb{E}^{max}_{S^{\infty}}[\boldsymbol{\mathcal{C}}\ \phi_1] = +\infty$. We thus preprocess the Convex-MDP underlying graph to disconnect all states $s \in S^{\infty}$. Second, the maximum expected rewards to satisfy ϕ, as defined in Equation (4.11), are computed for all states $s \in S \setminus S^{\infty}$ by calling the procedure "ComputeRewards()" (line 4). Finally, the computed probabilities are compared to the threshold v to determine the satisfiability set $Sat(\phi)$ (line 5).

Similarly to the analysis of the *Unbounded Until* operator, in the rest of the section we propose two implementations of the procedure "ComputeRewards()", both returning the exact solution modulo rounding errors due to the machine finite resolution. The first implementation is based on Convex Programming (CP), while the second is based on Value Iteration (VI). We report both implementations since they differ in applicability and runtime performance. The CP procedure has algorithmic complexity provably polynomial in $\mathcal{R}$, but it can only be applied to Convex-MDPs that satisfy Assumption 2.4. On the other hand, the VI procedure can be applied to any convex model of uncertainty, but its algorithmic complexity depends not only on the problem size but also on the problem data. In general, the user is invited to use both procedures for the model checking of the problem at hand and select the one that performs better for the specific scenario of interest.

Since most of the material in this section can be derived following the same steps already presented about the verification of the *Unbounded Until* operator in Section 4.2.3, in the following we only report the final results for each procedure. The reader is invited to refer to Section 4.2.3 for a complete derivation.

4.2.6.1 Convex Programming Procedure (CP)

We start from the classical Linear Programming (LP) formulation to solve the problem without the presence of uncertainty:

$$\begin{aligned} &\min_{\mathbf{x}} \mathbf{x}^{\mathbf{T}}\mathbf{1} \\ &\text{s.t. } x_s = 0; && \forall s \in S^{yes}; \\ &\quad x_s \geq r_s(s) + r_a(s,a) + \mathbf{x}^{\mathbf{T}}\mathbf{f_s^a} && \forall s \in S^{?}, \forall a \in \mathcal{A}(s) \end{aligned} \tag{4.29}$$

where $\mathbb{E}^{max}[\boldsymbol{\mathcal{C}}\ \phi_1] = \mathbf{x}^*$ is computed solving only one LP. Intuitively, Problem (4.29) selects the optimal (in the worst-case sense) adversary by selecting the highest upper bound on the values of the expected rewards x_s across the actions $a \in \mathcal{A}(s)$ available at each state $s \in S^{?}$. The main difference with respect to the formulation for the *Unbounded Until* operator in Problem (4.15) is the addition of the reward functions r_s and r_a evaluated $\forall s \in S, \forall a \in \mathcal{A}(s)$.

Since the *Cumulative Reward* operator admits optimal Markov deterministic adversaries and natures, as proved in Proposition 4.1, we can rewrite Problem (4.29) when uncertainties are added to the model as:

$$\begin{aligned} &\min_{\mathbf{x}} \mathbf{x}^{\mathbf{T}}\mathbf{1} \\ &\text{s.t. } x_s = 0; && \forall s \in S^{no}; \\ &\quad x_s = 1; && \forall s \in S^{yes}; \\ &\quad x_s \geq r_s(s) + r_a(s,a) + \max_{\mathbf{f_s^a} \in \mathcal{F}_s^a} (\mathbf{x}^{\mathbf{T}}\mathbf{f_s^a}) && \forall s \in S^{?}, \forall a \in \mathcal{A}(s) \end{aligned} \tag{4.30}$$

i.e., we further maximize the upper bound on the expected rewards $\mathbf{x}$ across the action range of the adversarial nature.

Following the same steps presented in Section 4.2.3.1, we substitute all inner primal problems in Problem (4.30) with the corresponding dual problems. Thanks to strong duality, we are guaranteed that this transformation does not change the result of the outer optimization problem. We then drop all the dual inner optimization operators and obtain the CP formulation:

$$\begin{aligned} &\min_{\mathbf{x},\boldsymbol{\lambda}} \mathbf{x}^{\mathbf{T}}\mathbf{1} \\ &\text{s.t. } x_s = 0; && \forall s \in S^{yes}; \\ &\quad x_s \geq r_s(s) + r_a(s,a) + g\left(\boldsymbol{\lambda}_s^a, \mathbf{x}\right); && \forall s \in S^{?}, \forall a \in \mathcal{A}(s); \\ &\quad \boldsymbol{\lambda}_s^a \in \mathcal{D}_s^a && \forall s \in S^{?}, \forall a \in \mathcal{A}(s) \end{aligned} \tag{4.31}$$

where $\boldsymbol{\lambda_s^a}$ is the (vector) Lagrange multiplier, $g\left(\boldsymbol{\lambda}_s^a, \mathbf{x}\right)$ is the cost function of the dual problem and $\mathcal{D}_s^a$ is the feasibility set for each dual problem.

In general, the CP procedure can only be applied to Convex-MDPs that satisfy Assumption 2.4, repeated here for convenience.

Assumption 4.2. ***Joint-Convexity.*** *Given a Convex-MDP $\mathcal{M}_\mathcal{C}$, for all convex uncertainty sets $\mathcal{F}_s^a \in \mathcal{F}$, the dual function $g(\boldsymbol{\lambda}_s^a, \mathbf{x})$ in Problem (4.31) is jointly-convex in both $\boldsymbol{\lambda}_s^a$ and $\mathbf{x}$.*

Remark 4.8. *We note that we can combine models of uncertainty different from one another within a single CP formulation, since each instance of the dual function $g(\boldsymbol{\lambda}_s^a, \mathbf{x})$ in Problem (4.31) is independent from the others. Moreover, according to Assumption 2.1, all uncertainty sets $\mathcal{F}_s^a \in \mathcal{F},\ \forall s \in S, \forall a \in \mathcal{A}(s)$ are independent from one another, so the convexity of the CP formulation is preserved. As an example, if both the interval and ellipsoidal models are used, the overall CP formulation is an SOCP.*

We can now summarize the results of this section in the following Lemma.

Lemma 4.7. ***Model Checking of the*** **Cumulative Reward** ***Operator Using the CP Procedure.*** *The CP procedure to verify the* Cumulative Reward *operator is sound, complete and guaranteed to terminate with algorithmic complexity polynomial in the size $\mathcal{R}$ of $\mathcal{M}_\mathcal{C}$, if $\mathcal{M}_\mathcal{C}$ satisfies Assumption 4.2.*

Sketch of proof. The proof follows the same reasoning of the one for Lemma 4.3 in Section 4.2.3.1. □

We verify $\phi = \boldsymbol{R}_{\leq 3}[\boldsymbol{\mathcal{C}}\ \omega]$ on the Convex-MDP shown in Figure 2.3 using the CP procedure. The solution reads: $\mathbb{E}^{max}[\boldsymbol{\mathcal{C}}\ \omega\ ] = [5, 2, 0, 5]$, and, in conclusion, $Sat(\phi) = \{s_1, s_2\}$.

4.2.6.2 Value Iteration Procedure (VI)

In this section, we present an alternative procedure for the model checking of the *Cumulative Reward* operator based on Value Iteration (VI). Analogously to the description of the CP procedure, also the derivation of the VI procedure closely follows the steps described for the *Unbounded Until* operator in Section 4.2.3.2. We thus report here only the main results.

Intuitively, the VI procedure unrolls the execution of the Convex-MDP until all the existing paths to reach the set of states satisfying ϕ_1 have been explored, so that the expected reward accumulated before reaching the target states can be computed. Since in general there might exist paths of arbitrary length to reach the target states, the expected reward can only be approximated. We will call ϵ_d the desired level of accuracy in the estimation of the expected reward $\mathbb{E}^{max}[\boldsymbol{\mathcal{C}}\,\phi_1]$. In the following, we prove the soundness and correctness of the procedure by showing that it converges to the exact solution in the limit of infinite iterations. We refer the reader to Section 4.2.3.2 for a discussion on the exiting conditions to achieve the desired accuracy ϵ_d.

We use the following mapping L:

$$\mathbf{x}^i = L(\mathbf{x}^{i-1}) = \begin{cases} +\infty & s \in S^{\infty} \\ 0 & \forall s \in S^{yes} \\ r_s(s) + \max\limits_{a \in \mathcal{A}(s)} \left[r_a(s,a) + \max\limits_{\mathbf{f}_s^a \in \mathcal{F}_s^a} \sum_{s'} f_{ss'}^a x_{s'}^{i-1} \right] & \forall s \in S^{?} \end{cases} \tag{4.32}$$

with the sets S^{yes}, S^{∞} and $S^{?}$ defined in Section 4.2.6.1.

We can now prove that:

Proposition 4.5. *Mapping L is a contraction over the metric space $(\mathbb{R}^N, \| \cdot \|_\infty)$.*

Sketch of proof. The proof closely follows the one for the *Unbounded Until* operator in Appendix B. □

We now state the main result of this section:

Lemma 4.8. ***Model Checking of the*** **Cumulative Reward** ***Operator Using the VI Procedure.*** *The VI procedure to verify the* Cumulative Reward *operator is sound and complete, i.e.,*

$$\mathbb{E}^{max}[\boldsymbol{\mathcal{C}}\ \phi_1] = \lim_{k \to +\infty} L^k(\mathbf{x}) \tag{4.33}$$

Sketch of proof. The proof closely follows the one for the *Unbounded Until* operator in Section 4.2.3.2. □

We use the VI routine with $\epsilon_d = 10^{-3}$ to verify again $\phi = \boldsymbol{R}_{\leq 3}[\ \boldsymbol{\mathcal{C}}\ \omega\ ]$ in the example in Figure 2.3. After 16 iterations, we get $\mathbb{E}^{max}[\ \boldsymbol{\mathcal{C}}\ \omega\ ] = [4.997, 1.999, 0, 4.998]$ and $Sat(\phi) = \{s_1, s_2\}$.

4.2.7 Summary of the Properties of the Model-Checking Routines

We conclude the section with a summary of the main properties of the presented model-checking routines. We refer to Table 4.1. For each operator defined in the PCTL syntax, we list in columns $2-3$ whether the optimal (in the worst-case sense) adversary and nature are Markov or History-Dependent (for all operators, there exist a deterministic optimal adversary and nature). In column $4-5$, we report the algorithmic complexity results for the model-checking routine. In particular, we differentiate between complexity in the size of the Convex-MDP model ($\mathcal{R}$) and in the size of the PCTL formula $\mathcal{Q}$.

Table 4.1: Summary of the Properties of the Model-Checking Routines

Operator	Optimal Adversary and Nature		Algorithmic Complexity	
	Markov	History-Dependent	in $\mathcal{R}$	in $\mathcal{Q}$
$\boldsymbol{\mathcal{X}}$	$\surd$	-	*poly*	*poly*
$\boldsymbol{\mathcal{U}^{\leq k}}$	X	$\surd$	*poly*	*pseudo-poly(k)*
$\boldsymbol{\mathcal{U}}$	$\surd$	-	*poly*	*poly*
$\boldsymbol{\mathcal{I}^{=k}}$	X	$\surd$	*poly*	*pseudo-poly(k)*
$\boldsymbol{\mathcal{C}^{\leq k}}$	X	$\surd$	*poly*	*pseudo-poly(k)*
$\boldsymbol{\mathcal{C}}$	$\surd$	-	*poly*	*poly*

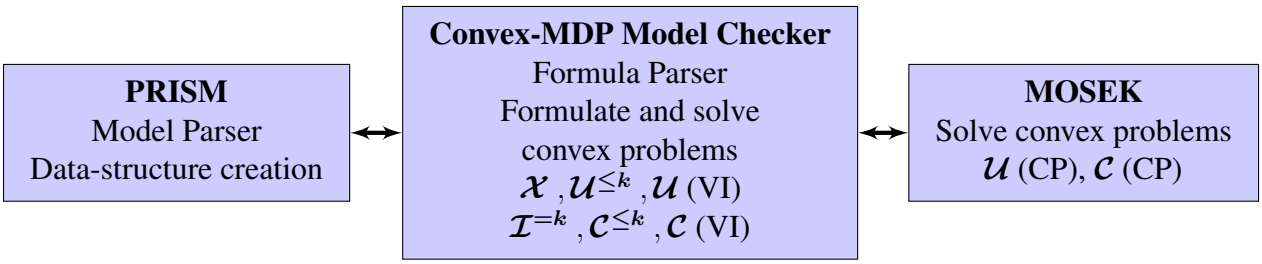


Figure 4.3: Block-diagram of the interfaces among the different software packages used to implement the verification algorithm. For each solver, we also list the supported routines for the verification of PCTL operators.

4.3 Experimental Evaluation of the Model Checker

The main focus of this dissertation is the application of the developed algorithms to problems of practical relevance. Two factors need to be considered to achieve this goal. First, the development of Convex-MDP models of relevant systems for which modeling uncertainties in the transition probabilities do have an impact in the model-checking results. Second, the runtime of the verification algorithm needs to scale acceptably also when running on models of size that reflects actual applications. In this section, we first present an overview of the software implementation of the developed algorithms. We then analyze three case studies of systems of practical importance to assess the impact of uncertainties on the verification results and to experimentally evaluate the runtime performance of the proposed model checker. We will instead present the newly developed model of the performance of a car driver and results about the verification of its properties in Chapter 5.

4.3.1 Overview of the Software Implementation

We implemented the proposed verification algorithm in Python, and interfaced it with PRISM [99] to extract information about the Convex-MDP model. We used MOSEK [122] to solve the LPs and SOCPs generated for the verification of Convex-MDPs with interval and ellipsoidal models of uncertainties, while we implemented customized numerical solvers for the other models of uncertainty. The core software implementation in Python uses PRISM as the front-end tool to read in the model description. It then formulates the convex problems required to model check the properties of interest. Finally, the convex problems are solved either using the internal solvers or the external back-end solver MOSEK, depending on the operator to be verified and on the adopted model of uncertainty. Figure 4.3 shows a block diagram of the interfaces among the different software packages.

The code implementing the verification routines is reported in Appendix C. It exploits the Object-Oriented Programming (OOP) paradigm to maximize the code reuse across the different models of uncertainty and ease the development of convex-programming solvers for additional models of uncertainties not considered in this dissertation. In particular, only the routines to for-

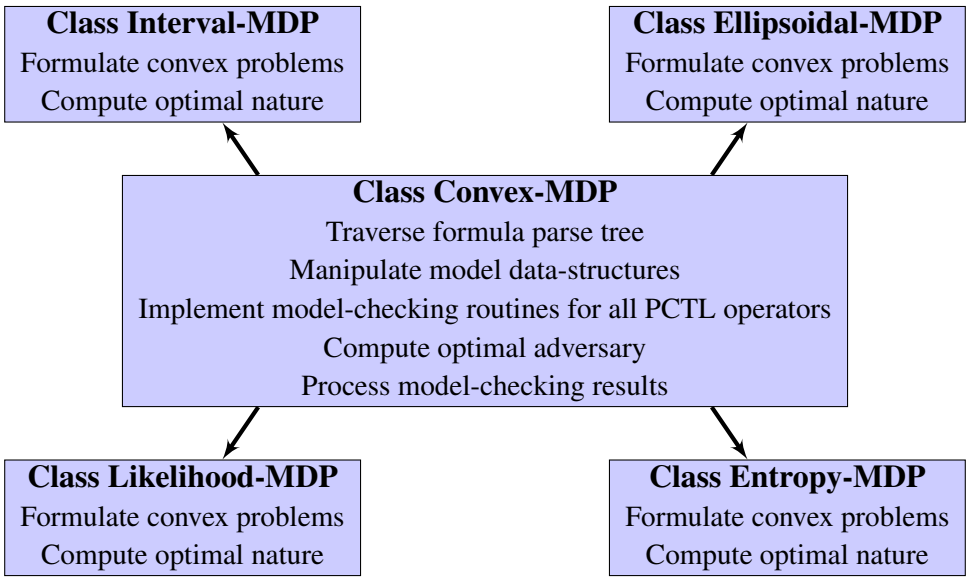


Figure 4.4: Class inheritance diagram for the Python implementation of the model-checking algorithm. Each block represents a class and the arrow points to the class which inherits. For each class, we also list the implemented functionality.

mulate and solve the inner optimization problems in Problems (4.9) – (4.12), for the computation of the optimal nature, are customized to each uncertainty model. Conversely, the code takes advantage of the weakly-typed data structures in Python to share all the other functionalities across all models of uncertainty, including the routines to formulate and solve the outer optimization problems in Problems (4.9) – (4.12) for the computation of the optimal adversary. We report a block-diagram of the class inheritance hierarchy in Figure 4.4.

The implemented tool is available open-source at [1]. At the time of writing, the verification algorithms are also being integrated within PRISM. The goals of such an effort are twofold. First, we aim to exploit the memory-efficient data-structures used in PRISM to further increase the scalability of the proposed approach. Second, we believe that the integration within the state-of-the-art tool for the verification of stochastic systems will ease the adoption of the techniques proposed in this dissertation also by users already familiar with the PRISM working environment. The interested reader is invited to refer to the PRISM website [2] for more details about the date of the public distribution or to contact the author of this dissertation directly.

4.3.2 Case Studies

We tested the implemented model checker by analyzing the following three case studies:

- a distributed stochastic consensus protocol where one of the participating processes is faulty or behaving maliciously because under a security attack;

- a wireless dynamic configuration protocol for IPv4 addresses operating across a lossy physical link;
- the dining philosopher problem, which can be used to model electronic systems sharing a limited resource (e.g., data memory).

The goals of these experiments are two-fold:

1. quantitatively evaluate the impact of uncertainty on the results of verification of PCTL properties of Convex-MDPs;
2. assess the runtime scalability of the proposed approach to problems of increasing size.

The runtime data were obtained on a 2.4 GHz Intel Xeon with 32GB of RAM.

4.3.2.1 Distributed Consensus Protocol

Consensus problems arise in many distributed environments, where a group of distributed processes attempt to reach an agreement about a decision to take by accessing some shared entity. A consensus protocol ensures that the processes will eventually take the same decision, even if they start with initial guesses that might differ from one another.

We analyze the randomized consensus protocol first presented by Aspnes and Herlihy [13] and subsequently further analyzed by Kwiatkowska et al. [101]. The protocol guarantees that the processes return a preference value $v \in \{1, 2\}$, with probability parameterized by a process independent value R ($R \geq 2$) and the number of processes P participating in the protocol. The processes communicate with one another by accessing a shared counter of value c. The protocol proceeds in rounds. At each round, a process flips a local coin, increments or decrements the shared counter depending on the outcome of the coin flip and then reads its value c. If $c \geq PR$ ($c \leq -PR$), the process chooses $v = 1$ ($v = 2$). Note that the larger the value of R, the longer it takes on average for the processes to reach a decision. Non-determinism is used to model the asynchronous access of the processes to the shared counter, so the overall protocol is modeled as an MDP.

We verify the property **Agreement:** *all processes must agree on the same decision, i.e., choose a value* $v \in \{1, 2\}$. We compute the minimum probability of **Agreement** and compare it against the theoretical lower bound $(R-1)/2R$, as derived by Aspnes and Herlihy [13]. In PCTL syntax:

$$\boldsymbol{P}^{\,min}\left[\psi\right] \;:= P_{s_0}^{min}\left[\boldsymbol{F}\left(\{finished\} \wedge \{all\_coins\_equal\_1\}\right)\right] \tag{4.34}$$

where $\boldsymbol{F}\left[\phi\right] := True\;\boldsymbol{\mathcal{U}}\;\phi$ and s_0 is the state representing the beginning of the protocol.

While previous work [101] analyzed the protocol properties assuming an ideal behavior from all participating processes, we consider the case where one of the processes is unreliable or adversarial, i.e., it throws a *biased* coin instead of a *fair* coin. Specifically, the probability of either outcome lies in the uncertainty interval $[(1-u)p_0, (1+u)p_0]$, where $p_0 = 0.5$ according to the protocol. This setting is relevant to analyze the protocol robustness when a process acts erroneously

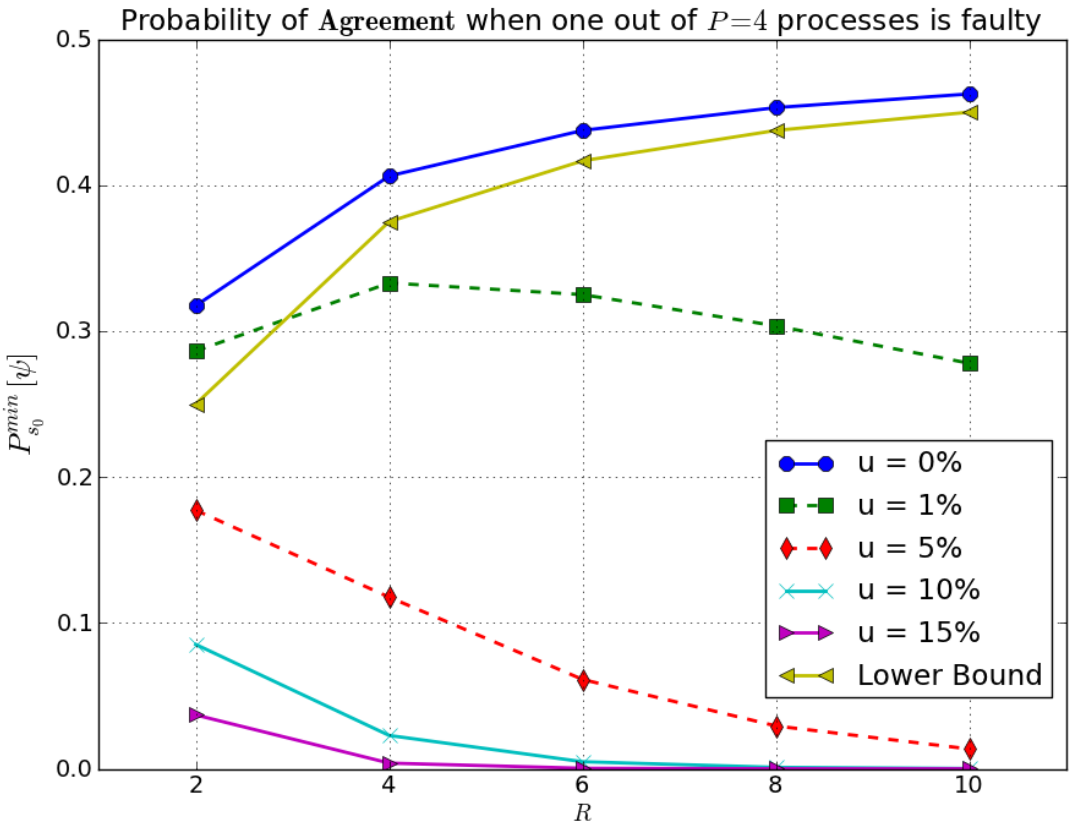


Figure 4.5: The figure shows the value of Equation (4.34) as a function of the value of parameter R while varying the uncertainty level u.

due to a failure or a security breach. In particular, our approach allows to study attacks that deliberately hide under the noise threshold of the protocol. In such attacks, the compromised node defers agreement by producing outputs whose statistical properties are within the noise tolerance of an uncompromised node, so that it is harder to detect its malicious behavior.

Figure 4.5 shows the effect of different levels of uncertainty on the computed probabilities when $P = 4$ processes participate in the protocol. With no uncertainty ($u = 0$), $P^{min}_{s_0}[\psi]$ increases as R increases, because a larger R drives the decision regions further apart, making it more difficult for the processes to decide on different values of v. As R goes to infinity, $P^{min}_{s_0}[\psi]$ approaches the theoretical lower bound $lim_{R\to\infty}(R-1)/2R = 0.5$. However, when uncertainties are added to the model, the behavior changes substantially. For a small level of uncertainty ($u = 1\%$), the satisfaction probability $P^{min}_{s_0}[\psi]$ reaches the maximum for $R = 4$ and it then soon decreases for increasing R. In this scenario, the proposed modeling and verification approaches can thus be used to determine the value of the system parameter R which maximizes system performance. With a larger uncertainty (e.g., $u = 15\%$), $P^{min}_{s_0}[\psi]$ instead goes monotonically to 0. A possible

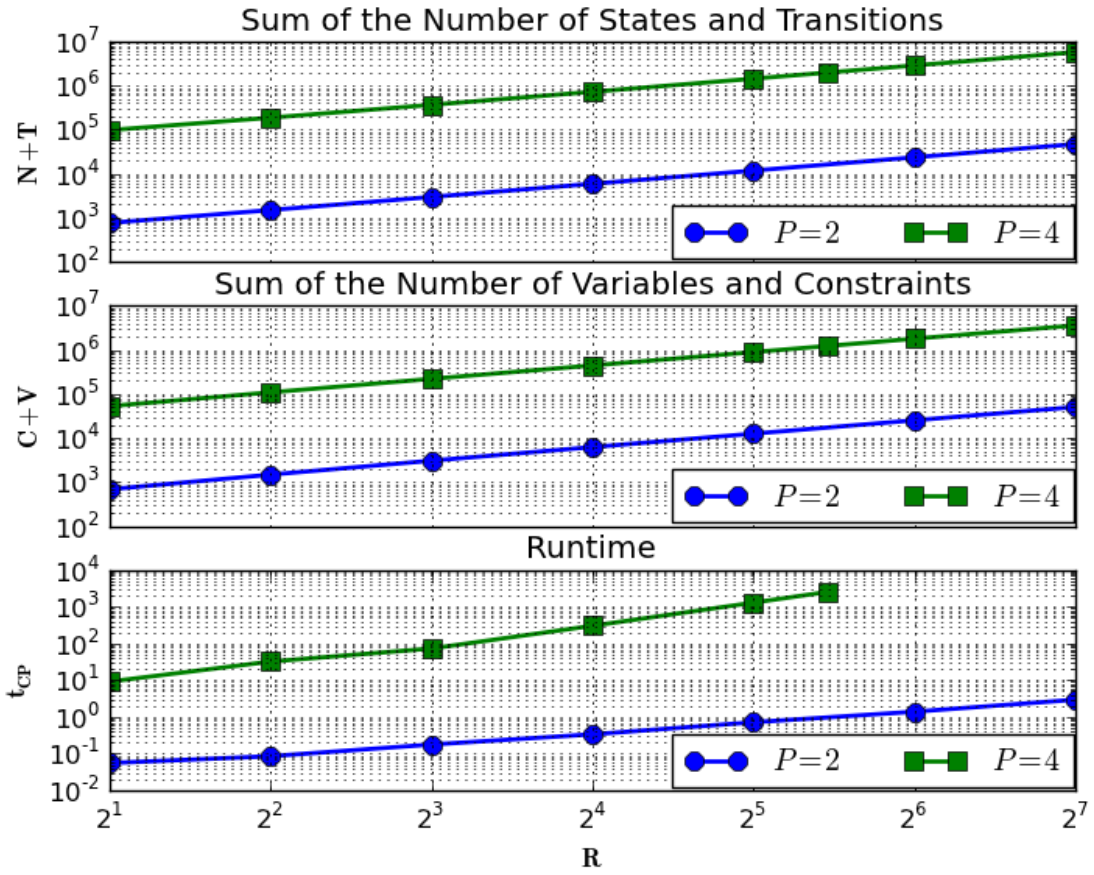


Figure 4.6: The figure reports the results of the analysis of the scalability of the CP procedure. In particular, it shows the trends of the size $N + T$ of the analyzed Convex-MDP (top), the sum of variables and constraints of the $C + V$ formulated convex program (center) and the runtime of the model-checking algorithm (bottom), as a function of the model parameter R.

explanation is that the faulty process has more opportunities to deter agreement for a high value of R, since R also determines the expected time to termination. Results thus show that the protocol is vulnerable to uncertainties. This fact may have serious security implications, i.e., a denial-of-service attack could reduce the availability of the distributed service, since a compromised process may substantially alter the expected probability of agreement. This analysis can also give insight to the protocol designer on how to optimally set the noise threshold to detect malicious behavior in the participating processes. The threshold should be the maximum value of uncertainty which gives a non-monotonic behavior in $P_{s_0}^{min}[\psi]$ and parameter R should be set to achieve the maximum of such satisfaction probability. For the problem at hand, the threshold should be set to $u = 1\%$ and parameter $R = 4$.

The consensus protocol gives us the opportunity to study the scalability of the CP procedure,

Table 4.2: Runtime Comparison

Tool	$P=2$ $R=2$ $N+T=764$	$P=2$ $R=7$ $2,604$	$P=2$ $R=128$ $47,132$	$P=4$ $R=2$ $97,888$	$P=4$ $R=32$ $1,262,688$	$P=4$ $R=44$ $1,979,488$	$P=6$ $R=4$ $14,211,904$
CP	$0.02s$	$0.1s$	$2.1s$	$8.3s$	$1,341s$	$2,689$	TO
PRISM	$0.01s$	$0.09s$	$196s$	$1s$	$2,047s$	TO	$1860s$
PARAM	$22.8s$	$657s$	TO	TO	TO	TO	TO

by evaluating Equation (4.34) while sweeping R both for $P = 2$ and $P = 4$. We set the Time Out (TO) to one hour. In Figure 4.6, we plot the sum ($N+T$) of the number of states (N) and transitions (T) of the Convex-MDP, which are independent of the uncertainty in the transition probabilities, to represent the model size (top), the sum ($V + C$) of the number of variables (V) and constraints (C) of the generated LP instances of Problem (4.19) (center), and the running time t_{CP} of the model-checking algorithm (bottom). $V + C$ always scales linearly with $N + T$ (the lines have the same slope), supporting the polynomial complexity result for our algorithm. Instead, t_{CP} scales linearly only for smaller problems ($P = 2$), while it has a higher-order polynomial behavior for larger problems ($P = 4$) (the line is still a straight line on logarithmic axes but with a steeper slope, so it represents a higher-order polynomial behavior). This behavior depends on the performance of the chosen numerical solver, and it can improve benefiting of future advancements in the solver implementation. In Table 4.2, we compare the CP procedure with two tools, PRISM [99] and PARAM [69], in terms of runtime, for varying values of P and R. Although neither tool solves the same problem addressed in this dissertation, the comparison is useful to assess the practicality of the proposed approach. In particular, PRISM only verifies PCTL properties of MDPs with no uncertainties. PARAM instead derives a symbolic expression of the satisfaction probabilities as a function of the model parameters, to then find the parameter values that satisfy the property. Hence, PRISM only considers a special case of the models considered in this work, while our approach only returns the worst-case scenario computed by PARAM. Results show that the CP procedure runs faster than PRISM for some benchmarks, but it is slower for larger models. This is expected since the scalability of our approach depends mainly on the problem size, while the performance of the iterative engine in PRISM depends on the problem size and on the number of iterations required to achieve convergence, which is dependent on the problem data. Finally, our approach is orders of magnitude faster than PARAM, so it should be preferred to perform worst-case analysis of system performances.

4.3.2.2 ZeroConf Dynamic Configuration Protocol for IPv4 Link-Local Addresses

The ZeroConf protocol [39, 100] is an Internet Protocol (IP)-based configuration protocol for local (e.g., domestic) networks. In such a local context, each device should configure its own unique IP address when it gets connected to the network, with no user intervention. The protocol thus offers a distributed "plug-and-play" solution in which address configuration is managed by individual devices when they are connected to the network. The network is composed of DV_{tot}

devices. After being connected, a new device chooses randomly an IP address from a pool of $IP_A = 65024$ available ones, as specified by the standard. The address is non-utilized with probability $p_0 = 1 - DV_{tot}/IP_A$. It then sends messages to the other devices in the network, asking whether the chosen IP address is already in use. If no reply is received, the device starts using the IP address, otherwise the process is repeated.

The protocol is both probabilistic and timed. Probability is used in the randomized selection of an IP address and to model the eventuality of message loss; timing defines intervals that elapse between message retransmissions. The protocol has been modeled by Kwiatkowska et al. [100] as an MDP using the digital clock semantic of time. In this semantic, time is discretized in a finite set of epochs which are mapped to a finite number of states in an MDP, indexed by the epoch variable t_e. To enhance the user experience and, in battery-powered devices, to save energy, it is important to guarantee that a newly-connected device manages to select a unique IP address within a given

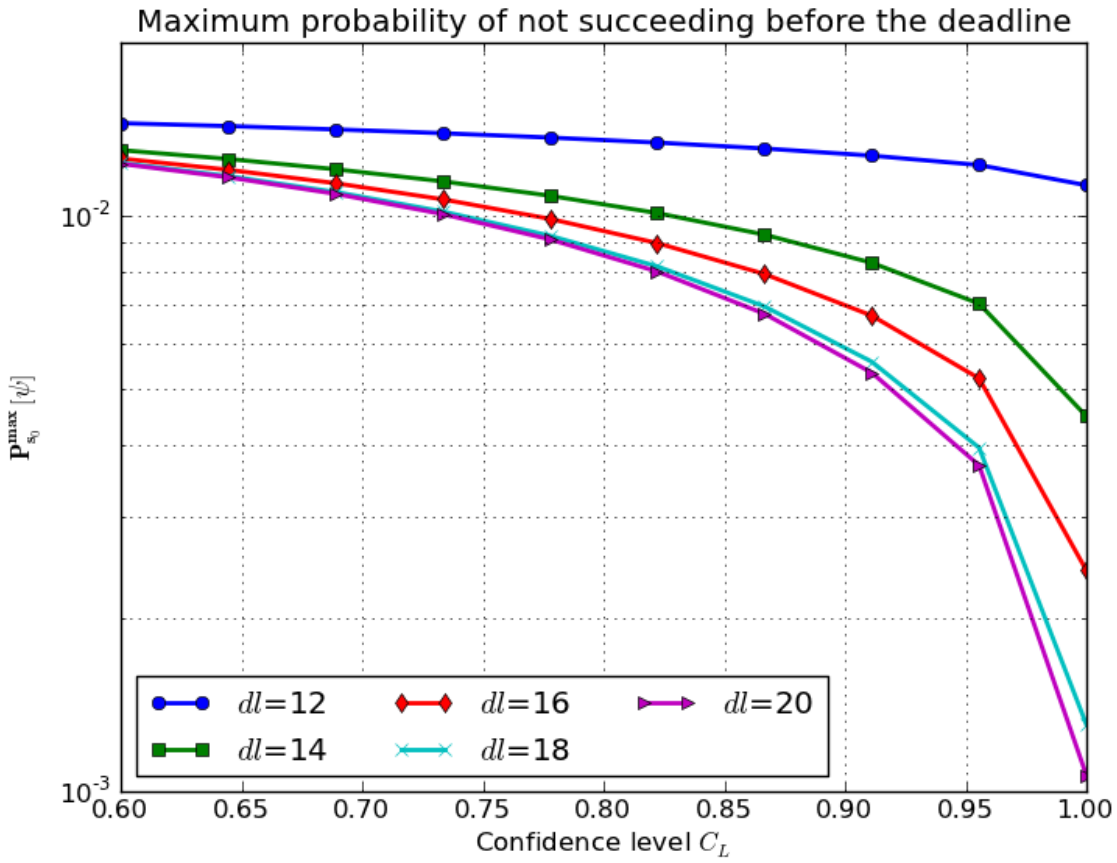


Figure 4.7: The figure shows the trend of the value of Equation (4.35) as a function of the confidence level C_L for different values of the protocol parameter dl, i.e., the time deadline after which the protocol declares failure.

deadline *dl*. For numerical reasons, we study the maximum probability of *not* being able to select a valid address within *dl*. In PCTL syntax:

$$\boldsymbol{P}^{\,max}\,[\psi] \;:=\; P^{max}_{s_0}\,[\neg\{unique\_address\}\;\boldsymbol{\mathcal{U}}\;\{t_e > dl\}] \tag{4.35}$$

where s_0 is the state representing the moment in which the device gets connected to the network.

We analyzed how network performances vary when there is uncertainty in estimating: 1) the probability of selecting an IP address, and; 2) the probability of message loss during transmission. The former may be biased in a faulty or malicious device. The latter is estimated from empirical data, so it is approximated. Further, the chosen semantic interpretation of the Convex-MDP behavior, as stated in Assumption 2.3, which allows a nature to select a different transition distribution at each execution step, properly models the time-varying characteristics of the transmission channel.

In Figure 4.7, we added uncertainty only to the probability of message loss using the likelihood model, which is suitable for empirically-estimated probabilities. Using classical results from statis-

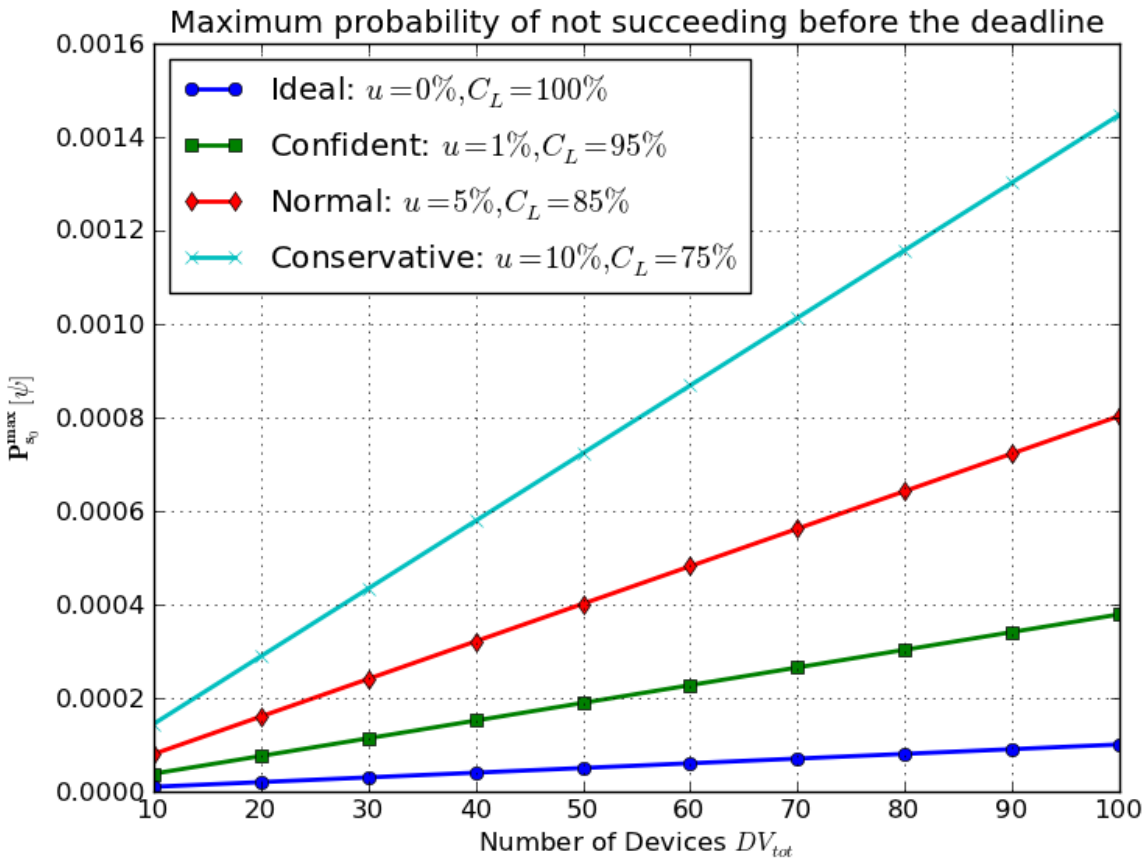


Figure 4.8: The figure shows the trend of the value of Equation (4.35) as a function of the number of devices in the network for different level of confidence in the estimations of the model parameters.

tics [125], we computed the value of parameter β_s^a in Set (2.3) corresponding to several confidence levels C_L in the measurements. In particular, $0 \leq C_L \leq 1$ and $C_L = 1 - cdf_{\chi^2_d}\left(2 * (\beta^a_{s,max} - \beta^a_s)\right)$, where $cdf_{\chi^2_d}$ is the cumulative density function of the Chi-squared distribution with d degrees of freedom ($d = 2$ here because there are two possible outcomes, message lost or received). Results show that the value of $P^{max}_{s_0}[\psi]$ increases by up to $\sim 10\times$ for decreasing C_L, while classical model checking would only report the value for $C_L = 1$, which coarsely over-estimates network performance. The plot can be used by a designer to choose *dl* to make the protocol robust to varying channel conditions, or by a field engineer to assess when the collected measurements are enough to estimate network performances.

In Figure 4.8, we combine different models of uncertainty, i.e., we also add uncertainty in the probability of selecting the new IP address using the interval model. This probability thus lies in the

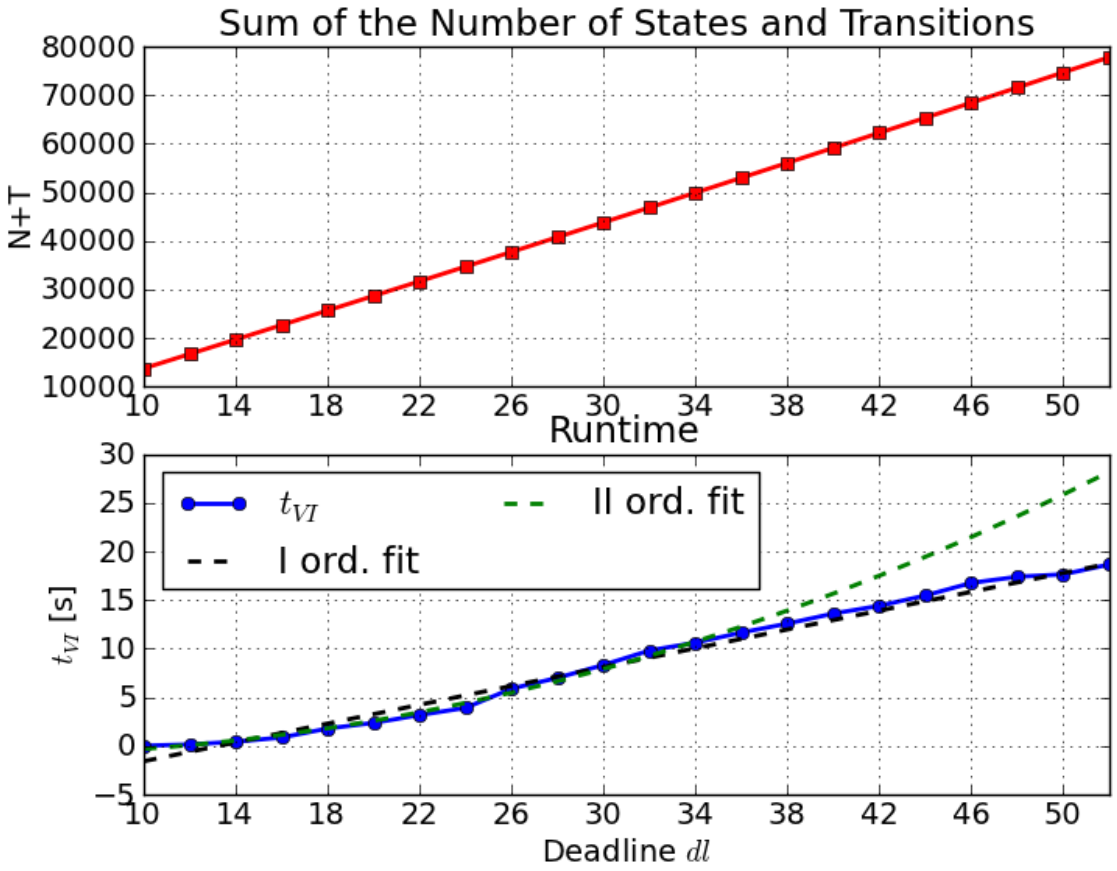


Figure 4.9: The figure shows the trend of the number of states and transitions of the analyzed Convex-MDP (top) and the runtime for increasing values of the protocol parameter dl. The runtime trend is also overlaid to a first-order and a second-order polynomial fit to highlight its linear increase.

interval $[(1-u)p_0, (1+u)p_0]$. We, arbitrarily, fix $dl = 25$ and sweep DV_{tot} in the range $[10-100]$, which covers most domestic applications, to study how network congestion affects the value of Equation (4.35). We study four scenarios: the *ideal* scenario, returned by classical model-checking techniques; the *confident, normal, conservative* scenarios, where we add increasing uncertainty to model different knowledge levels of the network behavior, a situation that often arises during the different design phases, from conception to deployment. Results show that $P_{s_0}^{max}[\psi]$ gets up to $\sim 15\times$ higher than the ideal scenario, an information that designers can use to determine the most sensitive parameters of the system and to assess the impact of their modeling assumptions on the estimation of network performances.

Finally, in Figure 4.9, we report at the top the sum of the number of states (N) and transitions (T) of the Convex-MDP, as defined in Section 4.3.2.1, and, at the bottom, the verification runtime t_{VI} of the Value Iteration procedure, as functions of the value of the model parameter *dl*. As the figure shows, the size of the model $N + T$ increases linearly with the value of dl. To highlight the fact that also the verification runtime t_{VI} scales linearly, we fit t_{VI} for $10 \leq dl \leq 35$ with a first and a second-order polynomials and extrapolated the two polynomial behaviors up to $dl = 52$. Results show that indeed the trend of t_{VI} is closer to the linear than to the quadratic behavior.

4.3.2.3 The Dining Philosophers Problem

We analyze the classical Dining Philosopher Problem [51, 107]. Briefly, n philosophers are sitting at a table with n available forks. Each philosopher can either think or eat: when he becomes *hungry*, he needs to pick both the fork on his right and on his left before *eating*. Since there are not enough forks to allow all philosophers to eat together, they need to follow steps according to a stochastic protocol to eat in turns. We consider this case study relevant because it can be used to model real shared-resources stochastic protocols [107].

We model the uncertainty of the philosophers in deciding which fork to pick first. While the nominal protocol assigns $0.5-0.5$ probability to the left and right fork, we assume that these values are only known with $C_L = \pm 10\%$ confidence. We will use all the uncertainty models presented in Section 2.1.3 to represent the uncertainty in the estimation of the transition probabilities. Even though only the interval model has a practical relevance for this case study, for reasons similar to the ones for the stochastic consensus protocol presented in Section 4.3.2.1, this analysis will allow us to compare the impact of the different models of uncertainty on the verification results. In particular, the parameters for each model of uncertainty corresponding to the chosen level of confidence C_L can be set using the approach suggested by Nilim and El Ghaoui [125]. For example, for the interval model, the probabilities lie in the interval $[45\% - 55\%]$. Within this setting, we aim to determine what is the *quantitative* minimum probability for any philosopher to *eat* within k steps of the protocol after he becomes *hungry*. In PCTL syntax:

$$\boldsymbol{P}^{\,min}[\psi] := \mathbf{P}_{S_0}^{min}\left[\mathrm{F}^{\leq \boldsymbol{k}}\{Eating\}\right] \tag{4.36}$$

with initial states $S_0 = Sat(Hungry)$. Figure 4.10 shows the evolution of the value of Equation (4.36) as a function of the number of protocol steps k. As expected, the probability of eating

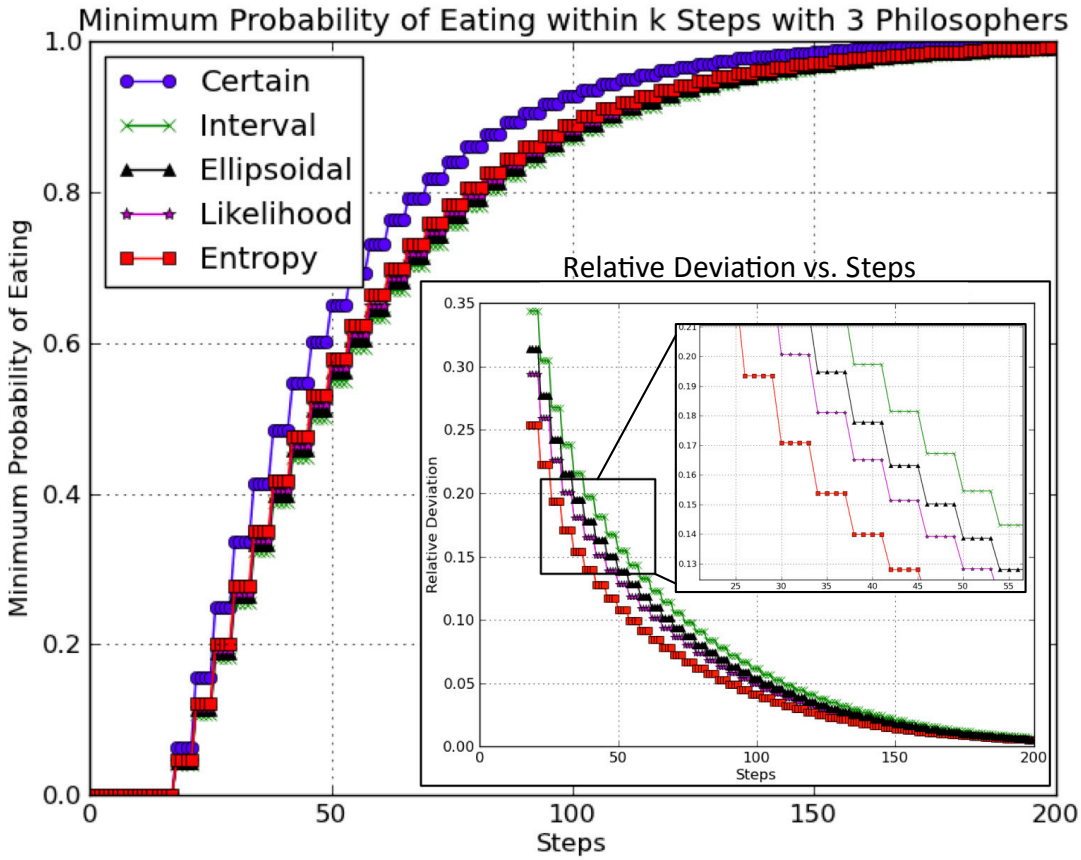


Figure 4.10: The figure shows the evolution of Equation (4.36) for increasing values of the number of protocol steps k for different models of uncertainties in modeling transition probabilities. The inset shows the relative deviation of the value of Equation (4.36) with respect to the model with no uncertainties.

steadily increases as the number of steps increases. However, the plot also shows that adding uncertainty *decreases* this probability with respect to the *certain* scenario. The inset of Figure 4.10 shows the relative deviation in probability with respect to the certain case, defined as:

$$\text{Relative deviation} = \frac{\boldsymbol{P}_{*}^{max}[\psi] - \boldsymbol{P}_{\text{Certain}}^{max}[\psi]}{\boldsymbol{P}_{\text{Certain}}^{max}[\psi]}$$

with $* \in \{\text{Interval}, \text{Ellipsoidal}, \text{Likelihood}, \text{Entropy}\}$.

As the figure shows, a $\pm 10\%$ uncertainty can cause a deviation up to 35% in the computed probabilities, and the deviation is always higher than 10% for $k \leq 60$. Further, the deviation is larger for the Interval and Ellipsoidal models, since they are the most conservative among the considered ones, as explained in Section 2.1.3.

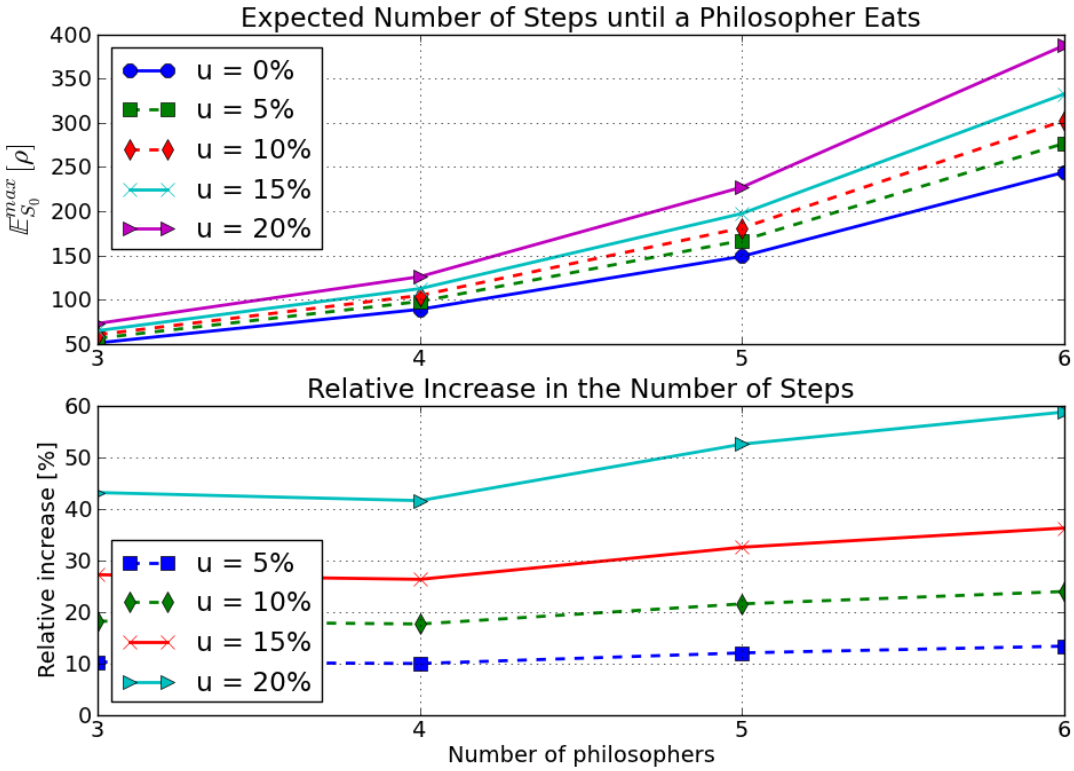


Figure 4.11: The figure shows the evolution of Equation (4.37) for increasing values of the number of protocol steps k for different models of uncertainties. The inset shows the relative deviation of the value of Equation (4.37) with respect to the model with no uncertainties.

While the results shown in Figure 4.10 guarantee that each philosopher will eventually be able to eat after becoming hungry, it might be interesting to determine what is the *expected* number of protocol steps after which this happens. We can evaluate such a protocol performance by equipping the Convex-MDP model of the protocol with a reward structure r defined as follows:

$$r = \begin{cases} r_s = 0 & \forall s \in S \\ r_a = 1 & \forall s \in S, \forall a \in \mathcal{A}(s) \end{cases}$$

Intuitively, reward structure r counts the number of steps of the protocol by assigning a cost of 1 to each transition. We can now compute the expected number of steps which a philosopher needs to wait before eating after becoming hungry, by evaluating the following property in PCTL syntax:

$$\boldsymbol{R}^{max}[\rho] := \mathbb{E}^{max}_{S_0}[\boldsymbol{C}\{Eating\}] \tag{4.37}$$

with initial states $S_0 = Sat(Hungry)$. In particular, we will compute such expected cumulative reward for models in which the uncertainty in the estimation of transition probabilities is represented using the interval model, while varying the uncertainty level u in the estimation of the

model transition probabilities. Figure 4.11 at the top shows the evolution of the expected number of steps as a function of the number of philosophers sitting at the table. As expected, the number of protocol steps steadily increases as the number of philosophers increases because of the higher contention of the shared resources. Moreover, the plot shows that adding uncertainty further *increases* the expected number of steps with respect to the scenario with no uncertainties ($u = 0\%$). At the bottom of Figure 4.11, we show the relative increase in the number of steps with respect to the scenario with no uncertainties, defined as:

$$\text{Relative increase} = \frac{\boldsymbol{R}^{\,max}_{u=*}[\rho] - \boldsymbol{R}^{\,max}_{u=0\%}[\rho]}{\boldsymbol{R}^{\,max}_{u=0\%}[\rho]}$$

with $* \in \{5\%, 10\%, 15\%, 20\%\}$.

A $\pm 20\%$ uncertainty can cause an increase up to 60% in the expected number of steps (for $n = 6$ philosophers), and even a $\pm 5\%$ uncertainty causes an increase higher than 10%. We conclude that an uncertainty in the transition probability distribution can have a drastic impact in the performance of the system. If this model was used to represent a real shared-resources stochastic protocol, we could infer that a misbehavior of one of the system agents, due for example to a fault, could substantially alter the expected system performance.

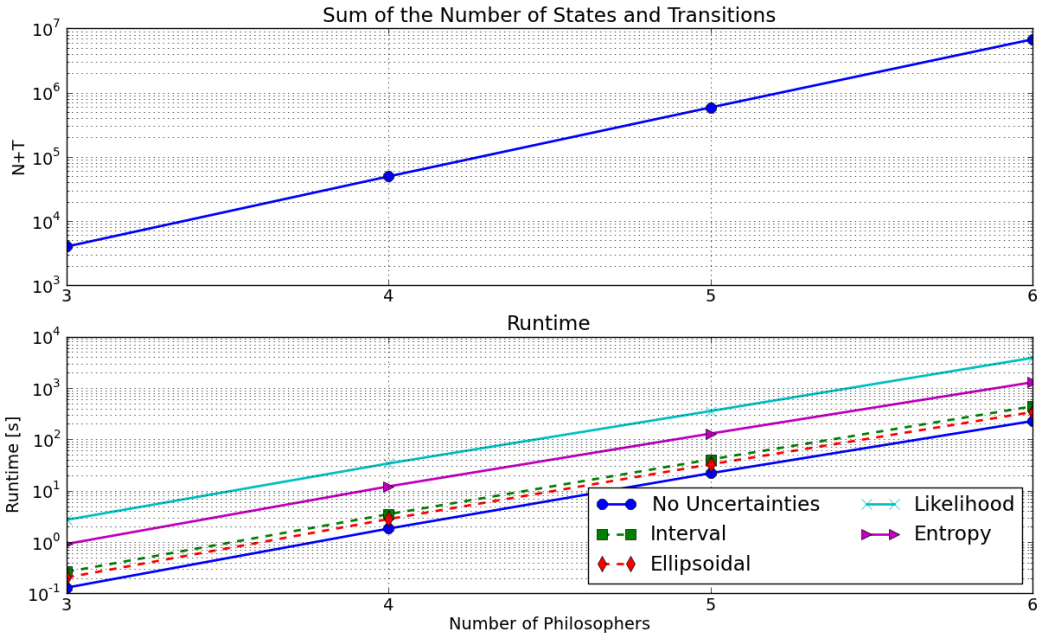


Figure 4.12: Runtime analysis of the proposed routine for the model checking of the *Bounded Until* operator. The figure shows the sum $N + T$ of the number of states and transitions in the Convex-MDP (top) and the runtime results for the different models of uncertainty for $k = 100$ steps.

Lastly, we evaluate the runtime performance of the routine to model check the *Bounded Until* operator while varying the number of philosophers at the table ($n \in \{3, 4, 5, 6\}$). We (arbitrarily) set $k = 100$. Figure 4.12 shows at the top that the size of the model, represented by the sum $N+T$ of the states and transitions of the Convex-MDP, increases polynomially with the number of philosophers sitting at the table (a straight line on the log axes). At the bottom, Figure 4.12 shows that also the runtime increases polynomially with the model size. The interval and ellipsoidal models run faster because the inner convex optimization problems can be solved using simpler atomic operations (sum and multiplication) than the likelihood and entropy models (logarithm and exponentiation). The experimental evaluations thus shows that also the routine for the verification of the *Bounded Until* operator scales polynomially with the size of the Convex-MDP.

Chapter 5

Formal Verification of the Performance of a Car Driver

In this chapter, we describe how we applied the proposed model-checking algorithm to the problem of automatically assessing the performance of individuals while driving a car. In particular, we will focus on how the driving performance gets influenced by environmental factors, e.g., the presence of an obstacle along the road, and by the level of attention of the driver, e.g., attentive or distracted by a text message on the phone. We first introduce the analyzed problem and motivate its relevance in applications ranging from the development of personalized teaching strategies to correct misbehaviors and the computation of personalized car insurance rates to, ultimately, the automated assistance to the driver in semi-autonomous cars. Secondly, we review related approaches proposed in the literature aiming to capture the performance and behavior of human drivers. We then present the Convex Markov Chain model that we developed to capture the behavior of individuals while performing complex maneuvers in a car, e.g., a double turn. Moreover, we show how to use convex uncertainties sets to capture in the model the inevitable inaccuracies introduced in the modeling of the intricacies of the human behavior. We conclude by reporting experimental results showing that it is indeed possible to automatically assess peculiar behaviors of individuals by applying the proposed approach.

5.1 Problem Description

In this section, we first present the main applications that motivated our research, we then introduce the problem of modeling the behavior of an individual while driving and, finally, we highlight the main contributions presented in this chapter.

5.1.1 Motivating Applications

The problem of modeling the behavior, intent and performance of a human subject while driving a car has long been studied, due to the relevance of the problem to applications ranging from teaching techniques for safer driving [83] and issuing of more effective driving regulations and norms [61] to, more recently, the personalization of car insurance rates [113] and the development of autonomous and semi-autonomous control techniques to reduce the number of fatalities [33].

The topic of driver behavior modeling lies at the intersection of multiple research disciplines, both in the engineering field and in social sciences. The psychology community has extensively researched the topic in order to develop the most effective teaching techniques for new and experienced (but reckless) drivers [83]. In particular, it is acknowledged that drivers' attitudes while learning may predict their future performance on the road after being licensed. In the current "era of rage" [83], it is thus fundamental to *quantitatively* study the consequences of the driver emotional status and attitude on the driving performance, so that it is possible to educate drivers to follow current regulations and avoid them to express their anger (or other feelings) while driving, by performing dangerous maneuvers.

Another area where the modeling of driver behavior has long been essential is in the decision process of authorities and regulators for transport safety, where the consideration of driver performance is used in setting standards and rules governing new and future regulations of the vehicle control systems, road infrastructures and traffic management [61]. Similarly, models of the driver behavior are necessary for the study of accidents and investigation of root causes.

As a third application of the proposed technique, we mention that a few car insurance companies in the United States (among which Progressive, State Farm and All State) have recently started voluntary programs in which drivers accept to have a small device installed into their car to monitor how they *actually* drive [113]. Instead of profiling drivers based just on the traditional factors – age, location of residence, history of accidents and traffic violations – these car insurance companies thus allow "usage-based" or "pay-as-you-drive" insurance rates. Noticeably, at the present time, the insurance rate can only *decrease*, but never increase, for the drivers participating in these programs. The main limitation of such an approach is the need for a large amount of data to accurately profile the driving habits of the participants. While, potentially, the monitoring devices could just be left constantly on, concerns have been raised about the privacy implications of such a technique, since, for example, also the location of the car (through a GPS) is relayed to the insurance company, which can thus know where the driver is at all times. An alternative approach could instead employ a relatively short *training* phase (e.g., 3 months), in which measured data get collected, followed by an off-line *analysis* phase, in which mathematical models of the driver behavior, trained with the collected data, are analyzed to estimate the likelihood of accidents and infer the driving patterns of the subject under analysis, without requiring an explicit monitoring of the driver.

As a final application for models of the behavior of human drivers, we report that there has been an increasingly high interest in the control community inside and outside academia for the development of techniques to automatically assist the driver while performing a maneuver. Four states in the United States (Nevada, Michigan, Florida and California) have already authorized test-

ing of autonomous vehicles on public roads, and several European and American companies have started equipping their high-tier models with control units capable of performing simple maneuvers automatically, like *distance keeping* and *lane changing*. As a further example, we note that the "automatic gearbox" of certain vehicles is an automated control systems that adapts dynamically and independently to different *driving styles*, measured through intrinsic evaluation of behavioral variables, such as rate of accelerator pressure, overall speed, etc. Another example is the system that manages the availability of in-vehicle information systems (IVIS), such as telephones or radios. In this case, certain IVIS managers adapt to the environmental situations, by inhibiting or discouraging the use of certain IVIS in risky situations.

Although the final goal is the deployment of fully autonomous cars (like the *Google Autonomous Car* [117]), the human driver will be likely required to perform occasional tasks in the first deployments of these systems, for example to counteract to unforeseen situations. The study of the interaction between humans and the automated system will thus be of paramount importance, for example to understand when and how to release the control of the vehicle back to the human in the presence of a sudden threat (e.g., a sharp turn on a slippery road) so that he or she has time to react. Models of the human behavior are thus required to develop the appropriate interface to the machine [27] and control algorithms are being developed to increase the accuracy of the maneuver and assess the presence of threats, to, ultimately, enhance the safety of the passengers [12, 95, 165, 166, 169].

5.1.2 Problem Description

The rapid sequence of "technology revolutions" that have characterized the last 40 years have pushed the execution of ever more complex tasks from the human operator towards automation. However, even in the cases of totally automatic systems, it is not possible to avoid the assessment of the human-in-control principle, as the (possibly remote) operator of the fully automatic systems remains to be accounted for in the design and development processes. For these reasons, the development of techniques to appropriately account for the user and controller of technologically advanced systems is one of the most relevant issues for the design of new products.

In parallel to the development of the technology, the need to account for the behavior of the human being has progressively evolved from considering the human subject as a *manual controller* to the concept of *human supervisor* of automatic control systems. The evaluation of the behavioral performance has been replaced by the analysis of cognitive and mental processes. In other words, the demand for modeling manual and behavioral activities has been replaced and combined with the need for modeling cognition.

The main challenge in capturing the human behavior in a mathematical model is that any mental process is fairly contextual and substantially different from one person to another. The initial linear models, capable of only capturing simple tasks, have thus been gradually replaced by nonlinear and even probabilistic models, based upon artificial intelligence (AI) principles, such as artificial neural networks, genetic algorithms and stochastic decision processes [112]. The problem of modeling the human behavior becomes even more challenging if we consider a complex behavioral task such as vehicle driving.

Usually, car driving is described as a task containing three different levels of demands [143]. At the *strategic* level, the driver plans the general route of a journey. For example, the driver chooses the route and the transportation mode and evaluates resulting costs and time consumption. At the *tactical* level, the driver has to perform maneuvers, for instance, turning at an intersection or accepting a given distance to the car in front. Finally, at the *control* (stabilisation) level the driver has to execute simple (automatic) action patterns, which together form a maneuver, for example, changing the gear and turning the steering wheel.

Following this division of the driving task, models of the performance of the driver are usually assigned to three different levels: knowledge-based, rule-based and skill-based behavior [129]. *Skill-based* behavior is described as data-driven, meaning that skills (e.g., lane keeping) are performed without conscious control and use of attention resources. They are immediate and efficient. *Rule-based* behavior, on the other hand, occurs under conscious control and requires attention (e.g., stopping at a traffic light). Therefore, it is less immediate and efficient. Finally, *knowledge-based* behavior involves problem solving and is relevant when it is not clear how to act in a specific situation (e.g., how to reach the desired destination). Thus, an important aspect of knowledge-based behavior is that reasoning is required.

Given the complexity of the analyzed problem, any successful strategy to cope with it needs to focus on modeling only the aspects of the driver behavior that are relevant to the specific application in mind, without aiming to create a single omni-comprehensive model. In the rest of the chapter, we will describe a model that captures the skill-based and rule-based behaviors of individuals. Indeed, we will create models of the performance of the driver while performing complex maneuvers, e.g., a double turn, which require conscious control of the vehicle and of the surroundings. Moreover, we will show how to personalize the performance predicted by the model to the specific characteristics (or *skills*) of each driver, which are performed unconsciously and differ from subject to subject.

The overall aim of our modeling effort will then be to predict the future trajectory driven by the individual under analysis over a relatively long time horizon (30 seconds to one minute). Our approach makes this prediction based on the analysis of the history of driving patterns of the human subject, on his or her mental state and on the surrounding environmental conditions.

We believe that such an analysis targets the needs of the applications described at the beginning of the section. Teaching techniques indeed focus on how to perform complex maneuvers to correct possible misbehaviors peculiar to each driver. An analysis of the common behaviors across a large population of individuals might help in shaping future road regulations, e.g., in setting the maximum speed limits. Car insurance rates can be computed for each individual based on the history of his or her driving patterns and on the likelihood of such patterns to cause accidents. Finally, an accurate prediction of the car trajectory lies at the core of any automated control system aiming to assist the driving task. A further extension of our approach to capture also knowledge-based behavior will further benefit these applications and is left as future work.

Finally, we note that a common thread of research across many of the applications described above is the study of the impact of driver distraction on the performance of the driver [10, 20, 42, 144]. This includes distraction caused by cell phone calls and text messaging, which account for $22 - 50\%$ of all accidents [90]. From an analysis perspective, the changes of driving performance

between attentive and distracted drivers have been researched to increase people awareness on the possible fatal consequences that distracted driving can cause. From a control perspective, potential solutions to the driver distraction problem rely on semi-autonomous or "human-in-the-loop" control techniques [111], some of which try to predict the car trajectory based on estimations of the driver behavior and actively take control of the car if the probability of threat is higher than a given threshold. For instance, existing techniques perform a braking maneuver if a collision is predicted. To correctly take human actions into consideration before intervention, modeling the driver behavior is thus of crucial importance.

In summary, the modeling approach presented in this chapter gives *quantitative* techniques to *exhaustively* evaluate the performance of an individual while driving. Moreover, we give particular emphasis to the study of the variation of driving performance for different attention levels. The main contributions presented in this chapter are highlighted in the next section.

5.1.3 Contributions

As a **first contribution**, we developed a novel *probabilistic model* of the driver performance, which predicts the driven trajectories using a Convex Markov Chain (Convex-MC) model, i.e., a Markov chain in which the transition probabilities are only known to lie in convex uncertainty sets, as presented in Chapter 2. The prediction of the trajectory is based on the future environment surrounding the car, the attention state of the driver (i.e., attentive or distracted), and the history of steering maneuvers for a given individual, which we collected using a car simulator [35]. For each environment and attention state of the driver, the model predicts a set of trajectories for the subsequent time interval based on empirical observations of past behaviors. These predictions are then used as transition probabilities within the stochastic model of the driver performance. Due to the ambiguity that inherently affects the measured data used to estimate the transition probabilities, we allow state-transition probabilities to be expressed in terms of uncertainty sets, which can be rigorously defined based on statistical techniques. This framework allows a more conservative prediction of the driver behavior and gives guidelines to the model developer to determine when the collected data are statistically relevant to correctly infer properties of the system.

Due to the criticality of the system under consideration, formal techniques to verify properties of the constructed model are required to rigorously assess the validity of the model and give guarantees of its safety and liveness. As a **second contribution**, we show how to analyze *quantitative* properties of the Convex-MC model of the driver expressed in Probabilistic Computation Tree Logic (PCTL), using the polynomial-time model-checking algorithm presented in Chapter 4. Our main focus is to quantify the effects of different attention levels on the quality of driving, by formally analyzing the driver behaviors while they are either attentive or distracted. PCTL is a suitable choice because it allows to express *quantitative* properties of a system, as opposed to other logics, e.g., LTL, which only allow *qualitative* properties. For example, we aim to determine whether "the maximum probability of exiting the road for a distracted driver is higher than 90%", while LTL would only allow to inquire whether "eventually a distracted driver will exit the road", a property that would trivially be always true for some of the executions of the model, without giving insight about how likely the event would actually take place.

Remark 5.1. *We have chosen to study the problem of modeling the behavior of car drivers (among other possible topics) because the number and variety of applications described in this section shows that the topic is a highly fertile field of research, where contributions from multiple disciplines are still needed in order to reach an omni-comprehensive understanding of the subject. In particular, quantitative techniques to analyze the system dynamics are still not fully developed, and the problem of estimating the long-term evolution of the system (30 seconds to one minute) is still far from having been solved, due to the variety and complexity of the events that can happen in such a long time span. We believe that the* stochastic *modeling techniques presented in this dissertation represent a promising approach to analyze the problem, since they allow the abstraction of complex deterministic dynamics with simpler probabilistic evolutions of the system and because they are also equipped with formal techniques to capture the uncertainty in the estimation of the model parameters.*

Moreover, the proposed fast (polynomial) implementation of the model-checking algorithm offers a scalable technique that allows the rapid *analysis of* large *models. These characteristics are necessary to capture the complexity of the system under analysis both when the models are used for off-line applications, like insurance rate calculation or the development of teaching strategies, and, even more so, when the application is the automated real-time assistance of the driver.*

5.2 Related Work

Models of the behavior of human subjects in the context of car driving can be classified into two main categories: *cognitive* models and *engineering* models (also referred to as non-cognitive or probabilistic). The model described in this chapter belongs to the second category of *engineering* models. Nevertheless, to give a full overview of the different approaches used in the literature, in the following we will describe each category and give examples of models developed using each strategy.

5.2.1 Cognitive Models

A cognitive model is an approximation of the human cognitive processes, i.e., the mental processing of information, the application of knowledge, and the selection of preferences, which involve using one's working memory, comprehending and producing language, calculating, reasoning, problem solving, and decision making.

Cognitive *models* can be developed within or without a cognitive *architecture*. In contrast to cognitive architectures, cognitive models tend to be focused on a single cognitive phenomenon or process (e.g., performing a driving maneuver), how two or more processes interact (e.g., the monitoring of other approaching cars), or to make behavioral predictions for a specific task (e.g., how to take over another slower car). Cognitive architectures tend instead to be focused on the structural properties of the modeled system, and help constrain the development of cognitive models within the architecture, i.e., they provide the *methodology* and *building components* that can then

be used to generate the cognitive model of a specific task. Some of the most popular architectures for cognitive modeling include Soar [106] and ACT-R [9].

A cognitive model in Soar is created by defining an *agent*, which represents the modeled individual. All the knowledge in a Soar agent is then represented as *if-then rules* (or "productions, in Soar terminology). The problem of modeling the behavior of a human while performing a task can be roughly described as a search through the problem space (the collection of different states which can be reached by the system at a particular time) for a goal state (which represents the completion of the task). This is implemented by searching for the states which bring the system gradually closer to its goal. Each move consists of a *decision cycle*. This is composed of an *elaboration* phase, in which a variety of different pieces of knowledge bearing the problem are loaded to the agent's working memory, and a *decision* procedure, which weighs what was found on the previous phase and assigns preferences to ultimately decide the next action to be taken.

All decisions are made through the combination of relevant knowledge at run-time. In Soar, every decision is based on the current interpretation of sensory data, on any relevant knowledge retrieved from the long-term memory and on the contents of the working memory, which contains all of a Soar agent's dynamic information about its world and its internal state and which can be expanded by knowledge created by prior problem solving.

The ACT-R architecture offers the following building blocks to create a cognitive model: modules, buffers and pattern matchers.

Modules are classified in perceptual-motor modules, which take care of the interface with the real world (or with a simulation of the real world), and memory modules, which describe the different layers in which the human memory is organized. The most critical perceptual-motor modules in ACT-R are the visual and the manual modules, since those are the most used sensing and actuating interfaces used by human beings. Further, there are two kinds of memory modules in ACT-R: *declarative* memory, which stores facts such as "Washington, D.C. is the capital of the United States", or "2+3=5", and; *procedural* memory, made of productions, i.e., knowledge about how we do things (e.g., knowledge about how to drive, or about how to perform addition).

A model written in ACT-R accesses its modules through buffers. For each module, a dedicated buffer serves as the interface with that module. The content of the buffers at a given moment in time represents the *state* of the ACT-R model at that moment.

Finally, the pattern matcher searches for a *production that matches the current state* of the buffers. Only one such production can be executed at a given moment. That production, when executed, can modify the buffers and thus change the state of the system. Thus, in ACT-R cognition unfolds as a succession of production firings.

We conclude this section by mentioning how to evaluate the quality of a cognitive model, since the "outputs" of such a model are made of a sequence of mental processes, and not of "mathematical" quantities which would be more familiar to the engineering community. According to the creators and developers of these cognitive architectures [9], the ultimate metric to determine whether a cognitive model properly captures the dynamics of the human mental process while performing a task is to compare its "results" to the behavior of a human subject in terms of the *time*

to perform the task and the *accuracy* in executing the task.

5.2.2 Engineering Models

Engineering models, also known as non-cognitive or probabilistic models, abstract the intricacies of the mental process using mathematical formalisms. We further subdivide these models into two categories.

In the first category, we collect models whose goal is to infer the *intentions* of the human driver (e.g., to change the lane) in order to determine the optimal way to assist him or her while performing the maneuver. Since it would be impossible to succinctly capture all the complexity of the human behavior in a deterministic fashion, these models usually employ probabilistic formalisms, like Markov chains [134], Hidden Markov Models [97], discrete-event stochastic simulations [49] and many others [4].

A second category of engineering models aims instead to solve the complementary problem of predicting the future *trajectory* of the vehicle in order to issue control commands to guarantee the safety of the performed maneuver (e.g., braking if the distance from the car in front is not maintained). In this context, often deterministic models with a short-term receding horizon are used [134, 169].

The approach proposed in the rest of the chapter combines features of both the categories described above, by using short-term deterministic predictions of the vehicle trajectory within a long-term probabilistic framework, capable of capturing the full evolution of a complex maneuver.

A similar approach mixing characteristics of both categories was described by Pentland and Lin [134]. This paper proposes to model the behavior of a car driver using a discrete-time Markov chain in which each state has continuous dynamics estimated using a Kalman filter. The usage of short-term continuous models is motivated by the need of capturing the smoothness and continuity of the human behavior while performing short actions. Longer-term discrete transitions among states of the Markov chain are instead used to capture the richness and variability of the human behavior, which is recognized as a highly non-linear system that cannot be predicted by a linear filter on a longer time frame. The required information to infer the actual state of the driver can be captured using measurements. Probabilities to transition among states of the Markov chain are computed by ranking the bank of Kalman filters representing the system dynamics depending on how well they predict the collected measurements. In order to avoid an explosion of the complexity of the transition estimation problem, the authors suggest to preprocess the Markov chain model. Indeed, not all states can transition to any other state, since the behavior of human drivers usually follows sequences of actions while performing a maneuver. For example, to change the lane, the driver will first check in the back mirror to make sure that the destination lane is empty, he will then activate the light signals and finally stir the wheel to perform the maneuver. Such a simplification substantially reduces the candidate next states when analyzing a complex maneuver, and contributes in assigning correct weights to the candidate next states. Finally, the authors suggest to decompose the human behavior into *modes*, each characterized by its own dynamics. For example, the authors distinguish between *relaxed* driver and *tight* driver.

5.3 Proposed Model

In this section, we give details on how we created a probabilistic model of the performance of individuals while driving a car. The primary goal of the modeling effort is to give an accurate prediction of the future trajectories of the car over a medium-term horizon (30 seconds to one minute), so that the automated system has time to either warn the driver about the possible presence of threats or even take the control of the car to perform defensive actions (e.g., braking) and guarantee the safety of the maneuver. In order to achieve this goal, it is fundamental to be able to capture and analyze in the model the whole execution of *complex* maneuvers, e.g., a sequence of a right and a left turn, and to study how decisions made early in the maneuver can affect later decisions. Following a growing consensus from recent literature on the topic [169], our approach postulates that the driver state, e.g., attentive or distracted, and the environmental conditions, e.g., the presence of an obstacle along the road, must be considered to increase the accuracy of the prediction. By means of this comprehensive system description, we believe that we can indeed reach the level of accuracy in the prediction of the driver performance that is required in the development of automated analysis and control algorithms to assist drivers in their task.

As it will be described in details in the following sections, we collected empirical observations of the behavior of multiple drivers in a simulation environment, in order to create personalized driver models tuned to the characteristics of each individual. Such a personalized tuning is fundamental to produce analysis and control results that can indeed address the peculiarities of the driving style of each subject. Conversely, a generic modeled averaged across a population of individuals could not be easily used to predict the performance of any of the subjects, due to the high diversity (variance) of the human behavior. Once the data are collected, they get categorized to form a library of *atomic* actions performed by the driver, e.g., driving straight. These atomic components get then composed together to form the model of the arbitrary maneuver to analyze. Finally, time and space of the execution of the maneuver are discretized to form a discrete time Convex Markov Chain (Convex-MC) representing the maneuver, so that it is possible to quantitatively analyze the model properties adopting the model-checking algorithm presented in Chapter 4. An arbitrary level of accuracy in the model can be obtained by appropriately choosing the discretization step, at the expense of longer runtime of the model-checking algorithm. In our experiments, we chose a time step of $1.2s$, which is approximately the human reaction time while driving, to make sure that our assumptions hold with negligible loss in terms of accuracy.

The following section describes the experimental setup and methods used to develop the driver model.

Remark 5.2. *The developed approach allows the analysis also of complex maneuvers that were* not *explicitly performed by the individuals while driving in the car simulator. The analyzed maneuvers get instead* composed *from a* library *of atomic actions, e.g., driving straight or turning left, which get pre-characterized for each driver starting from the empirical data. Such an approach is fundamental to keep the training data set and time limited, while still being able to analyze* a posteriori *the rich variety of more complex maneuvers and scenarios that any driver needs to face on a daily basis.*

5.3.1 Data-Driven Characterization of the Library of Atomic Actions

The study of short-term (atomic) actions of drivers has been long researched in the literature [153, 169], because it is useful to implement automated correction maneuvers to assist the driver, i.e., trajectory correction while making a turn. Briefly, these methods aim to infer the future trajectory of the vehicle on a short-time horizon, e.g., one second. While our ultimate goal is the analysis of *long-term* complex maneuvers to estimate the driver intensions and to anticipate possible threats, we start from the characterization of these atomic actions because they will then become the building blocks to create the stochastic model of the complex maneuver. Indeed, short-term actions can be represented with high accuracy also by employing simple linear dynamic models of the vehicle trajectory, while we will see that we need to use a stochastic framework to analyze longer-term maneuvers to account for the randomness intrinsic to the system evolution in the long-term. This two-tier modeling approach can provide reasonably accurate results if the following assumption, which is commonly used in the literature [169], holds:

Assumption 5.1. ***Repeatability of the Driver Habits.*** *It is assumed that humans act in a repeatable manner, meaning that they are likely to drive in a similar way if the environmental conditions are analogous with only some inconsistency due to environment variations and human errors.*

We train the models of the atomic actions with measured data collected by observing individuals while driving a car on CarSim, a standard car simulation software used by industry [35]. We use the Microsoft Kinect [119] as input sensor to observe the driver pose in real-time, and collect data on the observed steering angle every 30ms, to reconstruct the driven trajectories. Over 20 subjects, both male and female with ages ranging from 20 to 80 years and with varying levels of experience, were asked to drive for an hour through four predetermined courses to collect training data and observe peculiar driving habits in different external environments and for different levels of driver attention. In particular, to simulate distracted driving, we provided the subjects with a phone where an application was installed to prompt them to answer the call or text while continuing to drive. More details about the measurement setup and the data collection procedures are available in the work by Shia et al. [153].

In its simplest instance, a library of atomic actions would be populated of models corresponding to, for example, driving on a *straight* segment of road, and making a *right* or *left* turn. Such an approach would on the other hand result in an over-simplification of the dynamics of the driving task. In order to capture the variety of scenarios a driver needs to face on a daily basis, we instead create the models of the atomic actions using the approach described by Shia et al. [153] and Vasudevan et al. [169]. According to this method, even during an atomic action, the driver behavior is dependent on particular *modes*, which are determined by:

1. the future *environment* external to the car, e.g., an obstacle along the road, and;
2. the *attention* state of the driver, e.g., attentive or distracted by a text message.

We selected the four training courses to test different driving scenarios. In particular, we asked the driver to drive on:

- a course with no obstacles and without being distracted by a text message (scenario 1);
- a course with no obstacles but while being distracted by a text message (scenario 2);
- a course with an obstacle and without being distracted by a text message (scenario 3);
- a course with an obstacle and while being distracted by a text message (scenario 4);

Each scenario was composed of a sequence of straight segments of road, left and right turns, and, for the last three scenarios, the obstacle or the distraction were introduced only for part of the course. In order to associate the large body of collected data to the corresponding mode where the data were collected without making unnecessary assumptions, the data is automatically clustered using the k-means algorithm [73], which allows for flexibility in determining which data belong to which mode in an unsupervised manner. This also allows to test the quality of the characterization of each mode, since modes that were not assigned enough data by the k-means algorithm had to be re-characterized with additional training data in order to be faithful in predicting the behavior of the driver. In the setup utilized in this case study, these clusters or modes are created from the following data sets:

1. **Driver Pose.** This data set contains the past two seconds of observations of *skeleton* data of the individual, specifically the positions of the wrist, elbow, and shoulder joints. These data are used to infer automatically the attention state of the driver. For example, if the algorithm detects that the driver is turned watching outside the simulator screen, we associate the corresponding steering angle measurements to distracted driving.
2. **Environment Estimation.** This data set contains a feature vector for the future four seconds of the outside environment, including road bounds and curvature, obstacle locations, and the car's deviation from the lane center.

The clustering algorithm uses a tiered structure. It first clusters the steering angle data based on the vector of environmental modes, and it then further clusters the results of the previous iteration using the vector of driver pose modes. Empirical considerations suggested us to use 150 clusters, $k_1 = 50$ environment clusters at the first tier and $k_2 = 3$ pose clusters at the second tier, to achieve the desired level of accuracy in the model [153]. This allowed us to compare driver behaviors in matching environmental conditions for different driver attention states.

Once the data has been clustered into modes, the future 1.2 seconds (a time frame comparable to the human reaction time) of the driver steering angle inputs associated to each mode become the *prediction* for that scenario. The predicted steering angle inputs where passed to a linear model of the vehicle dynamics to generate the future vehicle trajectories [169]. In our setup, we assume that the driver is driving at approximately 60 miles per hour. This assumption does not limit our approach and is appropriate in a highway scenario. A sample trajectory set is shown in Figure 5.1. This graph shows the change of lateral direction of each trajectory in meters from a given initial condition for the future 1.2s. As it is apparent from the figure, some of the trajectories stay within the lane, some drift outside of the lane and some others diverge to the left lane.

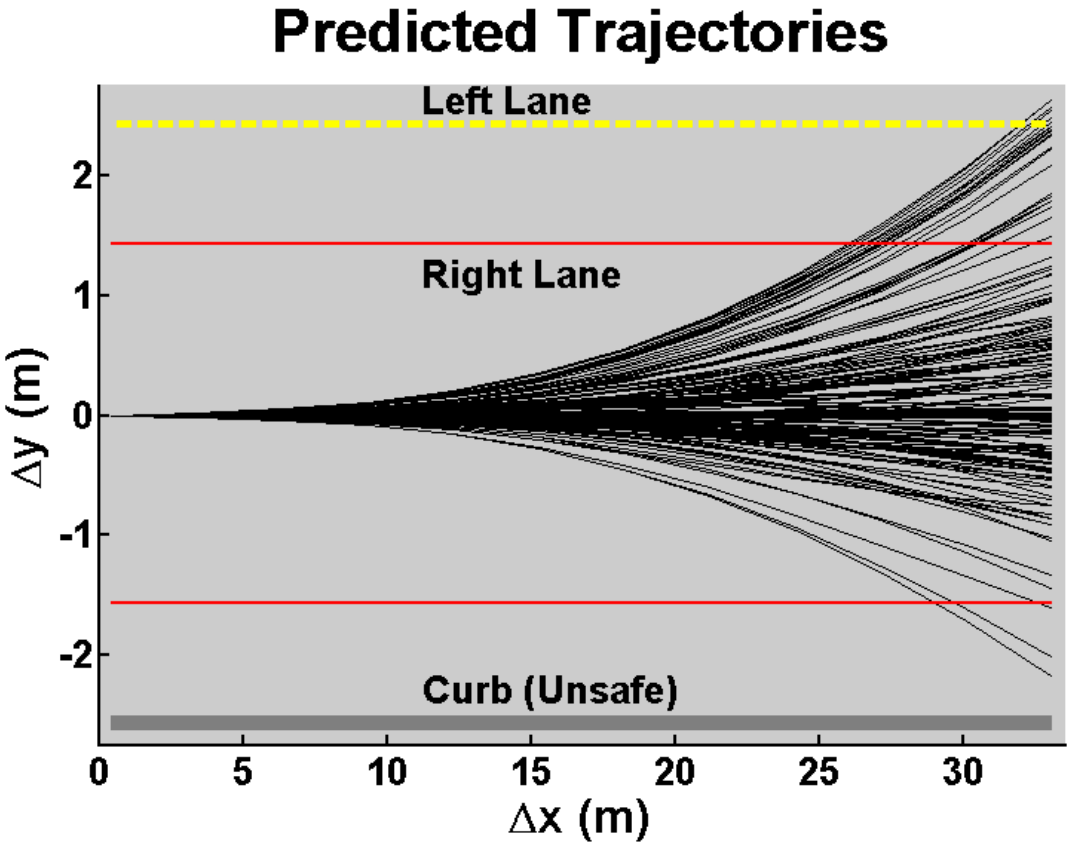


Figure 5.1: Example of an observed set of trajectories showing the change in lateral and longitudinal directions with respect to the center of the car (at $\Delta y = 0$). The bottom red line marks the beginning of the curb, while the yellow line marks the beginning of the left lane.

We can now create the library of atomic actions. We do so by processing each trajectory set and by thresholding the end-points of the trajectories. If most end-points have an horizontal displacement from the initial point smaller than half the lane width, we associate the set to the atomic action *driving straight* (as for the set of trajectories shown in Figure 5.1). Conversely, if most trajectories end up to the left (right) of the lane, we associate the trajectory set to the atomic actions *turning left* (*right*). Such an approach allows us to infer the driver's intentions while driving. We further note that our library of atomic actions is in fact composed of many flavors of each action, in which not only the direction is specified but also the environmental conditions and the driver attention state in which the corresponding trajectories were driven. For example, the atomic action associated to the data set shown in Figure 5.1 could be expressed as: "Driving straight in the presence of an obstacle and focused on driving". Indeed, we see that most trajectories do keep the lane, but sometimes the car tends to drift towards the left lane or even the curb, translating into dangerous behaviors. Overall, this approach creates a driver model that is able to continuously predict the future vehicle trajectories and, by extension, the behavior of the driver, as it moves through a given environment with a given attention level.

In conclusion, the models of the atomic actions developed so far answer the question: "Given the mode the driver is currently in, how will he or she drive in the next $1.2s$?" By identifying the modes using observations of the environment and of the driver attention level, the associated future steering angle inputs can be used as a prediction of the driver behavior.

5.3.2 Stochastic Modeling

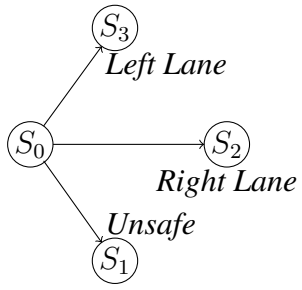
In this section, we show how to create a stochastic model of the driver behavior capable of representing complex maneuvers. In particular, we compose the atomic actions tuned to the characteristics of each individual as introduced in Section 5.3.1 to form the model of the complex maneuver. Each atomic action is annotated with *labels* describing the environmental and driver attention modes associated to it. We instantiate atomic actions from the library as *states* of our model. *Transitions* between instances of atomic actions are then defined in a probabilistic fashion by considering the predicted behavior within each atomic action.

To illustrate our methodology, we explain how to convert the trajectory data in Figure 5.1 to the model shown in Figure 5.2. We start by associating the state S_0 to the atomic action represented in Figure 5.1. Given the environmental information associated to the atomic action, we know that the trajectory starts in the center of the right lane of a two-lane road. Using standard values for the size of the car and width of the lane, we classify the trajectories within the set in three subsets, *lane changing*, *lane keeping*, or *drifting*, depending of the final $y-$coordinate. Moreover, the trajectories that exit the safe region of the road toward the curb are identified as *Unsafe*, those that remain in the middle of the road are marked *Right Lane*, while those that tend towards the left lane are identified as *Left Lane*. We can now associate a new state to each of these maneuvers, S_1, S_2, S_3, representing the three locations where the car may be in the next time step.

Finally, the empirical probabilities to perform each maneuver (and thus transition into a new state) are calculated by examining the percentage of trajectories within the cluster that terminate in the corresponding region (see labels in Figure 5.1). The second column of the table in Figure 5.2 reports the computed probabilities for the example under analysis.

One of the main limitations of the described approach stems from the fact that a limited set of training data is used to model the wide variety of the behaviors of an individual while driving a car. We now introduce uncertainty sets around the derived probabilities to accommodate for estimation errors and give quantitative means to express the quality of the estimation of the driver behavior starting from empirical measurements. In particular, the size of the uncertainty set will vary according to the level of confidence in the collected measures (a larger set indicates less confidence). In our experiments, we used both the interval and likelihood uncertainty models. The interval model is intuitive to understand and will be used as a baseline for our quantitative analysis. Given a confidence level $0 \leq C_L \leq 1$ in the measurements and an empirical probability p_0, we compute the transition probability interval $[C_L \times p_0, (2 - C_L) \times p_0]$. We also use the likelihood model of uncertainty because it is less conservative than the interval one and it is largely used in the scientific community to represent uncertainties on data collected through measurements. Using classical results from statistics [125], we can compute the value of parameter β_s^a from Set (2.3) corresponding to a confidence level C_L. In particular, $C_L = 1 - cdf_{\chi_d^2}\left(2 * (\beta_{s,max}^a - \beta_s^a)\right)$, where

$cdf_{\chi^2_d}$ is the cumulative density function of the Chi-squared distribution with d degrees of freedom ($d = 3$ in this example because there are three possible next states) and $\beta^a_{s,max}$ defined as in Section 2.1.3.2.



Transition	Probability	Transition Probability Interval
$S_0 \to S_1$	0.02	[0.019,0.021]
$S_0 \to S_2$	0.935	[0.888,0.982]
$S_0 \to S_1$	0.045	[0.043,0.047]

Figure 5.2: Example of a Convex-MC modeling a simple maneuver. The intervals expressing the uncertainty in estimating the transition probability among states from a trajectory set are collected in the third column in the table on the right. We used a confidence level $C_L = 95\%$.

We are now ready to formally describe the created stochastic model using the formalism of Convex-MCs[1], as introduced in Chapter 2. In our Convex-MC models, $\mathcal{M}_C = (S, S_0, \Omega, \mathcal{F}, X, L)$, we let S represent the set of instantiated atomic actions, and $S_0 \in S$ represent the set of initial states. We assign a subset of the labels collected in the set Ω to each atomic action to encode the environmental and driver attention modes. For example, labels can mark atomic actions performed on the *right* or *left lane*, and associate them to a right or left *turn* or to a *straight* segment of road. Labels are also used to mark *Safe* (*Unsafe*) states if they are within (outside) the road boundaries. We also label a state as *Accelerating* or *Braking*, if the value of acceleration is above or below a chosen threshold, and *Swerving* if the number of swerving trajectories is above a threshold (swerving marks potentially dangerous driving). We label the goal set of states as *Final*, to mark the end of the complex maneuver. Finally, we use the labels *Attentive* and *Distracted* to mark the corresponding driver attention mode. The full set of labels used in this case study is:

$$\begin{aligned}\Omega \;=\; \{&\textit{Right Turn, Left Turn, Straight, Right Lane,}\\ &\textit{Left Lane, Safe, Unsafe, Braking, Accelerating,}\\ &\textit{Swerving, Final, Attentive, Distracted}\}\end{aligned}$$

The set $\mathcal{F}$ collects all the convex sets of uncertain transition probability distributions, each encoding the chosen confidence level and uncertainty model. Mapping $X : S \to \mathcal{F}$ associates each state with the corresponding convex uncertainty set of probability distributions to the next states. Finally, the labeling function L maps each state to the corresponding set of labels in Ω.

Because the atomic actions used to create the stochastic model contain information about the driver attention state, the procedure described here can be applied to both attentive and distracted

[1]In this case study, we use Convex-MCs and not Convex-MDPs because we do not model any decision process during the execution of the maneuver, and our model just transitions stochastically from one state to the other. We note that Convex-MCs can be in fact interpreted as Convex-MDPs with only one action per state, i.e., $M = |A| = 1$, so all the results presented in Chapter 2 and Chapter 4 still hold.

drivers. This allows us to determine *quantitative* changes in behavior for a specific driver depending on his or her attention level.

Finally, we note that the presented modeling technique can be iterated to create a stochastic model of the behavior of the car driver while performing an arbitrary maneuver, under the assumption that the library of atomic actions contains enough elements to reconstruct the whole maneuver. We introduce a more elaborated example in Section 5.3.3, where we describe the procedure to create the model of a driver performing a double turn. Such a complex maneuver was not explicitly performed by any driver during the training session, and instead the stochastic model was created by composing sub-blocks from the library of atomic actions.

5.3.3 Model of a Complex Maneuver

As a more elaborated example of our modeling approach, we created a Convex-MC model of the road segment shown in Figure 5.3. The road consists of the sequence of sections: straight, right turn, left turn and another straight, so we can interpret the maneuver as a double turn. States are assigned to different locations on the road. Each state is an atomic action that is chosen from the library of atomic actions characterized for each driver. We let $\{S_0, \cdots, S_{15}\}$ represent the set of states. S_0 is the initial state, and $S_i = \{D_i, A_i\}$ for $i \in \{0, 1, 2, 7, 8, 11, 12\}$ represent a set of two different states, one for distracted driving and one for attentive driving. We labeled as *Attentive* the states where the human driver's pose suggests that both hands of the driver were on the steering wheel. States are labeled as *Distracted* if the human pose suggests that the driver was holding a phone or sending a text message.

We created two different models to represent scenarios with (without) an obstacle on the road just before the right turn (Figure 5.3 on the left and right, respectively). In the scenario with the obstacle, some trajectories in S_0 terminate on the left lane, represented by state S_1. With no obstacle, trajectories in S_0 instead either keep the right lane or drift outside of the boundaries to an *Unsafe* state. States $\{S_3, S_4, S_5, S_6, S_9, S_{10}, S_{13}\}$ correspond to *Unsafe* states since they are all out of the boundaries of the road; the rest of the states are *Safe* states. States $\{S_{14}, S_{15}\}$ are marked as *Final* states. The *Right Lane* and *Left Lane* labels correspond to states $\{S_0, S_2, S_8, S_{12}, S_{15}\}$ and $\{S_1, S_7, S_{11}, S_{14}\}$, respectively. The rest of the labels in Ω depend on the specific driving style of each subject. For example, some individuals tend to break more often while performing the double turn, while others drive more smoothly. These differences will affect the results of the analysis of the behavior of each individual driver, as it will be shown in Section 5.4. Probability distributions for transitions between states are assigned as we discussed in Section 5.3.2. In fact, Figure 5.2 shows the transition probability distribution from S_0 to S_1, S_2 and S_3 in Figure 5.3 (left), in the case of attentive driving. The comparison and analysis of the two models in Figure 5.3 are discussed in Section 5.4.

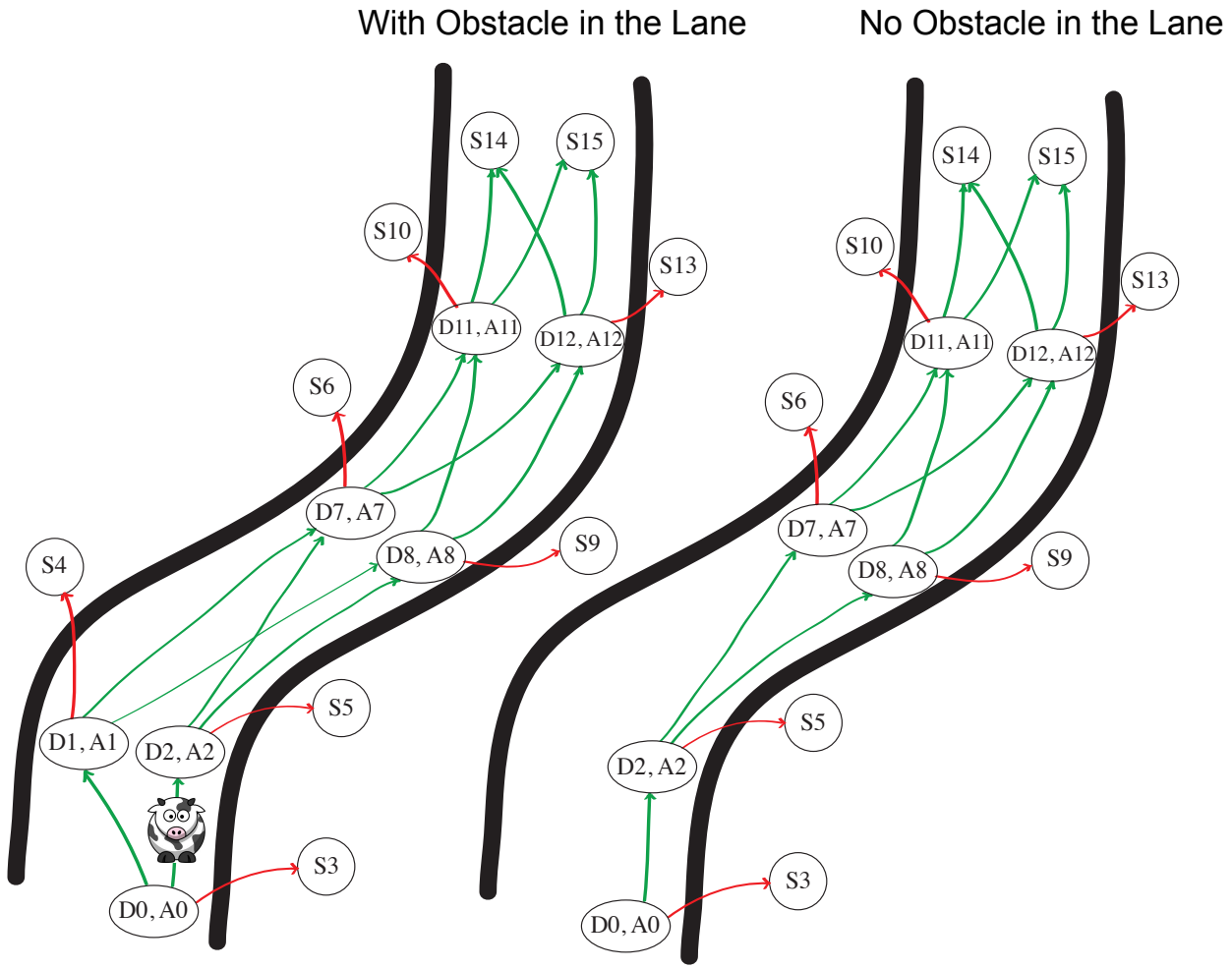


Figure 5.3: Convex Markov Chain representing a double turn. States $\{D_i, A_i\}$ represent a set of two states one labeled *Distracted* and the other *Attentive*. We created two models to describe a scenario in which there is an obstacle to be avoided at the beginning of the maneuver (left) or no obstacle along the path (right).

5.4 Experimental Results

In this section, we show how to analyze *quantitative* properties of the performance of car drivers for the two road models introduced in Section 5.3.3. We use the PRISM model checker [99], as our front-end tool to enter the model of the driver behavior, and the model-checking algorithm presented in Chapter 4 as our back-end engine to analyze the model properties, so that we can use likelihood and interval sets to express uncertainty in the estimation of state transition probabilities (PRISM does not support uncertainty sets).

We verify the PCTL properties reported in Table 5.1. For each property, we separately con-

Table 5.1: PCTL Properties Verified on the Driver Models

P1	P_{max}/P_{min} [Attention $\mathcal{U}$ $Unsafe$]
P2	P_{max}/P_{min} [(Attention $\wedge$ $\neg Swerving$) $\mathcal{U}$ $Final$]
P3	P_{max}/P_{min} [(Attention $\wedge$ $Right\ Lane$) $\mathcal{U}$ $Final$]
P4	P_{max}/P_{min} [(Attention $\wedge$ $\neg Braking$) $\mathcal{U}$ $Final$]
Attention is a placeholder for either *Attentive* or *Distracted*	

sider both attention levels, *attentive* and *distracted*, and compute both the maximum and minimum satisfaction probabilities, to give the range of predictions obtainable from the model. Property P1 computes the probability of reaching an *Unsafe* state while performing the maneuver. Properties P2 - P3 - P4 compute the probabilities of successfully reaching one of the final states *without*

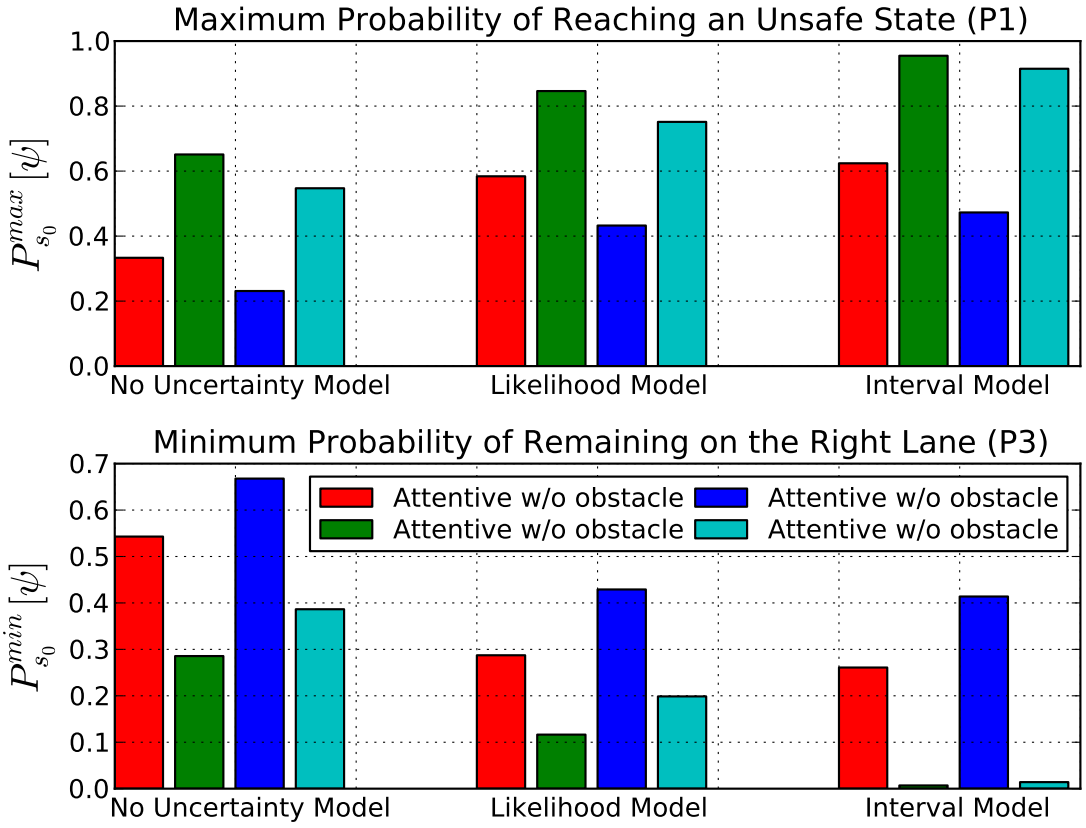


Figure 5.4: The figure shows a comparison among the model-checking results for properties P1 and P3 for the different models of uncertainty analyzed in this case study.

.

swerving, by always *staying on the right lane*, and *without braking*, respectively. Overall, these properties allow to capture different driving styles among subjects and to assess possible threats or misbehaviors of the drivers.

In Figure 5.4, we compare the model-checking results when using a model with no uncertainty, a model with interval uncertainties and a model with likelihood uncertainties. We assume 95% confidence level for both the interval and the likelihood models. For ease of graphical comparison, we only report results for one subject and only for the maximum satisfaction probability of P1 (top) and minimum satisfaction probability of P3 (*bottom*). The results for the other subjects and properties follow a similar trend. As expected, the probability of reaching an unsafe (*safe*) state is higher when the driver is distracted (*attentive*) and in the presence (*absence*) of an obstacle along the road. Further, we note that:

- probabilities computed for models with no uncertainty are significantly lower (*higher*), which implies that this method is potentially too optimistic for an appropriate threat assessment;

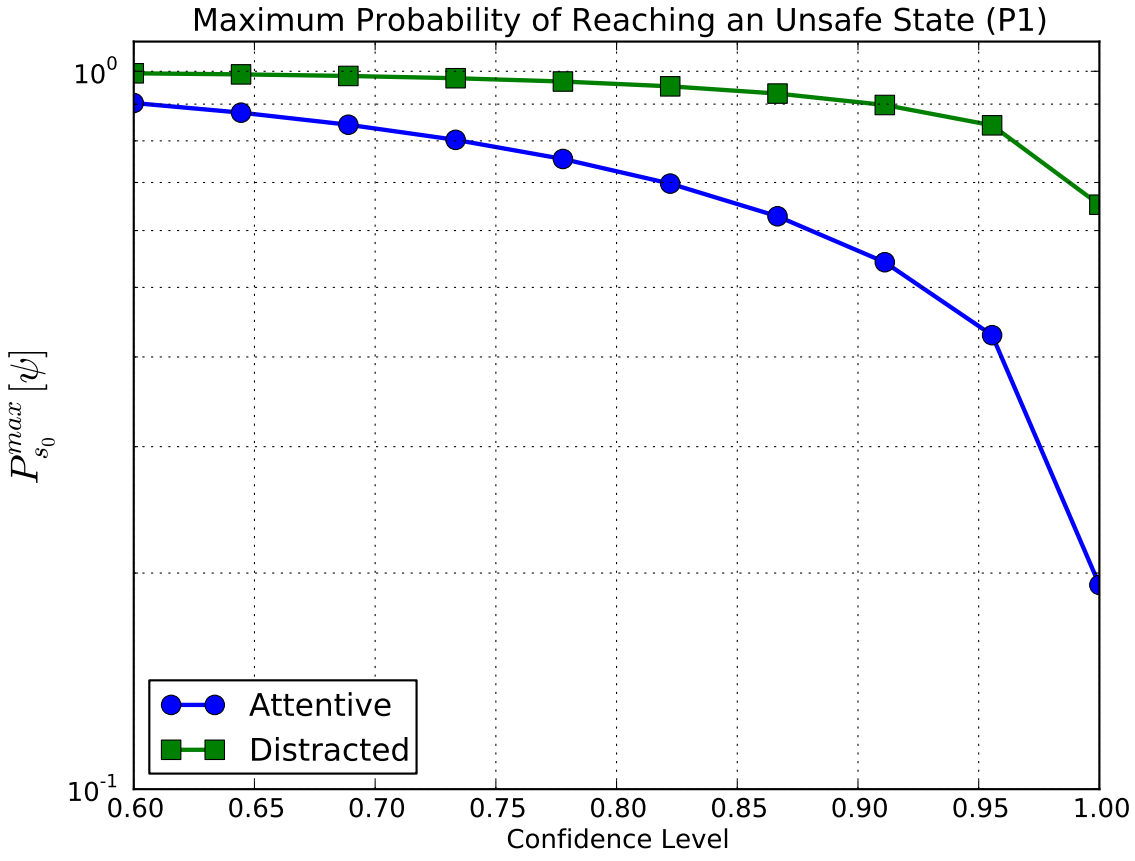


Figure 5.5: The figure shows a comparison of the values of maximum probability of reaching an unsafe state for distracted and attentive driving while sweeping the value of confidence level C_L.

- even a low value of C_L causes the computed probabilities to increase (*decrease*) substantially for the interval model, which might thus result in overly-conservative estimations, and;
- the likelihood model appears to be a good trade-off between the other two methods.

In the analysis presented in the rest of the section, we will use the likelihood model, which is often used when probabilities are estimated from experimental data because it is more statistically accurate than comparable methods.

Next, we examine the effects of different confidence levels on the results of the model-checking algorithm, and we compare the performance for a driver between when he is attentive and when he is distracted while driving. Figure 5.5 shows the model-checking results for property P1 and for values of confidence level C_L ranging from 60% to 99% for one driver. Results show that the probability of reaching an unsafe state is always lower in the case of attentive driving. The probability

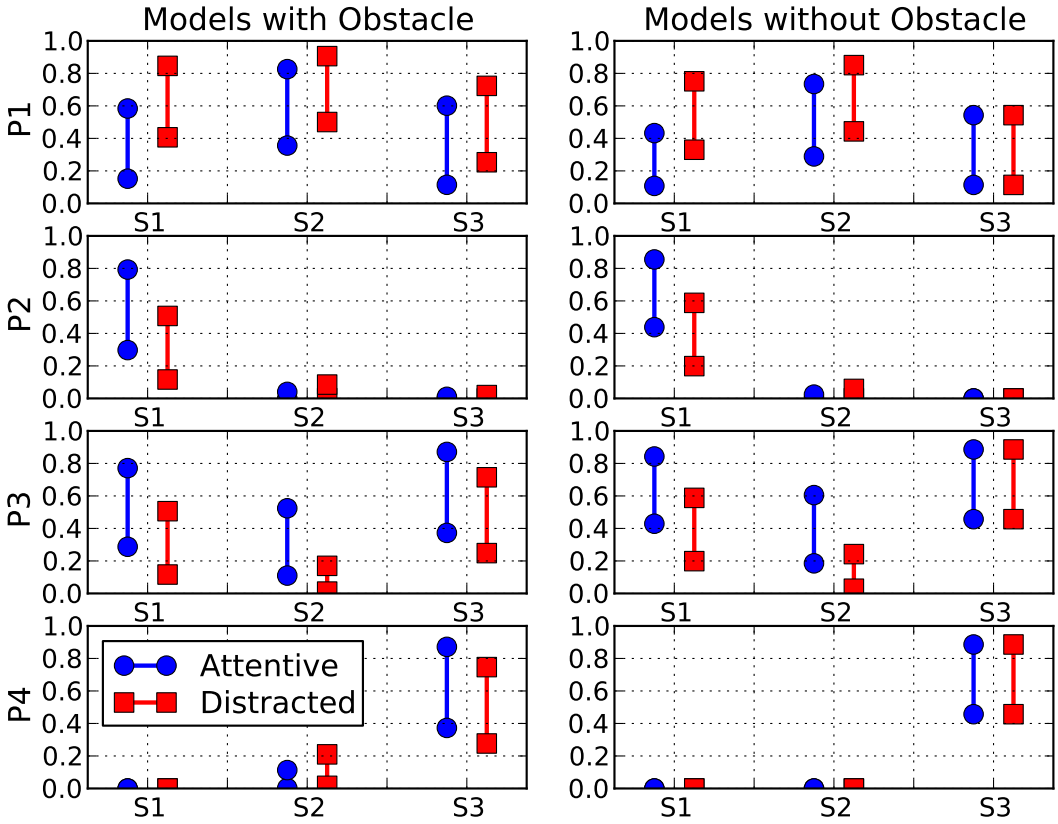


Figure 5.6: The figure reports the model-checking results of all the analyzed properties for three subjects. For each subject and property, the vertical bar marks the range between minimum and maximum satisfaction probability.

of reaching an unsafe state also decreases as we increase the confidence level, since the analysis becomes progressively less conservative. The model developer can use this plot to determine when the collected measurements are statistically relevant to estimate the driver performance, or, analogously, a developer of real-time control algorithms can use this information to assess how many data need to be collected from sensors and processed before being able to faithfully predict the future development of the system dynamics.

Moreover, we note that the trend shown in Figure 5.5 repeats for all the other subjects. However, the disparity between the results for attentive and distracted driving varies for each driver, and also the sensitivity to the level of confidence. Lower sensitivity to the confidence level is a sign of more repeatable behavior of the driver, since the few collected data points were already enough to predict his or her behavior accurately. This information can thus also be used to score the driving habits of each driver.

Finally, we report the collected model-checking results for three subjects (S) and for all four properties (P) in Figure 5.6. This analysis highlights the differences of driving styles among the subjects. For example, we note that results for property P3 show that subject S2 often ended up on the left lane while performing the maneuver, while subjects S1 and S3 managed to keep the right lane in most cases, indicating a higher precision in performing the maneuver, both in the presence of an obstacle and when the road was free. Further, results for property P4 show that subjects S1 and S2 tended to brake often when performing the maneuver, while S3 travelled along the road braking rarely. Since braking while performing a turn might cause threats due to loss of grip and sliding, subjects S1 and S2 might be advised to pay more attention and break *before* entering the double-turn and not *during* the maneuver. Analogously, a real-time control system might just detect the presence of a double turn (e.g., through the use of a GPS) and defensively break before starting the maneuver to increase the safety of the passengers. Finally, for all subjects, the presence of the obstacle increased the probability of reaching an unsafe state, as expected. Such a contingency might be detected by an automated system by using front-cameras and defensive maneuvers might be initiated to minimize the probability of threats.

Chapter 6

Optimal Control with Uncertainties

In this chapter, we address the problem of synthesizing control strategies for Convex-MDPs. The synthesized strategy optimizes a given system performance expressed in terms of total expected reward while guaranteeing that the system behavior complies to a specification written in PCTL under all resolutions of uncertainty. After formally defining the analyzed problem, we survey the classes of strategies available to control the execution of a Convex-MDP. We then focus our attention on Markov deterministic (MD) strategies. We first prove the NP-completeness of the problem of synthesizing an MD strategy from a PCTL specification with no operators with a finite time horizon such that the total expected reward of the Convex-MDP is higher than a given threshold. We then propose the first sound and complete algorithm to synthesize the MD strategy that maximizes the total expected reward of the Convex-MDPs, among those that satisfy the PCTL specification under all resolutions of uncertainty. Also PCTL specifications containing operators with a finite time horizon can be processed by the synthesis algorithm by first unrolling the finite execution of the Convex-MDP under analysis.

6.1 Problem Definition

The problem of controlling the behavior of a stochastic system naturally arises in many applications, ranging from robot path-planning and supply-chain management to the optimization of a financial portfolio. For example, in the context of robot path-planning, we might be interested in synthesizing a strategy such that the robot can *reach Destination A while avoiding all unsafe regions if no item of type 1 is available in such regions, and then reach Destination B through regions that are safe or at which items of type 2 are available with probability greater than* 50%*, while minimizing the total length of the path to be travelled.*

As it is evident from the previous example, the control problem aims to optimize some desired performance of the system (e.g., minimize the length of the path to be travelled), while also

constraining the behavior of the system to fulfill a given specification.

In this chapter, we address the problem of synthesizing control strategies for stochastic systems modeled using the Convex-MDP formalism. The synthesized strategies optimize the expected value of a given system performance expressed using the metric of the total expected reward, while at the same time constraining the execution of the Convex-MDP to fulfill a specification expressed using PCTL for any resolution of the uncertainty in the state-transition probabilities. We formally define the analyzed problem in the following.

Definition 6.1. ***Constrained Total Expected Reward Maximization for Convex-MDPs. Given*** *a Convex-MDP $\mathcal{M}_{\mathcal{C}}$, a reward structure r, and a PCTL specification ϕ,* ***synthesize*** *strategy σ^* for $\mathcal{M}_{\mathcal{C}}$ such that:*

$$\sigma^* = \underset{\sigma \in \Sigma_\phi}{argmax}\ \mathbb{W}^{\sigma}_{S_0} \tag{6.1}$$

where $\mathbb{W}^{\sigma}_{S_0}$ is the sum *of the expected rewards over all the initial states $s \in S_0$ of $\mathcal{M}_{\mathcal{C}}$, and Σ_ϕ is the set of strategies for which $\mathcal{M}_{\mathcal{C}}$ satisfies ϕ for any $\eta^a \in Nat$, starting from any state $s \in S_0$ and operating under $\sigma \in \Sigma_\phi$.*

Intuitively, we cast the strategy-synthesis problem into a constrained optimization problem. The feasible set of the problem is represented by all the strategies that guarantee that the execution of $\mathcal{M}_{\mathcal{C}}$ satisfies the PCTL specification ϕ for any resolution of uncertainty within the action range of the adversarial nature. Within the feasible set, we then aim to select the strategy that maximizes the total expected reward of $\mathcal{M}_{\mathcal{C}}$ over the reward structure r. From a game-theory perspective, we interpret strategies and natures as playing a game against one another, where strategies aim to maximize system performance while natures aim to minimize such performance to have $\mathcal{M}_{\mathcal{C}}$ fail specification ϕ.

Remark 6.1. *The techniques described in this chapter can also be applied to the dual problem of cost minimization, which arises when interpreting rewards as quantities to be minimized. This translates into replacing all "max" operators with "min" operators and vice versa in Problem (6.1) and in the mathematical derivations in the rest of the chapter.*

We review in the next session the different classes of strategies available to control a Convex-MDP.

6.2 Strategy Hierarchy for Convex-MDPs

We repeat for convenience the definition of the four classes of strategies for Convex-MDPs, as defined in Chapter 3.

1. **History-dependent (H).** A strategy is history-dependent if the choice of $a \in \mathcal{A}(s)$ for each state $s \in S$ is taken based on the sequence of previously visited states.

2. **Markov (M).** A strategy is Markov if the choice of a is taken based only on the last visited state.

3. **Randomized (R).** A strategy is randomized if the choice of a is taken stochastically among the available actions in $\mathcal{A}(s)$.

4. **Deterministic (D).** A strategy is deterministic if the choice of a is taken deterministically.

As it has been proved by Baier [15] and intuitively shown in the example in Figure 4.1, history-dependent strategies are in general strictly more powerful than Markov ones, i.e., there exist specifications written in PCTL that only history-dependent strategies can fulfill, while any Markov strategy would fail. Moreover, it is intuitive to understand that randomized strategies are strictly more powerful than deterministic ones, given that a deterministic strategy is in fact a special case of a randomized strategy, in which the probability distribution associated to the actions available in each state is a Dirac delta function. Formally, we can build the following hierarchy to classify strategies to control the execution of Convex-MDPs:

$$\text{MD} \prec \text{MR} \prec \text{HD} \prec \text{HR} \tag{6.2}$$

where we use $A \prec B$ to indicate that class of strategies A is strictly less powerful than class of strategies B. In particular, following the same reasoning presented in Section 4.1.2, we can argue that history-dependent strategies are more powerful than the Markov counterpart when the PCTL specification includes temporal operators with a finite time horizon (i.e., *Bounded Until*, *Instantaneous Reward* and *Bounded Cumulative Reward*), while Markov strategies are sufficient when these operators are disallowed and specifications are written only using the *Next*, *Unbounded Until* and *Cumulative Reward* operators.

In Section 6.4, we will show that the problem of strategy synthesis for Convex-MDPs is more complex than the verification one, and it requires higher computational effort. In fact, the problem is NP-complete even when considering the simplest class of MD strategies, and no efficient (polynomial-time) algorithm is known at the present time to solve it. The problem becomes even more complex for the synthesis of more powerful strategies. Since the focus of this dissertation is to develop scalable algorithms capable of analyzing real-world practical systems, in the following we will disallow the usage of temporal operators with a finite time horizon and only consider the synthesis of MD strategies.

This constraint on the allowable specifications represents a limitation in our approach. Nevertheless, we will show in Section 6.3 that it is possible to explicitly unroll the execution of a finite number of steps of the Convex-MDP within the model itself. An MD strategy synthesized on the unrolled model can then be interpreted as a HD strategy for the original model. As a consequence, we can argue that our approach is able to synthesize also HD strategies for PCTL properties containing a finite time horizon.

Moreover, also the exclusion of randomized strategies from the proposed synthesis algorithm might not be too stringent in practical applications. In fact, there exist classes of systems in which randomized strategies would not be acceptable by the user, because they would prevent the repeatability of the system executions. For example, if the Convex-MDP model is used to synthesize optimal dynamic voltage scaling strategies in multi-power domain Systems-on-Chip, only deterministic strategies would be acceptable to guarantee that the system operates always in the same

manner and to be able to trace back potential problems. In general, it can be proven that randomized strategies achieve better performance in the *expected* value sense [15]. On the other hand, they obviously have higher variance in the produced outcomes, thus possibly causing extra-challenges in the testing and maintenance of a system.

For completeness, we mention that there are instead other applications which would indeed benefit of randomized strategies, for example the problem of synthesizing strategies to maximize the profit of a financial portfolio. In general, randomized strategies are more powerful when they hinder the capability of an adversarial agent (e.g., another financial firm) to counteract effectively to the strategy of the system to be controlled. While deterministic strategies are easier to predict and challenge for the adversarial agent, a randomized strategy creates a less predictable behavior, thus possibly generating *on average* a better performance. The study of algorithms to generate random strategies for Convex-MDPs is thus left as an exciting further research direction for future work.

6.3 Execution Unrolling for Finite-Horizon Convex-MDPs

In this section, we show by means of an example how to synthesize history-dependent strategies for a Convex-MDP $\mathcal{M}_{\mathcal{C}}$ satisfying a PCTL specification ϕ containing operators with a finite time horizon (i.e., *Bounded Until*, *Instantaneous Reward* and *Bounded Cumulative Reward*). In particular, we reduce the problem of synthesizing a history-dependent strategy for $\mathcal{M}_{\mathcal{C}}$ to the problem of synthesizing a Markov strategy for a new Convex-MDP model $\mathcal{M}'_{\mathcal{C}}$ containing the unrolling of the execution of the original Convex-MDP $\mathcal{M}_{\mathcal{C}}$ for k_{max} steps, where k_{max} is the maximum time horizon in the operators contained in specification ϕ. Moreover, we show how to reformulate the PCTL specification into a new formula ϕ' containing only operators with no finite time horizon (i.e., *Next*, *Unbounded Until* and *Cumulative Reward*). The optimal HD strategy for $\mathcal{M}_{\mathcal{C}}$ can then be obtained by collecting the sequence of actions selected by the optimal MD strategy for $\mathcal{M}'_{\mathcal{C}}$ along the unrolled execution. This is, in fact, a classical construction [142], and we will use it also in the case study presented in Chapter 7.

We now present the example under analysis. We refer to the Convex-MDP $\mathcal{M}_{\mathcal{C}}$ shown in Figure 6.1, which was already introduced in Section 4.1.2 and which is reported here for convenience.

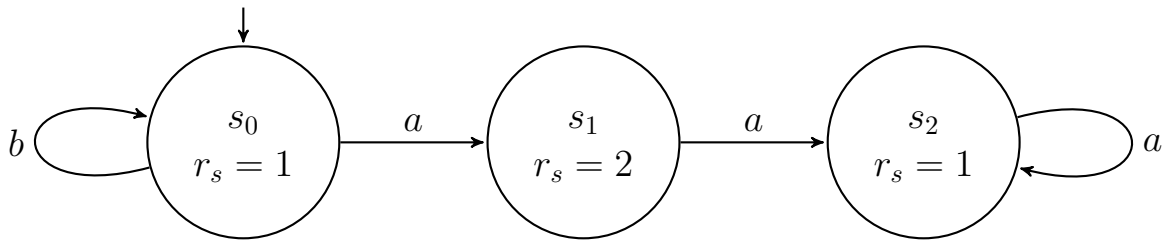


Figure 6.1: Example of a Convex-MDP for which only history-dependent strategies are optimal.

We aim to synthesize the optimal strategy σ to satisfy specification:

$$\phi = \boldsymbol{R}^{r}_{\geq 2}[\boldsymbol{\mathcal{I}^{=3}}]$$

It is easy to see that the optimal strategy σ is history-dependent. It takes the self-loop b twice and then selects action a, which yields expected instantaneous reward of 2, so that $\mathcal{M}_{\mathcal{C}}, \sigma \models \phi$.

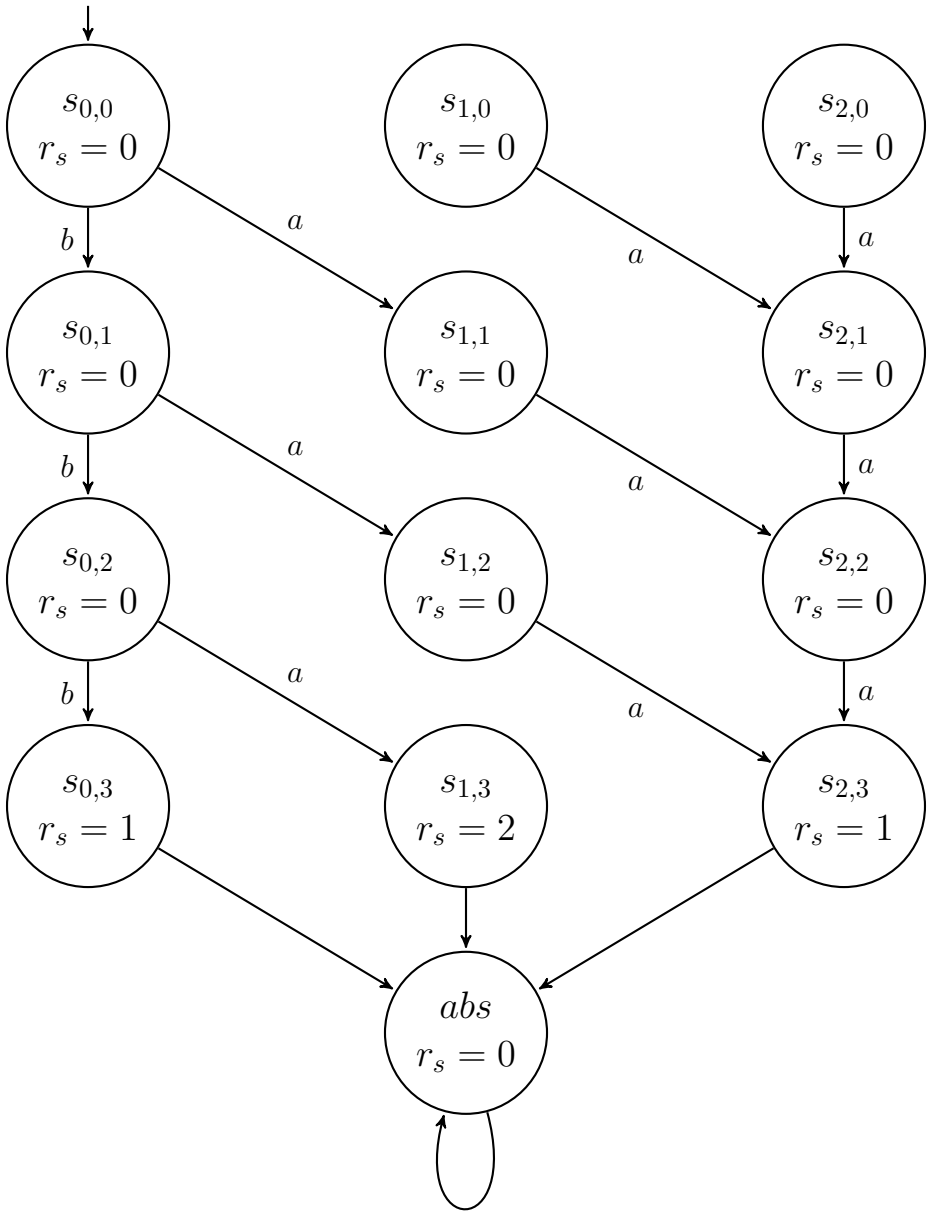


Figure 6.2: Example of the transformation of a Convex-MDP model required to encode 3 steps of the execution history.

Moreover, there is no Markov strategy for $\mathcal{M}_{\mathcal{C}}$ that could achieve such a reward, so the model would not be able to satisfy ϕ if strategies were limited to be Markov.

We now apply the reduction introduced at the beginning of the section. In particular, we refer to the Convex-MDP $\mathcal{M}'_{\mathcal{C}}$ shown in Figure 6.2, which was obtained by unrolling the execution of $\mathcal{M}'_{\mathcal{C}}$ for $k = 3$ steps and by adding a final deterministic transition to an absorbing state, labeled as abs. In $\mathcal{M}'_{\mathcal{C}}$, we add a further index to each state to record the step of the execution history that the state belongs to. For example, state $s_{1,2} \in S'$ represents state $s_1 \in S$ at the second step of the execution. Moreover, we annotate with rewards only the states in $\mathcal{M}'_{\mathcal{C}}$ associated to the last execution step, to avoid counting the same reward multiple times.

We now reformulate specification ϕ as ϕ' using the *Cumulative Reward* operator:

$$\phi' = \boldsymbol{R}^{r}_{\geq 2}[\boldsymbol{C}\ abs]$$

The optimal Markov strategy σ' for $\mathcal{M}'_{\mathcal{C}}$ selects action b in states $s_{0,0}$ and $s_{0,1}$ and action a in state $s_{0,2}$. By collecting this sequence of optimal actions selected by σ', we obtain the same history-dependent strategy σ synthesized for $\mathcal{M}_{\mathcal{C}}$ and the same expected reward of 2. Also for $\mathcal{M}'_{\mathcal{C}}$, we can thus write $\mathcal{M}'_{\mathcal{C}}, \sigma' \models \phi'$.

The reduction introduced in this section reformulates the problem of synthesizing history-dependent strategies to the problem of synthesizing Markov strategies. Such a reduction allows us to argue that the algorithm presented in Section 6.5 can handle properties expressed with the full PCTL syntax, i.e., also with operators with a finite time horizon.

However, we notice that the reduction comes at the cost of an increase of the number of states and transitions in the Convex-MDP model which is linear in the time horizon k. Such a penalty is particularly severe in the context of strategy synthesis because such a problem is NP-complete (as it will be proved in Section 6.4), and the best-known algorithms to solve it have worst-case runtime execution exponential in the size of the Convex-MDP model. As a consequence, great attention must be paid in setting the number of execution steps, i.e., the time horizon k, that need to be unrolled, in order to limit the exponential increase in runtime to the strict necessary.

6.4 Theoretical Complexity of the Synthesis Problem for MD Strategies

In this section, we present the main theoretical results of this chapter about the theoretical complexity of the problem of synthesizing optimal MD control strategies for Convex-MDPs. We begin by redefining the synthesis problem introduced in Definition 6.1 with the additional limitation of considering only MD strategies.

Definition 6.2. ***Constrained Total Expected Reward Maximization for Convex-MDPs using MD Strategies. Given*** *a Convex-MDP $\mathcal{M}_{\mathcal{C}}$, a reward structure r, and a PCTL specification ϕ containing no operator with a finite time horizon,* ***determine*** *the MD strategy σ^* for $\mathcal{M}_{\mathcal{C}}$ such that:*

$$\sigma^* = \underset{\sigma \in \Sigma_\phi^{MD}}{argmax}\ \mathbb{W}_{S_0}^{\sigma} \tag{6.3}$$

where Σ_ϕ^{MD} *is the set of Markov-Deterministic strategies for which* $\mathcal{M}_\mathcal{C}$ *satisfies* ϕ *for any* $\eta^a \in Nat$*, starting from any* $s \in S_0$ *and operating under* $\sigma \in \Sigma_\phi^{MD}$*.*

Before formalizing the results on theoretical complexity, we give an intuitive explanation of why the complexity of the synthesis problem for MD strategies differs from the polynomial-time result proven for the model-checking problem in Theorem 4.1. For simplicity, in this explanation we set the reward for each state and action in $\mathcal{M}_\mathcal{C}$ to be zero, so that all strategies result in the same (null) total expected reward and we can pick any strategy that satisfies specification ϕ. To make the explanation more concrete, we will aim to synthesize an MD strategy for the Interval-MDP $\mathcal{M}_\mathcal{C}$ shown in Figure 6.3 such that $\mathcal{M}_\mathcal{C}$ satisfies the following PCTL specification:

$$\phi = \boldsymbol{P}_{\geq 0.4}[\boldsymbol{\mathcal{X}}\ \omega] \wedge \boldsymbol{P}_{\geq 0.4}[\boldsymbol{\mathcal{X}}\ \vartheta]$$

We consider the satisfaction probabilities for state s_1, which is the initial state of $\mathcal{M}_\mathcal{C}$. We consider one *Next* operator at a time, as it was done in the context of model checking. We notice that action a maximizes the satisfaction probability for the first *Next* operator, i.e., it would result in $P_{s_1}^{a,min}[\boldsymbol{\mathcal{X}}\ \omega] = 0.7$. On the other hand, it would cause $\mathcal{M}_\mathcal{C}$ to fail the requirement expressed using the second *Next* operator, since $P_{s_1}^{a,min}[\boldsymbol{\mathcal{X}}\ \vartheta] = 0.2$. Action b is suboptimal in satisfying the first *Next* operator, since it results in $P_{s_1}^{b,min}[\boldsymbol{\mathcal{X}}\ \omega] = 0.45$. Nevertheless this probability value is still higher than the threshold $p = 0.4$, so b would satisfy the first *Next* operator. Moreover, action b satisfies also the second *Next* operator, since it results in $P_{s_1}^{b,min}[\boldsymbol{\mathcal{X}}\ \vartheta] = 0.45$. As a result, the optimal MD strategy is:

$$\sigma^* : \{s_0, s_2\} \rightarrow a, \{s_1\} \rightarrow b$$

In the context of verification, we need to verify that *all* strategies satisfy the PCTL specification (universal quantification). The model-checking algorithm can consider one PCTL operator at a time, determine the *worst-case* adversary for that operator and verify that the property still holds for such an adversary. When run on the previously described example, the algorithm would have

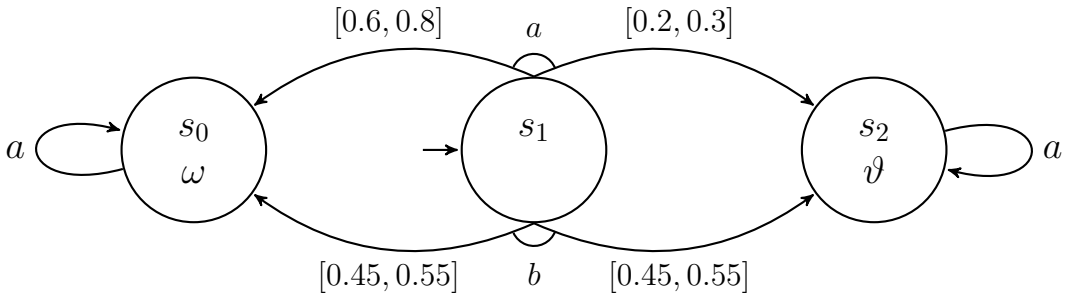


Figure 6.3: Simple Interval-MDP used to illustrate the differences between the model-checking and the strategy-synthesis problems.

reported that $\mathcal{M}_{\mathcal{C}}$ does not satisfy ϕ. In the context of control of a Convex-MDP, instead, the strategy-synthesis algorithm needs to determine whether it exists a strategy that does satisfy the PCTL specification (existential quantification). As a consequence, the synthesis algorithm cannot simply compute the *best-case strategy* for each operator, because a strategy that is sub-optimal for each operator could in fact be the only one that satisfies all of them. For each operator in ϕ, and for each state $s \in S$, the algorithm needs instead to store all the actions that satisfy the operator, and then find the combination of actions that satisfy all operators, if such a combination exists. It should be apparent from this example that the combinatorial nature of the strategy-synthesis problem will require more computational effort than the model-checking problem in order to be solved.

We now formalize the results on theoretical complexity. In particular, as it is commonly done when proving results of this kind, we will consider the decision problem version of Problem 6.2, i.e., we do not ask for the MD strategy that maximizes the total reward of $\mathcal{M}_{\mathcal{C}}$, but simply for an MD strategy that results in a total expected reward higher than a given threshold $\mathbb{W}_T$ and which satisfies the PCTL specification ϕ under all resolutions of uncertainty.

In preparation of the main result, we introduce the following lemmas.

Lemma 6.1. ***Complexity of PCTL Model Checking for Convex-MDPs.*** *Given a Convex-MDP $\mathcal{M}_{\mathcal{C}}$, the problem of model checking a PCTL formula ϕ containing no operator with a finite time horizon is decidable in P. Further, there exist sound and complete algorithms to solve the model-checking problem.*

Sketch of proof. The result on theoretical complexity immediately follows from the first part of Theorem 4.1. Further, a sound and complete algorithm for the PCTL model checking of Convex-MDPs was presented in Section 4.2. □

Lemma 6.2. ***Computation of the Total Expected Reward for Convex-MCs.*** *It is possible to compute the total expected reward $\mathbb{W}_{S_0}$ of a Convex-MC in polynomial time.*

Sketch of proof. We refer the reader to the results presented by Nilim and El Ghaoui [125], which introduce polynomial-time algorithms to compute the total expected reward of Convex-MDPs and Convex-MCs. Their approach relies on the classical Bellman recursion on the state transitions of the Convex-MDP, and on clever formulations of the convex programs to be solved at each step of the recursion to resolve the uncertainty in the transitions. □

Lemma 6.3. ***Complexity of the Synthesis of MD Strategies for MDPs from PCTL Specifications.*** *The problem of determining the existence of an MD strategy σ for an MDP $\mathcal{M}$ such that $\mathcal{M}, \sigma \models \phi$ is NP-complete.*

Sketch of proof. We omit the technicalities of the proof for brevity. The interested reader is referred to the work by Baier [15]. Intuitively, the same reasoning illustrated on the example of Figure 6.3 can be applied also when analyzing MDPs. In particular, the proof by Baier [15] does not consider the presence of a total expected reward for the MDP to be maximized and just focuses on the problem of synthesizing an MD strategy σ that satisfies the PCTL specification ϕ, as we did in the example above. □

We are now ready to prove the main theoretical result of this chapter about the complexity of the strategy-synthesis problem introduced in Definition 6.2.

Theorem 6.1. ***Complexity of MD Strategy Synthesis for Convex-MDPs from a PCTL Specification.*** *The problem of determining the existence of an MD strategy σ for a Convex-MDP $\mathcal{M}_{\mathcal{C}}$, with total expected reward $\mathbb{W}^{\sigma}_{S_0}$ larger or equal to $\mathbb{W}_T$ and satisfying a PCTL specification ϕ containing no operator with a finite time horizon under all resolutions of uncertainty is NP-complete.*

Proof. As in all proofs of NP-completeness, we need to prove the following facts:

- that the analyzed problem is in NP, i.e., given a candidate solution it is possible to verify in polynomial-time whether the solution does satisfy the problem or not;
- that the analyzed problem in NP-hard, i.e., that another known NP-hard problem can be reduced in polynomial-time to it.

For the first part of the proof, we notice that, given a candidate solution σ_c, we can in polynomial time:

1. check whether $\mathcal{M}_{\mathcal{C}}, \sigma_c \models_{Nat} \phi$ by Lemma 6.1, and;
2. compute $\mathbb{W}^{\sigma_c}_{S_0}$ on the induced Convex-MC $\mathcal{M}^{\sigma_c}_{\mathcal{C}}$ by Lemma 6.2.

Strategy σ_c is a solution of the problem if and only if check 1) passes and if the computed total expected reward is larger than the given threshold, i.e., $\mathbb{W}^{\sigma_c}_{S_0} \geq \mathbb{W}_T$. This concludes the proof that the problem is in NP.

To prove NP-hardness, we reduce the problem introduced in Lemma 6.3 to the one under analysis. We set $\mathbb{W}_T = 0$, and describe state-transition probabilities with point intervals (in the case of Interval-MDPs), i.e., intervals in which the lower and upper bound coincide. By doing this, the solution returned by an algorithm solving the analyzed problem can directly be interpreted as a solution for the problem in Lemma 6.3. □

The result of NP-hardness in the proof of Theorem 6.1 should have been expected given the result reported in Lemma 6.3 on a simpler problem. The main contribution of Theorem 6.1 is instead in showing that the synthesis problem of MD strategies for Convex-MDPs remains in the same complexity class of the analogous problem applied to MDPs, i.e., considering uncertainties in the estimation of the state-transition probabilities does not increase the theoretical complexity of the problem.

6.5 A Synthesis Algorithm for MD Strategies

In this section, we describe the first sound and complete algorithm to solve the strategy-synthesis problem for Convex-MDPs introduced in Definition 6.2. While previously proposed approaches

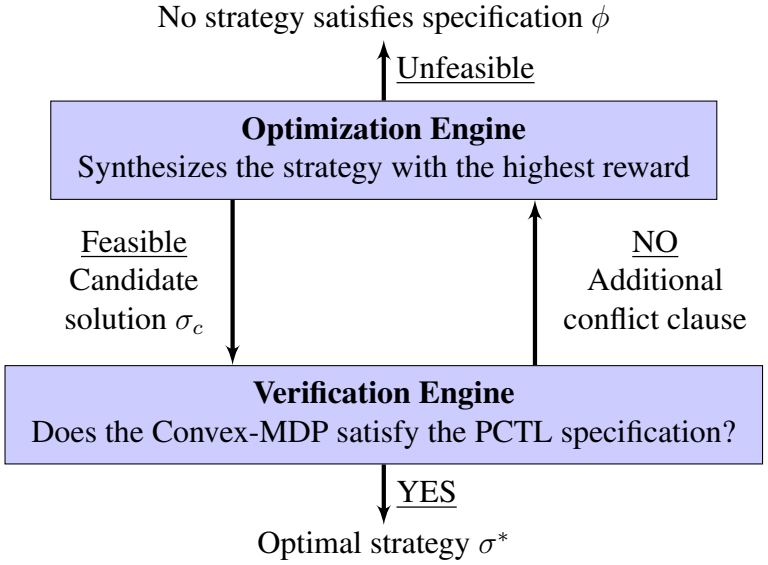


Figure 6.4: The figure shows a block diagram of the proposed algorithm for the synthesis of MD strategies.

were not complete or were valid only for a sub-class of PCTL properties, as explained in Section 3.3, our approach can handle arbitrary PCTL specifications[1] and it is sound and complete, as it will be proven at the end of this section.

We begin with an overview of the chosen approach which will help understanding the rest of the results proposed in the section.

We use a *lazy* approach based on strategy-iteration, conceptually similar to works previously proposed in the literature [71, 126]. As shown in Figure 6.4, the algorithm is split into two main routines communicating in a loop. At each iteration, the *optimization* engine (OE) is responsible to generate a candidate strategy σ_{c_1}. Strategy σ_{c_1} is guaranteed to maximize the total expected reward $\mathbb{W}^{\sigma}_{S_0}$ of the Convex-MDP under analysis, but it does not necessarily satisfy the PCTL specification ϕ, since the OE formulation does not contain any information about ϕ. The candidate solution σ_{c_1} is then passed to the *verification* engine (VE) which checks whether the Convex-MDP satisfies ϕ for all resolutions of uncertainty under the action range of the adversarial nature, i.e., $\forall \eta^a \in Nat$, when operating under σ_{c_1}. If the check passes, $\sigma^* = \sigma_{c_1}$ and the algorithm terminates. Otherwise, the VE generates an additional constraint for the OE to prevent σ_{c1} to be selected again. We can interpret this constraint as a *conflict clause*, i.e., a formal explanation (clause) that the problem is unfeasible (conflict). The OE reads-in this additional constraint and generates a new candidate strategy σ_{c_2}, which is the strategy that maximizes the total expected reward $\mathbb{W}^{\sigma}_{S_0}$ among the remaining ones, i.e., the strategies in the set $\Sigma^{MD} \setminus \{\sigma_{c_1}\}$, and the loop repeats. The algorithm terminates when either a strategy σ_{c_j} is found at the j^{th} iteration which satisfies specification ϕ (termination with success)

[1] If the PCTL specification contains an operator with a finite time horizon k, the execution of the Convex-MDP needs to be unrolled for k steps, as explained in Section 6.3, before the proposed algorithm can be applied.

or when all strategies $\sigma \in \Sigma^{MD}$ have been explored and none satisfies specification ϕ (termination with failure).

The novelty of our approach is devising the mathematical formulation for the OE capable of generating at each iteration the candidate strategy that maximizes the total expected reward among those still available. The first candidate strategy that also satisfies the PCTL specification ϕ, as verified by the VE, becomes the solution of the synthesis problem. Such a strategy is feasible (satisfies ϕ) and it is guaranteed to result in the maximum total reward among the strategies that satisfy ϕ, so it solves the strategy-synthesis problem introduced in Definition 6.2 exactly.

Remark 6.2. *In general, there is not guarantee on the uniqueness of the optimal strategy σ^*, i.e., multiple strategies with the same expected reward might exist, each satisfying ϕ. Since every such strategy is equivalent from the user perspective, the algorithm just reports the first one found. Alternatively, all strategies with the same total expected reward could be generated by continuing the iteration between the OE and the VE until a reduction in the expected reward was detected, and by reporting all synthesized strategies satisfying ϕ.*

The next subsections give details on the mathematical formulations of the optimization and verification engines and analyze the algorithm properties.

6.5.1 Optimization Engine

In this section, we give details about the mathematical formulation for the optimization engine (OE).

We start with the classical linear-programming (LP) formulation to maximize the total expected reward for MDPs [142].

$$\begin{aligned} &\min_{x,l} \sum_{s\in S_0} x_s \\ &\text{s.t. } x_s - l_s^a = r_s^a + \mathbf{x^T f_s^a}; && \forall s \in S, \forall a \in \mathcal{A}(s) \\ &\quad x_s, l_s^a \geq 0 && \forall s \in S, \forall a \in \mathcal{A}(s) \end{aligned} \tag{6.4}$$

Vector $\mathbf{x}$ collects the total expected reward for each state $s \in S$ (at the end of the optimization $\mathbb{W}_s^{\sigma_c} = x_s, \forall s \in S$), and the cost function sums the total expected rewards for all the initial states $s \in S_0$. We then set $\mathbb{W}_{S_0}^{\sigma_c} = \sum_{s\in S_0} x_s$. Variables l_s^a are slack variables for each constraint. Further, we write $r_s^a = r_s(s) + r_a(s,a), \forall s \in S, \forall a \in A$ to simplify the notation. Since the slack variables have negative sign, the slack can only be negative, i.e., the left-hand side (LHS) can only be larger or equal than the right-hand side (RHS). The "min" operator makes sure that, for each state, the constraint with the highest RHS has null slack, i.e., $l_s^a = 0$. The optimal MD strategy σ_c can then be reconstructed by selecting the action $a \in \mathcal{A}(s), \forall s \in S$ corresponding to the constraint with null slack, e.g., $\sigma_c(s_0, a) = 1$ if $l_{s_0}^a = 0$.

Our goal is to modify such a formulation to allow a sub-optimal solution to be selected, because we will use this feature to discard strategies σ_c that have been proven by the VE not to satisfy the PCTL specification ϕ. To achieve this goal, we now describe an *equivalent* formulation of

Problem (6.4). This formulation produces the same result of Problem (6.4), but it is more suitable to allow the selection of sub-optimal solutions. We will describe in Section 6.5.2 how to *add constraints* to this new formulation to actually select sub-optimal solutions in decreasing order of total expected reward $\mathbb{W}^{\sigma}_{S_0}$.

We refer to Problem (6.5):

$$\begin{aligned}
&\max_{x,z,l,n} \sum_{s\in S_0} x_s \\
&\text{s.t. } x_s - l^a_s + n^a_s = r^a_s + \mathbf{x^T f^a_s}; && \forall s \in S, \forall a \in \mathcal{A}(s) && \text{(6.5a)}\\
&\quad l^a_s \le B z^a_s, \quad n^a_s \le B z^a_s; && \forall s \in S, \forall a \in \mathcal{A}(s) && \text{(6.5b)}\\
&\quad \mathbf{z^T_s 1} = M_s - 1; && \forall s \in S && \text{(6.5c)}\\
&\quad x_s, l^a_s, n^a_s \ge 0, \quad z^a_s \in \{0,1\} && \forall s \in S, \forall a \in \mathcal{A}(s)
\end{aligned}$$

We associate a binary variable z^a_s to each action for every state, so the problem becomes a Mixed-Integer Linear Program (MILP). At the end of the optimization, $z^{a_j}_{s_i} = 0$ if action a_j is chosen for state s_i, and Constraint (6.5c) guarantees that only one action can be selected for each state ($M_s = |\mathcal{A}(s)|$). For example, $\sigma_c(s_0, a) = 1$, if $z^a_{s_0} = 0$.

We then associate to each constraint a second slack variable n^a_s, with sign opposite to l^a_s. If action a_j is selected at state s_i, i.e., $z^{a_j}_{s_i} = 0$, Constraint (6.5b) makes sure that $z^{a_j}_{s_i} = 0$ implies $l^{a_j}_{s_i} = 0 \wedge n^{a_j}_{s_i} = 0$, so that the corresponding Constraint (6.5a) can set the correct value of x_s. We use constant B to represent a *big* number with respect to the problem data. Constant B needs to be higher than the reward computed for any state, i.e., $B \ge x_s, \forall s \in S$, but not excessively high to avoid convergence problems in the optimization algorithm when solving Problem (6.5).

If action a_k is *not* selected at state s_i, i.e., $z^{a_k}_{s_i} = 1$, variable $l^{a_k}_{s_i} > 0$ ($n^{a_k}_{s_i} > 0$) implies that selecting action a_k would have resulted in a lower (higher) value of x_{s_i}. With these constraints, *any* action can be selected. We, finally, change the optimization operator to "max", so that, at the first iteration of the algorithm, the total expected reward gets maximized.

We now proceed to consider uncertainties in the transition probabilities. Constraint (6.5a) gets updated to Constraint (6.6), since the adversarial nature tries to minimize the expected reward. The decision variable of the inner problem is $\mathbf{f}^a_s$ and its optimal value $\nu(\mathbf{x})$ is parametrized in the outer problem decision variable $\mathbf{x}$. The new constraint can be made linear again for an arbitrary uncertainty model by replacing it with a set of constraints, one for each point in $\mathcal{F}^a_s$. However, this approach results in infinite constraints if the set $\mathcal{F}^a_s$ contains infinitely many points, as for the models of uncertainty considered in this dissertation, thus making the problem not solvable. We solve this difficulty using duality, which allows to rewrite Constraint (6.6) with a number of additional constraints only polynomial in the size $\mathcal{R}$ of the Convex-MDP. In Constraint (6.7), we replace the primal inner problem with its dual, $\forall s \in S, a \in \mathcal{A}(s)$:

$$x_s - l^a_s + n^a_s = r^a_s + \min_{\mathbf{f^a_s} \in \mathcal{F}^a_s} \mathbf{x^T f^a_s} \tag{6.6}$$

$$\Downarrow$$

$$x_s - l^a_s + n^a_s = r^a_s + \max_{\boldsymbol{\lambda}^{\boldsymbol{a}}_{\boldsymbol{s}} \in \mathcal{D}^a_s} g\left(\boldsymbol{\lambda}^{\boldsymbol{a}}_{\boldsymbol{s}}, \mathbf{x}\right) \tag{6.7}$$

where $\boldsymbol{\lambda}_{\boldsymbol{s}}^{\boldsymbol{a}}$ is the (vector) Lagrange multiplier, $\mathcal{D}_s^a$ is the feasibility set of the dual problem and $g\left(\boldsymbol{\lambda}_{\boldsymbol{s}}^{\boldsymbol{a}}, \mathbf{x}\right)$ is the dual cost function. We notice that the optimal value of the dual problem $d(\mathbf{x})$ is again parametrized in the outer problem decision variable $\mathbf{x}$.

The dual problem is convex by construction in the dual variable $\boldsymbol{\lambda}_{\boldsymbol{s}}^{\boldsymbol{a}}$ [30] and it has size polynomial in $\mathcal{R}$, according to results from convex theory. Since also the primal problem is convex, strong duality holds, i.e., the primal and dual optimal values coincide $\nu(\mathbf{x}) = d(\mathbf{x})$, because the primal problem satisfies Slater's condition [30] for any non-trivial uncertainty set $\mathcal{F}_s^a$. Any dual solution underestimates the primal optimal solution. When substituting the primal problem with the dual in Constraint (6.7), we can drop the inner optimization operator because the outer optimization operator will nevertheless aim to find the least underestimate to maximize its cost function. We thus get the formulation:

$$
\begin{aligned}
&\max_{x,\lambda,l,n,z} \sum_{s\in S_0} x_s \\
&\text{s.t. } x_s - l_s^a + n_s^a = r_s^a + g\left(\boldsymbol{\lambda}_{\boldsymbol{s}}^{\boldsymbol{a}}, \mathbf{x}\right); && \forall s \in S, \forall a \in \mathcal{A}(s) \\
&\quad l_s^a \le B z_s^a, \quad n_s^a \le B z_s^a; && \forall s \in S, \forall a \in \mathcal{A}(s) \\
&\quad \mathbf{z_s^T 1} = M_s - 1; && \forall s \in S \\
&\quad x_s, l_s^a, n_s^a \ge 0, \boldsymbol{\lambda}_{\boldsymbol{s}}^{\boldsymbol{a}} \in \mathcal{D}_s^a, z_s^a \in \{0,1\} && \forall s \in S, \forall a \in \mathcal{A}(s)
\end{aligned}
\tag{6.8}
$$

Finally, we get the full formulation for the OE in Problem (6.9), where we replace each dual cost function $g\left(\boldsymbol{\lambda}_{\boldsymbol{s}}^{\boldsymbol{a}}, \mathbf{x}\right)$ with the epigraph variable t_s^a and add an additional constraint on t_s^a to bound its value to the value of g. This transformation is quite common in the formulation of convex problems, since only linear functions are allowed in equality constraints. We notice that the transformation does not, in fact, change the optimization result, since the outer optimization operator will enforce $t_s^a = g\left(\boldsymbol{\lambda}_{\boldsymbol{s}}^{\boldsymbol{a}}, \mathbf{x}\right)$ at optimum.

$$
\begin{aligned}
&\max_{x,\lambda,l,n,z,t} \sum_{s\in S_0} x_s \\
&\text{s.t. } x_s - l_s^a + n_s^a = r_s^a + t_s^a; && \forall s \in S, \forall a \in \mathcal{A}(s) \\
&\quad t_s^a \le g\left(\boldsymbol{\lambda}_{\boldsymbol{s}}^{\boldsymbol{a}}, \mathbf{x}\right); && \forall s \in S, \forall a \in \mathcal{A}(s) \\
&\quad l_s^a \le B z_s^a, \quad n_s^a \le B z_s^a; && \forall s \in S, \forall a \in \mathcal{A}(s) \\
&\quad \mathbf{z_s^T 1} = M_s - 1; && \forall s \in S \\
&\quad x_s, l_s^a, n_s^a \ge 0, \boldsymbol{\lambda}_{\boldsymbol{s}}^{\boldsymbol{a}} \in \mathcal{D}_s^a, z_s^a \in \{0,1\} && \forall s \in S, \forall a \in \mathcal{A}(s)
\end{aligned}
\tag{6.9}
$$

Similarly to the CP formulation in Problem (4.18), the decision variables of Problem (6.9) include both $\mathbf{x}$ and $\boldsymbol{\lambda}_s^a$, so Problem (6.9) is convex only if the dual function $g(\boldsymbol{\lambda}_s^a, \mathbf{x})$ is jointly convex in $\boldsymbol{\lambda}_s^a$ and $\mathbf{x}$. While this condition cannot be guaranteed for arbitrary uncertainty models, it is possible to show constructively that it holds for the ones considered in the dissertation, following the same reasoning presented in Section 4.2.3.1.

For example, for the interval model of uncertainty, Problem (6.9) can be written as Problem (6.10), which is a Mixed-Integer Linear Program (MILP), so trivially jointly convex in $\mathbf{x}$

and $\boldsymbol{\lambda}_s^a$.

$$
\begin{aligned}
&\max_{x,\lambda,z,l,n} \sum_{s\in S_0} x_s \\
&\text{s.t. } x_s - l_s^a + n_s^a = r_s^a + t_s^a; && \forall s \in S, \forall a \in \mathcal{A}(s) \\
&\quad t_s^a \le \lambda_{1,s}^a + (\underline{\mathbf{f}}_a^s)^T \boldsymbol{\lambda}_{\mathbf{2},\boldsymbol{s}}^{\boldsymbol{a}} - (\overline{\mathbf{f}}_a^s)^T \boldsymbol{\lambda}_{\mathbf{3},\boldsymbol{s}}^{\boldsymbol{a}}; && \forall s \in S, \forall a \in \mathcal{A}(s) \\
&\quad l_s^a \le B z_s^a, \quad n_s^a \le B z_s^a; && \forall s \in S, \forall a \in \mathcal{A}(s) \\
&\quad \mathbf{z_s^T 1} = M_s - 1; && \forall s \in S \\
&\quad x_s, l_s^a, n_s^a, \boldsymbol{\lambda}_{\mathbf{2},\boldsymbol{s}}^{\boldsymbol{a}} \ge 0,\ \boldsymbol{\lambda}_{\mathbf{3},\boldsymbol{s}}^{\boldsymbol{a}} \ge \mathbf{0},\ z_s^a \in \{0,1\}; && \forall s \in S, \forall a \in \mathcal{A}(s) \\
&\quad \mathbf{x} - \boldsymbol{\lambda}_{\mathbf{2},\boldsymbol{s}}^{\boldsymbol{a}} + \boldsymbol{\lambda}_{\mathbf{3},\boldsymbol{s}}^{\boldsymbol{a}} - \lambda_{1,s}^a \mathbf{1} = \mathbf{0} && \forall s \in S, \forall a \in \mathcal{A}(s)
\end{aligned}
\tag{6.10}
$$

with $\boldsymbol{\lambda}_{\boldsymbol{s}}^{\boldsymbol{a}} = [\lambda_{1,s}^a, \boldsymbol{\lambda}_{\mathbf{2},\boldsymbol{s}}^{\boldsymbol{a}}, \boldsymbol{\lambda}_{\mathbf{3},\boldsymbol{s}}^{\boldsymbol{a}}]$.

For the ellipsoidal model, Problem (6.9) can be written as Problem (6.11), which is a Mixed-Integer Quadratic-Constrained Program (MIQCP), so again trivially jointly convex in $\mathbf{x}$ and $\boldsymbol{\lambda}_s^a$.

$$
\begin{aligned}
&\max_{x,\lambda,z,l,n} \sum_{s\in S_0} x_s \\
&\text{s.t. } x_s - l_s^a + n_s^a = r_s^a + t_s^a; && \forall s \in S, \forall a \in \mathcal{A}(s) \\
&\quad t_s^a \le \lambda_{1,s}^a - \lambda_{2,s}^a - (\mathbf{h_s^a})^T E_s^a \boldsymbol{\lambda}_{\mathbf{3},\boldsymbol{s}}^{\boldsymbol{a}}; && \forall s \in S, \forall a \in \mathcal{A}(s) \\
&\quad l_s^a \le B z_s^a, \quad n_s^a \le B z_s^a; && \forall s \in S, \forall a \in \mathcal{A}(s) \\
&\quad \mathbf{z_s^T 1} = M_s - 1; && \forall s \in S \\
&\quad x_s, l_s^a, n_s^a, \lambda_{2,s}^a \ge 0,\ \boldsymbol{\lambda}_{\mathbf{3},\boldsymbol{s}}^{\boldsymbol{a}} \ge \mathbf{0},\ z_s^a \in \{0,1\}; && \forall s \in S, \forall a \in \mathcal{A}(s) \\
&\quad \|\boldsymbol{\lambda}_{\mathbf{3},\boldsymbol{s}}^{\boldsymbol{a}}\|_2 \le \lambda_{2,s}^a,\ \mathbf{x} - \lambda_{1,s}^a \mathbf{1} + (E_s^a)^T \boldsymbol{\lambda}_{\mathbf{3},\boldsymbol{s}}^{\boldsymbol{a}} = \mathbf{0} && \forall s \in S, \forall a \in \mathcal{A}(s)
\end{aligned}
\tag{6.11}
$$

The Python code used to generate the MIQCP formulation is reported in Appendix D.

For general Convex-MDPs, we repeat here Assumption 2.4:

Assumption 6.1. ***Joint-Convexity.*** *Given a Convex-MDP $\mathcal{M}_C$, for all convex uncertainty sets $\mathcal{F}_s^a \in \mathcal{F}$, the dual function $g(\boldsymbol{\lambda}_s^a, \mathbf{x})$ in Constraint (6.7) is jointly-convex in both $\boldsymbol{\lambda}_{\boldsymbol{s}}^{\boldsymbol{a}}$ and $\mathbf{x}$.*

Remark 6.3. *We note that we can combine models of uncertainty different from one another within a single formulation of Problem (6.9), since each instance of Constraint (6.7) is independent from the others. As an example, if both the interval and ellipsoidal models are used, the overall formulation of Problem (6.9) is an MIQCP.*

6.5.2 Verification Engine

In this section, we give details about the mathematical formulation for the verification engine (VE), i.e., the routine responsible to determine if the analyzed Convex-MDP satisfies specification ϕ for any resolution of uncertainty when operating under the control of strategy σ_c.

After fixing the candidate strategy σ_c returned by the OE, all the non-determinism in $\mathcal{M}_\mathcal{C}$ is resolved. We can thus reduce $\mathcal{M}_\mathcal{C}$ to the *induced* Convex-MC $\mathcal{M}_\mathcal{C}^{\sigma_c} = (S, S_0, \Omega, \mathcal{F}, \mathcal{X}, L)$. The VE can then be run on $\mathcal{M}_\mathcal{C}^{\sigma_c}$ to determine whether it satisfies the PCTL specification ϕ for all resolutions of uncertainty, i.e., $\forall \eta^a \in Nat$. To accomplish this task, we use the sound and complete model-checking algorithm presented in Chapter 4, and run it to determine whether $\mathcal{M}_\mathcal{C}^{\sigma_c} \models_{Nat} \phi$.

If the Convex-MC satisfies ϕ, then the optimal strategy has been found and $\sigma^* = \sigma_c$. Otherwise, the VE needs to generate an additional constraint to be passed to the OE, so that the same candidate solution does not get selected anymore. If vector $\mathbf{z_c} = [z^a_{s_0} \cdots z^a_{s_{N-1}}]$ collects all the binary decision variables that were set to zero in the previous round of optimization, i.e., the variables corresponding to the actions selected at each state under strategy σ_c, we simply need to add the following constraint:

$$\sum_{i=0}^{N-1} \mathbf{z_c}[i] \geq 1 \tag{6.12}$$

in order to prevent the selection of the same set of actions again. Intuitively, Constraint (6.12) guarantees that at least one action from the previously selected set does not get selected any more (its value is set to 1) in the subsequent iterations.

As an example, we optimize the total expected reward of the Ellipsoidal-MDP in Figure 2.5 subject to specification $\phi = \boldsymbol{P}_{\geq 0.8}[\vartheta\ \boldsymbol{\mathcal{U}}\ abs]$. The first iteration of the OE generates strategy σ_{c_1}, which selects actions $[b, a, a, a]$ for states $[s_0 \cdots s_3]$, and which results in $\mathbb{W}^{\sigma_{c_1}}_{s_0} = 10.625$. On the other hand, the VE reports $P^{\sigma_{c_1}, min}_{s_0}[\vartheta\ \boldsymbol{\mathcal{U}}\ abs] = 0.207$, so strategy σ_{c_1} is rejected. The VE adds the constraint $z^b_{s_0} + z^a_{s_1} + z^a_{s_2} + z^a_{s_3} \geq 1$ to the OE formulation. At the second iteration, the OE generates strategy σ_{c_2} which selects actions $[a, a, a, a]$, and results in $\mathbb{W}^{\sigma_{c2}}_{s_0} = 10.188$. The VE computes $P^{\sigma_{c_2}, min}_{s_0}[\vartheta\ \boldsymbol{\mathcal{U}}\ abs] = 1$, which satisfies specification ϕ. The algorithm thus terminates reporting $\sigma^* = \sigma_{c_2}$. The full formulation of the MIQCP problem used to generate the optimal strategy is reported in Appendix D.

6.5.3 Algorithm Analysis

In this section, we prove the *soundness* and *completeness* of the proposed strategy-synthesis algorithm and we analyze its runtime performance. We summarize the results of this section in the following theorem.

Theorem 6.2. *The algorithm presented in Section 6.5 to solve the strategy-synthesis problem introduced in Definition (6.2) is sound, complete and has worst-case runtime exponential in the size $\mathcal{R}$ of the Convex-MDP $\mathcal{M}_\mathcal{C}$ and polynomial in the size $\mathcal{Q}$ of the PCTL specification ϕ, if $\mathcal{M}_\mathcal{C}$ satisfies Assumption 6.1.*

Proof. To prove soundness, we need to show that the strategy σ^* returned in a run of the algorithm reporting success is indeed the strategy that maximizes the total expected reward $\mathbb{W}^\sigma_{S_0}$ of the Convex-MDP among those that satisfy the PCTL specification ϕ under all resolutions of uncertainty. Problem (6.9) returns the MD strategy σ_c that maximizes $\mathbb{W}^\sigma_{S_0}$ among those still available.

By Lemma 6.1, the VE is sound, so if it returns that $\mathcal{M}_{\mathcal{C}}, \sigma_c \models_{Nat} \phi$, indeed $\sigma^* = \sigma_c$. This exit condition is represented by the exit arrow at the bottom of Figure 6.4. By Lemma 6.1, the VE is also complete, so if it returns $\mathcal{M}_{\mathcal{C}}, \sigma_c \not\models_{Nat} \phi$, the current σ_c under analysis can indeed be discarded, since it cannot be the solution of the synthesis problem. This is done by generating a constraint of the form of Constraint (6.12), which removes only the current σ_c from the strategies to be explored by the OE. This proves the soundness of the overall algorithm.

To prove completeness, we need to show that indeed no strategy $\sigma \in \Sigma^{MD}$ fulfills the PCTL specification ϕ if the synthesis algorithm reports failure. We can do so by realizing that it is only the OE to declare failure of finding a solution. This happens when Problem (6.9) becomes *unfeasible* because all the available strategies $\sigma \in \Sigma^{MD}$ have previously been discarded by the VE. This exit condition is represented by the exit arrow at the top of Figure 6.4. This proves the completeness of the overall algorithm.

Finally, we evaluate the worst-case runtime of the algorithm by counting the maximum number of required iterations and then analyzing the computation requirements to perform each iteration. To ease the reader in following the derivation below, we recall from Section 2.1.2 that M is the cardinality of the set of actions A available in the Convex-MDP, and N is the cardinality of the set of states S of the Convex-MDP.

To evaluate the total number of iterations, we need to determine the cardinality of the set Σ^{MD} of all the available MD strategies. There are in total $I = |\mathcal{A}_{s_0}| \times |\mathcal{A}_{s_1}| \times \cdots \times |\mathcal{A}_{s_{N-1}}| = O(M^N)$ MD strategies, i.e., all possible permutations of the actions available at each state $s \in S$. We set $|\Sigma^{MD}| = I$ and the algorithm goes at most through $I = O(M^N)$ iterations.

Each iteration of the algorithm requires solving an instance of Problem (6.9) and a verification check in the VE. Problem (6.9) can be solved by branch-and-bound algorithms in time exponential in the number of binary variables (whose number is $O(MN)$ and it remains constant across iterations) and polynomial in the number of constraints. The number of constraints is polynomial in $\mathcal{R}$ at the first iteration, and it grows by one at each iteration for a maximum number of additional constraints limited by $I = O(M^N)$. Overall, the complexity of solving an instance of Problem (6.9) is $O\left(2^{MN} \times poly(M^N)\right)$. Furthermore, the verification check in the VE can be done in time polynomial in $\mathcal{R}$ and $\mathcal{Q}$ by Lemma 6.1, if $\mathcal{M}_{\mathcal{C}}$ satisfies Assumption 6.1.

Finally, we conclude that the worst-case runtime of the proposed strategy-synthesis algorithm is $O(M^N \times \left(2^{MN} \times poly(M^N) + poly(\mathcal{Q})\right))$, which is exponential in $\mathcal{R}$ and polynomial in $\mathcal{Q}$. □

The analysis of the algorithmic runtime shows that the algorithm performs better on problems that do have a feasible solution, arguably the most interesting ones, since they do not require to visit all the MD strategies $\sigma \in \Sigma^{MD}$. On the other hand, the optimization step could be removed to save time if the goal was to prove unfeasibility.

We further notice that, as an alternative to our approach, σ^* could be determined by testing all I available MD strategies, and selecting the one with the highest reward among those satisfying ϕ, as hinted to by Baier et al. [16]. We believe (and experimentally show in Section 7.4) that our approach can achieve shorter runtime by partitioning the overall strategy-synthesis problem into an optimization and a verification step and by testing strategies in order of optimality.

Finally, speed-ups can be obtained by implementing online routines for integer-constraint simplification and to produce more succinct certificates of unfeasibility from the VE [71, 126]. The description and implementation of these routines are outside the scope of this dissertation and are left as an interesting direction of future work to improve the scalability of the proposed algorithm to the analysis of larger problems.

Chapter 7

Optimal Energy Scheduling and Pricing in Smart-Grids with Renewable Sources

In this chapter, we describe how we applied the proposed synthesis algorithm to the problem of generating optimal energy pricing and purchasing strategies for a per-profit energy aggregator whose portfolio of energy supplies includes renewable sources, e.g., wind. We first introduce the analyzed problem and motivate its relevance in the context of the energy-market structure. Secondly, we review related solutions proposed in the literature to set a comparison point with the proposed approach. We then present the Convex-MDP model that we developed to capture the decision-making process that is set up by the energy aggregator to maximize its economic profits while accounting for the fluctuations in the availability of the renewable sources. In particular, we show that the model is capable of capturing the uncertainties in the prediction of the availability of wind-generated energy. We conclude by reporting experimental results on the performance of the energy aggregator operating under the guidance of the optimal strategy synthesized using the proposed control algorithm. Results show both that the proposed analysis can help in gaining further insight in the economic structure of the decision process and that, in the average case, the energy aggregator can pursue market strategies that are more robust to uncertainties in the wind prediction with respect to other solutions proposed in the literature.

7.1 Problem Description

In this section, we first define the analyzed problem and we then summarize the contributions presented in this chapter.

As stated in a recent report from the International Energy Agency, electricity consumption world-wide is projected to grow from 18 trillion kWh in 2006 to 32 trillion kWh in 2030, a 77% increase [82]. To avoid catastrophic pollution damage to the planet, it is necessary to employ

energy sources alternative to fossil fuels [65]. In fact, an acceleration in the deployment of renewables is already taking place in Europe, Asia and North America. In this chapter, we focus on wind energy, which nowadays has higher capacity than solar energy, and is expected to constitute a significant portion of renewable generation integrated to the power grids of North America [65]. We nevertheless note that our approach is easily extendable also to solar energy, should the application require it.

The correct operation of power systems requires the balance between energy supply and demand at all times. The operation *risk* for the power-network can be quantified both by the probability of not meeting such a balance constraint, and by the (positive) gap between demand and supply. High values of either indicator make the occurrence of disruptions, faults and ultimately blackouts more likely [167]. In grids that only integrate fossil energy sources, the task for the system operator amounts to *dispatch* the production of energy during the day, based on averaged demand profiles.

This topic has been studied in several publications due to its relevance both from a political and an economic perspective. As a first approach, *deterministic* optimization frameworks have been developed to solve the energy dispatch problem, aiming to maximize the *social welfare* of the different agents (suppliers and consumers) operating in the energy market. Such an optimization translates into finding an equilibrium point in which no system agent can achieve higher benefits (e.g., lower energy cost for the consumers) without damaging another agent in the system. Furthermore, the optimization was carried out while also enforcing constraints on the resiliency of the network in the presence of one fault in the network, the so-called *N-1* worst-case dispatch [167].

When considering the need for limiting operation risk, it is apparent that a high penetration of wind generation puts forth big challenges [133]. Unlike fossil energy resources, wind generation is non-dispatchable, i.e., it cannot be harvested by request. Further, wind availability exhibits high variability across all timescales, which makes it difficult to forecast (errors can be up to 20% of the forecast value [138]).

To compensate for supply uncertainty, researchers have proposed the concept of *demand response* (also known as demand side management (DSM)), i.e., the management of customer energy consumption in response to supply conditions [76, 89, 92, 130, 156]. In these studies, it was implicitly assumed that every building (or even apartment) is equipped with a *smart meter* connected to the power grid. The smart meter communicates in real time with the grid and it is capable of rescheduling tasks that require energy consumption depending on the energy *price*. Indeed, a large fraction of the total daily electricity consumption in the U.S. is from residential and small commercial energy users, e.g., water heaters and dish washers, which do not need to operate at a specific moment but only within a time interval, e.g., some time overnight [138]. Traditionally, these energy users pay a fixed price per unit of electricity, which represents an average cost of power generation over a given time-frame (e.g., a season). In smart grids with two-way communications, *real-time* pricing protocols can be implemented so that price can vary according to the availability of energy supply, to incentivize (disincentivize) energy demand [28]. As a simplified example, we could foresee a scenario in which the smart meter does schedule the load only when the energy price communicated by the smart grid goes below some (user-) preset threshold.

A wealth of *stochastic* optimization frameworks have thus been proposed [25, 34, 45, 76, 92,

121, 123, 131, 132, 133, 145, 146, 154, 157, 161, 164, 171, 175]. These works aim first to optimize the dispatch of non-renewable baseline energy to be integrated with the renewable energy supply to meet the total demand. In fact, in current power-networks, wind penetration usually accounts only up to 30-40% of the total energy generation, so fossil generation is still required. The goal of these stochastic frameworks is still to maximize the *expected value* of the social welfare of the agents participating in the energy market. On the other hand, they employ stochastic optimization techniques to guarantee the satisfaction of the power balance constraint despite the uncertainty in the predictions of wind availability. More recent works also include the co-synthesis of optimal energy pricing strategies to manage the customer consumption through economic incentives [76, 92] or the rescheduling of customer loads to maintain more constant load profiles [89, 130, 156].

The scenario described so far refers to the management of smart-grids at the national (or state) level by an Independent System Operator (ISO). Such operation is generally controlled by well-defined regulations (e.g., set by the Federal Energy Regulatory Commission (FERC) [58] in the United States). ISOs (or energy suppliers) are non-profit organizations which aim to maximize the social welfare of the agents participating in the system and which, in the United States, set energy prices following the principle of "Just and Reasonable rates" mandated in the Federal Power Act Section 205.

At the regional or local level, instead, a wider variety of businesses and organizations operate within the energy market. In this chapter, we will consider the interaction of an *energy aggregator* (or *energy broker*) with the energy market. An energy aggregator can be a for-profit company acting as a middleman or negotiator on behalf of a group of customers, to obtain the best deal or contract terms from the energy suppliers operating the power-network. Examples of such organizations in the United States include ENERNOC [55], Opower [127] and GoodEnergy [64], just to name a few.

In particular, we will focus on the problem of generating optimal energy pricing and purchasing strategies for a for-profit energy aggregator whose portfolio of energy supplies includes renewable sources, e.g., wind. We consider a micro-system in which the energy aggregator purchases energy from the energy market and sell energy products to the users that have subscribed to its services. Being a for-profit organization, the energy aggregator aims to maximize its own economic profit. In fact, regulations do not strictly dictate the behavior of private agents operating in the energy market. On the other hand, the aggregator operates in a competitive market, so it needs to wisely set its energy pricing and purchasing strategies in order to successfully remain in business.

Similarly to the behavior of the full smart-grid, we assume that, at all times, the aggregator decides to maintain a balance between supply and demand within its portfolio. We consider such a constraint as a reasonable choice to be included in the aggregator business plan, to reduce the variability of its portfolio. We note in any case that such a requirement is not as stringent for an energy aggregator as it is instead at the full smart-grid level, since the energy aggregator usually operates only on a fraction of the total market and it can potentially buy or sell energy with a short notice on the spot energy market.

The scenario analyzed in this chapter is similar in spirit to the work analyzed by He et al. [76], since also that work aims to optimize the economic profit of the agent under consideration.

That work though does not explicitly consider the risk of power unbalance during optimiza-

tion, and only evaluate the probability of loss of load (also known as load shedding) *after* synthesis, via Monte Carlo simulation, to evaluate the quality of the proposed solution. Unfortunately, these results offer little insight to the aggregate when the estimated risk is too high. Indeed, Varaiya et al. [167] advocated the need for a stochastic optimization framework capable of bounding the risk *a priori*, i.e., at optimization time. The returned control strategy can then be considered the optimal one among those that guarantee an acceptable risk of power unbalance.

Moreover, more constraints to the optimization problem need to be added to guarantee that a minimum amount of energy gets actually delivered to the users. Without such a constraint, the energy aggregator could potentially over-increase the energy price to force users out of the system. This behavior of the energy aggregator would indeed guarantee power balance at times of little wind generation, but it would be *unfair* to the service subscribers. In the following of the chapter, we will refer to this minimum guaranteed delivered energy as Quality of Service (QoS) for the users.

The scenario depicted in this chapter represents only a small portion of the intricacies of the energy market. As future work, we plan to incorporate the interaction between the energy aggregator and the full market structure to more accurately model the complex real-world application.

We summarize in the following the main contributions of this chapter.

7.1.1 Contributions

Our **first contribution** is a novel stochastic model capable of capturing the decision process set up by an energy aggregator to generate energy pricing and purchasing strategies in a market structure integrating wind energy sources. The model is an Ellipsoidal Markov Decision Process $\mathcal{M}_E$ (Ellipsoidal-MDP), a special case of Convex-MDP, which was introduced in Section 2.1.3.3. While previous works used analytical distributions, e.g., Gaussian [76], to model uncertainty in wind availability, we use measured data (from the wind farm at Lake Benton, Minnesota, USA [170]), to train a likelihood model of the wind generation. Moreover, the statistical framework presented in Section 2.1.3 gives quantitative means to represent the confidence in the forecast values. We then approximate the likelihood region with an ellipsoidal model, which is more accurate than the linear ones often used in the literature, while remaining computationally tractable. Our empirical approach has the promise of more faithfully representing the probability distribution of the generated energy because it is tailored to the specific wind farm under analysis, and it is robust to forecast errors.

As a **second contribution**, we cast the strategy-synthesis problem as a constrained optimization problem for Ellipsoidal-MDPs. The optimization aims to maximize the economic profits for the energy aggregator, while constraints limit the risk of power unbalance and guarantee the desired QoS for the users. We show how to formulate the constraints in terms of a PCTL specification and use the total expected reward to measure the profit of the energy aggregator. We can thus use the strategy-synthesis algorithm presented in Chapter 6 to solve the problem. We focus on finite-horizon History-dependent Deterministic (HD) strategies, i.e., for each state of the Ellipsoidal-MDP an optimal action to take is chosen deterministically, based on the entire (finite) execution history of the decision process. The limitation to finite-horizon strategies is not restrictive, since

energy pricing and purchasing decisions are taken on a daily basis. As explained in Section 6.2, History-dependent Random (HR) strategies are in general more powerful, i.e., they can produce a higher expected reward. Nevertheless, we focus on deterministic strategies because we believe that deterministic pricing strategies are easier to adopt in a real-world scenario, since they can be better understood by the system agents (e.g., the household users). Using the reduction introduced in Section 6.3, we then unroll within the model the finite sequence of decision epochs over the day, and construct a second Ellipsoidal-MDP $\mathcal{M}'_E$ in which we replicate the states of the original Ellipsoidal-MDP $\mathcal{M}_E$ at each decision epoch. In the strategy synthesis for $\mathcal{M}'_E$, we focus on Markov deterministic (MD) strategies. As it was shown in Section 6.3, the desired HD strategy for each state s of $\mathcal{M}_E$ can then be reconstructed by collecting the sequence of optimal MD actions for each replica s' of s along the decision epochs of $\mathcal{M}'_E$.

Finally, as our **third contribution**, we experimentally show that the energy pricing and purchasing strategies synthesized using the proposed formulation better fulfill the system specifications and produce a higher expected profit for the energy aggregator than other solutions proposed in the literature [76, 167]. In particular, we use a Monte Carlo simulation to validate our approach. We first synthesize three different control strategies using our approach and the two previously proposed approaches [76, 167]. We then run a Monte Carlo simulation of the system under the control of each of the synthesized strategies and collect the system performance in terms of profit for the energy aggregator, risk of power unbalance and QoS for the users. We finally compute the average of these performances across the Monte Carlo runs to estimate the expected value of these quantities and compare the performances among the three different approaches. In addition, we note that our approach is the only one that allows studying the trade-off between a more accurate forecast of the wind-availability (which is more costly) and the expected profit in managing the energy portfolio, thus it gives further insight to the energy operator about the economic trade-offs that are present in the decision process for optimal pricing and purchasing of renewable energy.

Remark 7.1. *We have chosen to analyze the problem of synthesizing control strategies to be adopted by a for-profit energy aggregator, among other available applications, mainly for two reasons. First, the integration of renewables in the energy supply chain nowadays represents a top priority in the agenda of most developed countries, and a lot of research is being developed to solve a variety of challenges that are still present. This is an exciting field in rapid expansion and efforts on several aspects of the problem will still be needed to make the scenario described in this section truly possible. The development of more effective control techniques for such power networks is one of these challenges.*

Second, the planning of energy pricing and purchasing strategies is an application that can indeed benefit of formal and exhaustive optimization techniques. Given the time scales of the network operation (strategies get roughly update on a daily basis) and the amount of energy processed by a smart-grid (a wind-farm with 100 turbines produces on average 600GWh in a year, enough to power 50,000 households), the runtime and the utilization of computational resources of the proposed strategy-synthesis algorithm are not going to play a crucial role to determine the applicability of this approach to the problem. On the other hand, performance improvements of even a few percents enabled by exhaustive optimization techniques can quickly become relevant, if

considered in absolute terms. Continuing the previous example, an improvement of performance of one wind-farm by only 1% might result in 500 more households to be powered by renewable energy!

7.2 Related Work

In this section, we review related approaches proposed in the literature to solve the more general problem of energy pricing and dispatch in smart-grids that integrate renewables. For each analyzed work, we briefly describe the proposed strategy-synthesis technique.

The problem of optimal energy dispatch in power-grids has been studied for a long time, starting from networks which integrated only non-renewable supply sources [91]. In particular, the first proposed techniques used a *deterministic* approach, in which all system quantities were supposed to be known with certainty and in which the system dynamics were assumed to evolve in a repeatable way. The goal of these techniques was to synthesize the network structure and to dispatch enough reserve energy supply to guarantee that the overall network could sustain the failure of *any* of its components (either a generator or a transmission line) without compromising its correct functionality, i.e., with no interruption of the service. In these frameworks, it was assumed that only one failure, also called *contingency*, could happen at a single time, since the probability of the occurrence of a second failure before the first one could be repaired was deemed to be negligible. Not surprisingly, these techniques were referred to as *N-1 contingency analysis*, where N is the number of network components. Moreover, the optimization problem aimed to maximize the social welfare of the agents operating in the system. These deterministic approaches had the advantage of allowing more compact formulations of the strategy-synthesis problems, and faster solution times. On the other hand, they could not be trivially extended to the analysis of power networks integrating renewable sources, because of the obvious shortcoming of not being able to capture the variability in the availability of renewables. In particular, deterministic approaches could work well only when the renewables penetration, i.e., the percentage of energy provided by renewable sources, was just a small fraction of the total supplied energy [167].

In order to synthesize control strategies also in smart-grids with higher penetration of renewables, a wealth of stochastic approaches have been proposed [25, 34, 45, 76, 92, 121, 123, 131, 132, 133, 145, 146, 154, 157, 161, 164, 171, 175]. Two sources of stochastic behavior were modeled. First and foremost, the availability of renewable energy, which can vary substantially even between consecutive decision epochs and which cannot be dispatched but only harvested when available. Due to such variability, reserve non-renewable energy supplies need to be scheduled to guarantee power balance at all times and their dispatch needs to be optimized to maximize the aggregate *social welfare* of the system agents [29, 105, 133]. Secondly, the user demand, which started being considered not only as an environmental variable which the smart-grid operator only had to react to, but also as a control knob to help guaranteeing power balance in the grid at all times [89, 92, 115, 155]. Moreover, both these two sources of stochastic behavior started being considered at the same time to obtain even higher performances from the smart-grid [76, 130, 167]. The results obtained employing these approaches showed that it is indeed possible to allow higher

penetration of renewables by adopting a stochastic synthesis framework.

Our approach focuses on a portion of the complex structure of the energy market and considers the problem of generating optimal energy pricing and purchasing strategies for a for-profit energy aggregator whose portfolio of energy supplies includes renewable sources. Our contribution expands previously proposed stochastic approaches because it not only considers the randomness in the prediction of availability of renewables and user demand, but it also allows capturing mathematically the uncertainty in modeling such random predictions. In fact, as it will be shown in Section 7.4, our technique allows to quantitatively assess the impact of uncertainties in the predictions on the decision process, offering the energy aggregator a more comprehensive analysis of the economic and functional dynamics of the system.

Due to their higher relevance to the techniques proposed in this chapter, in the following we will focus only on stochastic optimization frameworks and present an overview of the most influential works presented in the literature.

7.2.1 Stochastic strategy-synthesis Frameworks

Bouffard et al. [29] solved the problem of energy dispatch in smart-grids integrating renewable sources by adapting previously proposed (by the same authors) stochastic techniques for the scheduling of supply reserves in networks where one of the generators can fail. In fact, they consider the wind variability analogous to a "fault" (or malfunction) of one of the dispatchable energy supplies, since in both cases the generated supply is lower than expected and energy reserves need to be scheduled to counteract to this contingency. Analogously to our approach, they define a finite number of possible *scenarios*, i.e., a sequence of pairs of real numbers expressing the wind availability and the user demand at each decision epoch (in our setting, a scenario is an *execution path* of the Ellipsoidal-MDP), and compute the probability of each scenario to occur. They then cast the energy dispatch problem into a constrained optimization problem. Their cost function tries to minimize the expected value of a *social welfare* cost function, which includes the network operator costs, the energy price for users and the amount of "load-shedding", i.e., the amount of energy that is not delivered to the users because it exceeds the available supply. Constraints are then added to guarantee the power balance and the operation within the thermal and voltage ratings of the network (this set of constraints is referred to as "security") at all times and that each supply generator operates within its correct functional regime. The constrained optimization is run over all scenarios at the same time and each quantity is weighted by the probability for the corresponding scenario to take place. In this way, scenarios that are less likely to happen are allowed to result in worse performances in terms of profits for the network operator and costs for the users.

Differently from our approach, they consider the expected value of the user demand as an environmental variable, i.e., fixed externally and *not controllable* by the network operator, when determining the available operating scenarios. At optimization time, they then assume that the operator can instead *deterministically* finely tune the actual user demand using economic incentives. We take an opposite approach and assume that the energy aggregator can *control* the *expected value* of user demand by using economic incentives, but that the *actual realization* of user demand is *stochastic* and non-controllable by the aggregator. Moreover, our framework not only considers

a finite number of scenarios about the availability of renewable energy sources, but it also allows capturing mathematically the uncertainty in creating such a finite discretization of the continuous random variable representing the availability of renewables.

The problem of optimal energy dispatch (or unit commitment) was also extensively analyzed by Papavasiliou and Oren [132, 131] and Papavasiliou et al. [133]. These works expand the contributions by Bouffard et al. [29] in at least three ways. First, they consider a larger scale centralized unit commitment problem, aiming to provide a complete strategy for the Independent System Operator (ISO) spanning a longer time horizon and a wider network size. In order to do so, as a second contribution, these authors propose novel stochastic optimization techniques to allow higher scalability of the proposed synthesis algorithm. In particular, they propose a two-stage Lagrangian relaxation algorithm which was experimentally proven to better spread the computation load across subproblems than other solutions presented in the literature [52, 67, 77]. Furthermore, as a third contribution, they propose a more accurate model of the available wind-energy availability, based on the inverse Gaussian distribution.

In our work, we consider uncertainty sets to model the variability in the available wind energy, which are derived starting from empirical data. Our approach has the promise of being independent to any specific analytical distribution, thus potentially better tailoring the specific characteristics of the wind mill farm under analysis.

The effectiveness of adopting techniques to manage *user demand* via economic incentives to help balancing supply and demand in power grids was studied, among others, by Koch et al. [92] and by Lu et al. [115]. In particular, these researchers focused on the analysis of Thermostatically Controlled Loads (TCLs), e.g., refrigerators, which are usually controlled by a dead-band hysteretic (ON/OFF) control. In short, the main idea is to devise techniques to coordinate the functionality of multiple TCLs and to spread their ON times by *load shifting*, so that not all loads are ON (OFF) at the same time, causing spikes (droops) in the energy demand. While from the user perspective such a load shifting would have negligible consequences, the dispatch of energy sources by the network operator could be substantially simplified.

While these approaches share a common problem description, they differ in the way the solve the optimal control problem. Koch et al. [92] formulate the control problem in terms of a Model Predictive Control (MPC) framework. The underlying model of the system is a linear time-invariant representation of a population of TCLs. The system state captures the distribution of the population across discretized temperature-related bins, and the system evolves with linear dynamics across the different states, depending on how many loads are ON/OFF at a given time. The optimal control strategy is inferred in a receding-horizon fashion by predicting at all decision epochs the future evolution of the system and by turning ON/OFF the loads in order to achieve the maximum spread of the ON times while guaranteeing that all loads remain below the target maximum temperature.

Lu et al. [115] instead develop a state-queuing model of TCLs capable of capturing the dynamics of the load power consumption in response to different load shifting strategies, in order to evaluate the economic benefits and feasibility of each strategy. The advantage of this approach is its high expressivity and the researchers show how to embed into the model information about the

environment temperature and about the randomness of the user interaction with the system (e.g., a user might increase the temperature of a refrigerator by opening its door), in order to obtain more accurate estimations of the system dynamics. The model is then simulated for different available load shifting strategies and the optimal one is selected.

Three different techniques for demand response were analyzed by Papavasiliou and Oren [130]. In particular, the authors focus on the *deferrable* nature of several demand response resources (e.g., the charging of electrical vehicles). The first analyzed strategy, which is used as a comparison point, considers the *centralized co-optimization* of generation and demand by the system operator. Although this scenario is not achievable in practice, it sets a reference of the best attainable performances of the grid. As a second strategy, they analyze *demand bids* with real-time pricing [28], in which part of the total demand is *elastic*, i.e., variable, to changes in the energy price. The system operator can thus partially influence the total demand by changing the energy price. As a third strategy, the authors study the *coupling* of renewable suppliers with deferrable loads. In this scenario, an energy aggregator can purchase renewable energy and it then manages a set of loads. If the supply is below the demand, it can defer some of the loads (by paying a penalty) within a maximum time window or purchase more energy from the real-time wholesale market in case the maximum time window for deferral has expired. This approach distributes the centralized co-optimization of the first strategy and thus it makes it more realistic to be implemented in practice. The comparison among the three strategies show that demand-side bidding results in lower cost with respect to coupling, but it fails to capture the user elasticity across different time windows thus resulting in higher load losses.

The scenario presented by He et al. [76] is the one that we follow most closely in setting up our strategy-synthesis framework. In that work, the researchers consider both the availability of renewable energy and the user demand as stochastic processes, and they assume that the system operator can *control* the expected value of user demand by using economic incentives. The decision process evolves on two time scales. On *day-ahead*, the network operator can decide how much baseline non-renewable energy to schedule for the following day. In *real-time*, after observing the realization of wind availability, it further sets the energy price for users to incentivize (disincentivize) their demand, and it schedules further *fast-start* non-renewable supply, if needed, to guarantee power balance at all times. The strategy-synthesis problem is then cast as an *unconstrained* optimization problem and solved analytically.

We follow very closely the same setup, but we add *constraints* to the optimization problem to guarantee acceptable values of risk of power unbalance for the system operator and quality of service to the users at optimization time. Indeed, the approach proposed by He et al. could only evaluate system performance *after* the optimization, via Monte Carlo simulation. Such an approach thus gives little insight to the energy aggregator on how to manage its portfolio if the synthesized performances are not acceptable, and domain expertise and trial-and-error on the optimization inputs would be needed to obtain admissible results. Our approach has instead the advantage of automatically exploring the search space and detect the optimal strategy among those that satisfy all system specifications.

In our work, we aim to answer the call by Varaiya et al. [167] to develop strategy-synthesis techniques capable of quantitatively constraining the risk for the energy aggregator at optimization time. Also in that work, the researchers consider the randomness of both the renewable energy availability and of the user demand and they investigate pricing strategies to manage user demand in order to help the operator to guarantee power balance at all times. With respect to the formulation proposed by He et al. [76], they also enforce limits at optimization time on the maximum acceptable probability of loss of load. They then cast the optimization problem as a recursion on multiple time scales, and solve the problem analytically when the probability of loss of load is enforced to be equal to 0.

Our approach extends this technique in at least two ways. First, the proposed strategy-synthesis algorithm also processes system specifications for which the satisfaction threshold is an arbitrary real number p in the closed interval $p \in [0, 1]$. Second, our approach is capable of considering more complex specifications, expressed using arbitrary reward structures. In particular, we will put constraints at optimization time not only on the Loss-of-Load Probability (LoLP) as in the work by Varaiya et al. [167], but also on the *quantitative* amount of Expected Energy Not Served (EENS) and on the Quality-of-Service (QoS) for the service subscribers. Moreover, our framework is easily extendible to consider even further specifications if needed because it can process arbitrary PCTL specifications, as explained in Chapter 6.

7.3 Proposed Model

In this section, we present the Convex-MDP model and the formulation of the strategy-synthesis problem that we used to generate energy pricing and purchasing strategies that are robust to uncertainties in the forecast of the renewable energy supplies. All the symbols used in the case study are collected in Table 7.1.

While many possible pricing strategies have been proposed, in the following we will mainly refer to the scenarios presented by He et al. [76], Stoft et al. [160] and Varaiya et al. [167], because the scenario depicted in those works is the closest to our setup of the synthesis problem.

The analyzed scenario is sketched in Figure 7.1. Three agents operate in the system:

1. **Energy Aggregator.** The energy aggregator manages a portfolio of energy supply sources and it provides energy to a pool of customers that subscribe to its services. Its goal is to maximize its profit, while guaranteeing that, at all times, the aggregate energy supply matches the demand of the users subscribed to the service, to avoid disruptions in the service or excessive economic losses due to the need of selling or buying energy on the spot market at the last minute.

2. **Traditional Users.** Traditional users are users of the network whose behavior more closely resembles the one of users nowadays (e.g., households). These users can decide *a priori* (e.g., when signing the contract) how much energy to use based on its expected price, which is computed based on the average cost of power generation over a given time-frame (e.g., a season). On the other hand, traditional users cannot adjust their energy consumption in

Table 7.1: Table of symbols for the energy pricing and purchasing case study

W	Wind energy (stochastic variable)
μ_W	Empirical probability distribution of W
w	Observation of W
D_t	Traditional-user demand (stochastic variable)
d_t	Observation of D_t
D_o	Opportunistic-user demand (stochastic variable)
d_o	Observation of D_o
T_1-slot	Unit day-ahead decision timeframe
T_2-slot	Unit real-time decision timeframe
K	Number of T_2-slots in a T_1-slot
u	Unit energy price for traditional users
v	Unit energy price for opportunistic users
v_{obs}	Actual value of v set in real-time
Q	baseline energy supply purchased for a T_1-slot
q	baseline energy supply purchased for a T_2-slot
f	fast-start energy supply purchased for a T_2-slot
f_{obs}	Actual value of f set in real-time
c_1	Unit cost of baseline energy
c_2	Unit cost of fast-start energy
c_p	Unit cancellation cost of baseline energy
$\mathcal{M}_E$	Ellipsoidal-MDP used to model the energy pricing and purchasing problem
$\mathcal{M}'_E$	Unfolded version of $\mathcal{M}_E$ across the K T_2-slots
FS	Maximum fast-start energy available to purchase in a T_2-slot
γ_t	Elasticity of traditional users
γ_o	Elasticity of opportunistic users
Δ_k	Surplus of supply on demand
$EENS_M$	Maximum allowed value of Expected Energy Not Served
$LoLP_M$	Maximum allowed value of Loss of Load Probability
QoS_m	Minimum allowed value of QoS to guarantee to the users

real-time in case the actual energy price is raised by the energy aggregator. While users nowadays sign contracts that hold unchanged for months (or years), we will consider the more general case in which traditional users can decide on a daily basis how much energy they are going to use in the *following* day. Nevertheless, they will not be able to adaptively change their energy consumption in *real-time*, i.e., right before scheduling the load. We will use the stochastic variable D_t to refer to the energy demand of traditional users, and the positive real number $d_t \in \mathbb{R}_+$ to refer to an observation of such demand.

3. **Opportunistic Users.** Opportunistic users differ from traditional users because they are

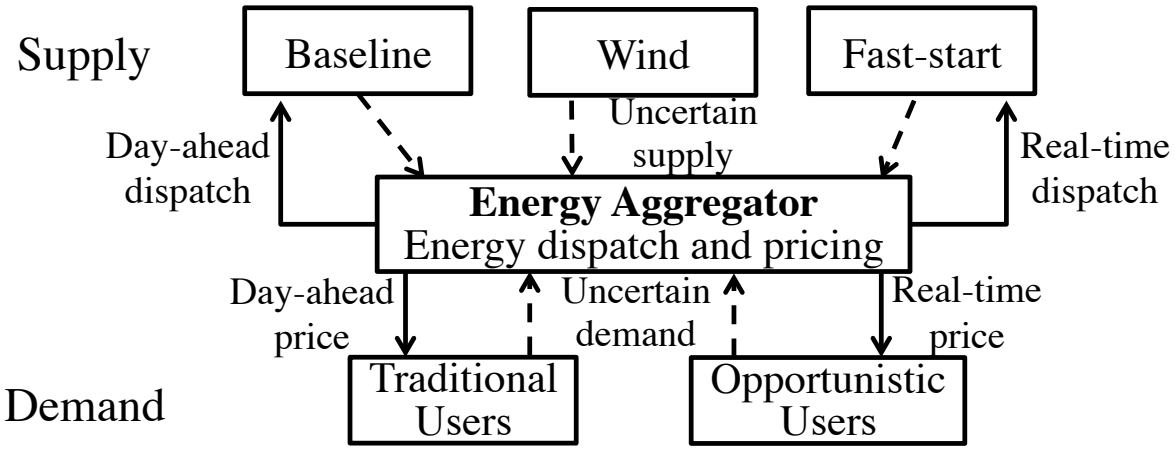


Figure 7.1: The figure shows the input data available to the energy aggregator to guide its decision process (dashed) and the control actions of the aggregator (solid).

capable of rescheduling in *real-time* their energy demand, depending on the energy price. At the time of signing the contract with the energy aggregator, these users accept the possibility of paying higher worst-case energy prices when the supply is lower, in exchange of lower *expected* prices. We will use the stochastic variable D_o to refer to the energy demand of opportunistic users, and the positive real number $d_o \in \mathbb{R}_+$ to refer to an observation of such demand.

Three energy sources are available to the energy aggregator:

1. **Wind.** Wind is the renewable energy source considered in this case study, although a similar analysis could be carried out for any other renewable energy (e.g., solar). In general, the two characteristics of renewables that are the most relevant for the analyzed problem are: 1) they are a cheap source of energy, so the energy aggregator tries to use them as much as possible, when available; 2) they are non-dispatchable, so their availability can only be forecast. For example, wind energy gets harvested by wind mills in a wind farm. We created a stochastic model to represent the distribution of the available energy produced by a wind farm. In order to train our model, we used experimental data directly measuring the value of *energy* produced by the wind farm. The alternative approach of measuring the wind *speed* and infer analytically the energy produced by the farm is usually less accurate [32], so it was not considered. Moreover, for simplicity, in the following we will not consider the problem of energy curtailment, which occurs when the wind is too strong and the wind mills need to be stopped to avoid damages [29]. We will use the stochastic variable W to refer to the energy generated by the wind, and the positive real number $w \in \mathbb{R}_+$ to refer to an observation of such generated energy.

2. **Baseline Generators.** Baseline generators (e.g., thermal units) are a dispatchable source of energy. Supplies that belong to this category have medium range cost (unit cost, c_1) for

the energy aggregator, but they have slow ramp rates and limited unloaded capacity, so they cannot be used in the event of an unforecast supply droop. We will use variable $q \in \mathbb{R}_+$ to refer to the amount of purchased baseline energy.

3. **Fast-start Generators.** Fast-start generators (e.g., gas turbines) are a dispatchable source of energy. They are the most expensive supply source for the energy aggregator (unit cost, c_2), but they have fast ramp rate, so they can be used in case of emergency to avoid the demand to be in excess of the supply. We will use variable $f \in \mathbb{R}_+$ to refer to the amount of purchased fast-start energy.

We note that, at least in the short term, it is expected that the penetration of renewables in the energy supply of a power-network (i.e., the percentage of energy supplied by renewable sources) will be at most around $30-40\%$, so both renewable and not-renewable sources are needed to cover the whole demand.

To achieve its goals, the energy aggregator needs to take two kinds of decisions. First, it needs to purchase non-renewable sources through bilateral contracts with the ISO or from the wholesale spot market, to guarantee that the aggregate energy supply matches the demand. In other words, it needs to determine the values of q and f, such that:

$$w + q + f = d_t + d_o \tag{7.1}$$

at all times. Second, it needs to set the retail price of energy for its service subscribers, to maximize its profits and incentivize users to increase or decrease consumption depending on the expected energy availability.

As proposed in the literature [76], we assume that energy pricing and scheduling decisions are made on a daily basis. Such a timeframe allows for reasonably accurate predictions of wind availability and gives enough time to each system agent to react to the decisions of the other agents. As a consequence, the decision process needs to span a 24-hour time horizon. In particular, the 24-hour period gets divided into T_1-slots of equal length (e.g., $T_1 = 1$h), and each T_1-slot into K T_2-slots (e.g., $T_2 = 30$min, $K = 2$).

The energy aggregator maximizes its economic profit by taking decisions on two time-scales, *day-ahead* and *real-time*. On day-ahead, for each T_1-slot, it purchases Q units of baseline energy, with $q = Q/K$ units per T_2-slot, and sets the price for traditional users (u), so that they can decide on day-ahead when to schedule their demand in the following day. The choice of u determines the expected demand of traditional users ($\mathbb{E}[D_t]$). In real-time, for each T_2-slot, the aggregator first *observes* the values of traditional-user demand d_t and wind availability w. It then sets the price for opportunistic users (v), which sets the expected demand of opportunistic users ($\mathbb{E}[D_o]$). Third, it purchases more fast-start energy (f) or sells back part of the already purchased baseline energy (q), depending on wind availability and user demand, to balance supply and demand. We notice that it is profitable for the energy aggregator to sell back the over-purchased baseline energy q because by doing so it pays only unit cost c_p instead of c_1 (we model a loss associated to buying and selling back energy). In fact, in real scenarios, $c_p < c_1 < c_2$ and wind-energy is assumed to be free for brevity [76]. The aggregator thus tries to use as much wind energy as possible and to

purchase on *day-ahead* only the exact amount of required baseline energy by paying $c_1 \times q$, not to incur in *real-time* in economic losses ($c_p \times q$) or in the extra cost for fast-start supplies ($c_2 \times f$). Since more profitable strategies might imply a higher reliance on the uncertain wind energy or an increase in the energy prices (u and v), correct system functionality needs limits on the risk of energy unbalance and guarantees on the QoS for the users.

There are three sources of stochastic behavior in the analyzed system: traditional (D_t) and opportunistic (D_o) user demand and wind-energy supply (W). We thus use a stochastic optimization framework. The result of the optimization should return optimal strategies about:

1. the day-ahead decisions (Q and u), and;
2. the real-time decisions (f and v) for *each possible observation* of W and D_t at each T_2-slot.

In real-time, the actual decisions (v_{obs} and f_{obs}) will be taken deterministically among the synthesized ones based on the *observed* values w and d_t, i.e., the actual wind availability and traditional-user demand. Moreover, we notice that, by rearranging Equation (7.1), we can write an expression for the amount of fast-start energy to be purchased:

$$f = max(0, d_t + d_o - w - q) \tag{7.2}$$

which needs to hold at all times to guarantee power balance in the aggregator portfolio. As a consequence, we will not explicitly consider variable f in the following and just compute its value on-the-fly when needed, using Equation (7.2). In summary, we will optimize over one T_1-slot (the decision problem is periodic, so we can run one optimization for each T_1-slot stand-alone), and aim to determine optimal values for Q, u and v.

We use the Ellipsoidal-MDP $\mathcal{M}_E = (S, S_0, A, \Omega, \mathcal{F}, \mathcal{A}, \mathcal{X}, L)$, which we partially sketch in Figure 7.2 (top). The range of values of all problem variables are bounded and uniformly discretized to keep the state and action spaces finite. States $s \in S$ are a tuple $s = (w, d_t, d_o)$, where w, d_t, d_o refer to the observed values of available wind energy and user demand in that state. Since we consider only one T_1-slot per time, we model only one choice of optimal energy purchase Q and pricing for traditional users u, which the aggregator will announce on day-ahead for the corresponding T_1-slot. This decision is taken at the initial state s_0, where the model forks among the available pairs $(Q, u) \in A$. The process then transitions through K decision epochs corresponding to each T_2-slot, as follows. First, values of wind energy (w) and traditional-user demand (d_t) are stochastically chosen according to the corresponding distributions (described below). Note that the expected value of traditional-user demand $\mathbb{E}[D_t]$ depends on the decision u taken in s_0, while the distribution of wind energy W is the same for all decisions (Q, u) and it depends on the forecast values. Second, for each observation of W and D_t, a decision on $v \in A$ is taken[1]. This choice sets the expected value of the opportunistic-user demand $\mathbb{E}[D_o]$. Third, the opportunistic-user demand (d_o) is stochastically chosen. To transition between epochs a new value of wind energy w (the backward arrow in Figure 7.2 (top)) is chosen and the steps repeat.

[1]We note that the data-type of action v is different from the data-type of pair (Q, u), as allowed in Section 2.1.

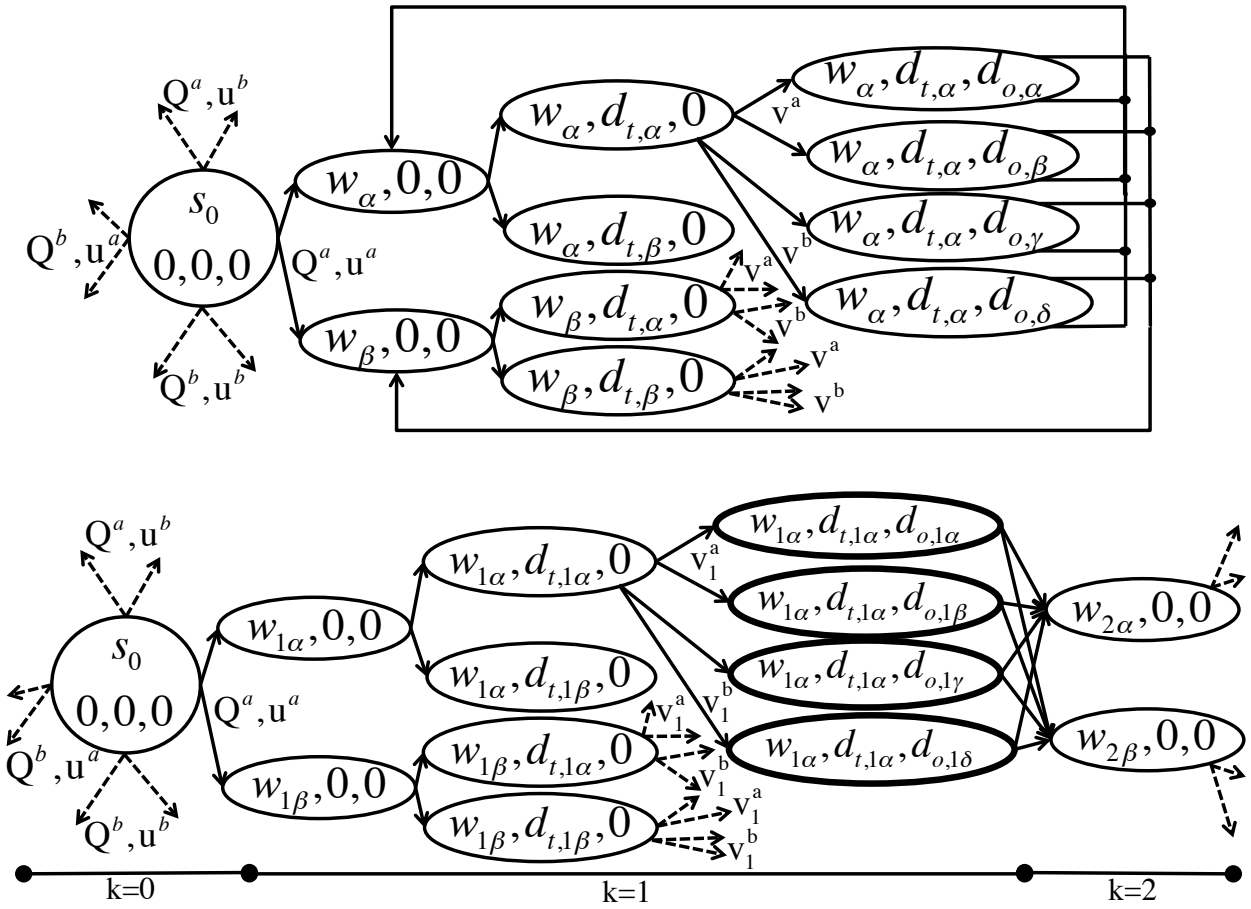


Figure 7.2: Sketch of $\mathcal{M}_E$ (top) and $\mathcal{M}'_E$ (bottom), which is $\mathcal{M}_E$ unrolled across decision epochs. For the two Ellipsoidal-MDPs, the corresponding initial state s_0 is shown on the left. Each state is represented by the tuple (w, d_t, d_o) of observations of W, D_t, D_o. The pairs (Q, u) represent the day-ahead decisions about purchasing of baseline energy Q and energy pricing for traditional users u. Two arrows per decision depart from each state because in this figure we assumed only two *discretization levels* for each quantity (labeled with greek letters $\alpha, \beta, \cdots$). We only show the state graph in the Ellipsoidal-MDPs related to decision (Q^a, u^a), but similar state graphs are present also for the other decision pairs $(Q^a, u^b), (Q^b, u^a), (Q^b, u^b)$, as hinted by the dashed arrows departing from s_0.

In general, the optimal decision v at each epoch depends on previous observations of wind availability (w) and user demand (d_t, d_o), so v is history-dependent. To synthesize control strategies using the algorithm proposed in Chapter 6, we need to unroll the sequence of decision epochs in the Ellipsoidal-MDP $\mathcal{M}'_E = (S', S_0, A', \Omega, \mathcal{F}', \mathcal{A}', \mathcal{X}', L')$, as shown in Figure 7.2 (bottom). In $\mathcal{M}'_E$, we have explicitly marked each quantity with an additional subscript $k = 1, \cdots, K$ to refer to the corresponding T_2-slot. Each state $s \in S$ of $\mathcal{M}_E$ (apart from the initial state) has been replicated K times in $\mathcal{M}'_E$, $s \to s_1, s_2, \cdots, s_K$. After K decision epochs, the states transition to

an absorbing state (not shown in Figure 7.2). Since now all decision epochs are explicitly codified in the model, Markov strategies are optimal for $\mathcal{M}'_E$. The corresponding optimal history-dependent strategy for $\mathcal{M}_E$ can be reconstructed for each state $s \in S$ by collecting the sequence of optimal decisions returned by the algorithm in the replicas $s_1, s_2, \cdots, s_K$. In the following, we will thus only consider $\mathcal{M}'_E$. Moreover, as an additional advantage, different wind distributions can be used to transition between different epochs in $\mathcal{M}'_E$, while only one distribution could be used in $\mathcal{M}_E$. This gives the possibility to account for the time-varying nature of the wind availability also on the finer time scale of the T_2-slot. Overall, the modeling expressivity is thus increased. In Figure 7.3, we give a more detailed example of $\mathcal{M}'_E$ for $K = 2$.

State transition probabilities are computed using the following stochastic models.

User Demand. The demand of both traditional (D_t) and opportunistic (D_o) users is modeled using Gaussian distributions [76], with $D_t \sim \mathcal{N}(\alpha_t u^{\gamma_t}, \beta_t \mathbb{E}[D_t])$, and $D_o \sim \mathcal{N}(\alpha_o v^{\gamma_o}, \beta_o \mathbb{E}[D_o])$. Parameter $\gamma_t < 0$ ($\gamma_o < 0$) is the *elasticity* of the traditional (opportunistic) users, i.e., the ratio of the percentage change of the expected demand to that of price variation. Formally:

$$\gamma_t = \frac{u}{\mathbb{E}[D_t]} \cdot \frac{\partial \mathbb{E}[D_t]}{\partial u} \qquad \gamma_o = \frac{u}{\mathbb{E}[D_o]} \cdot \frac{\partial \mathbb{E}[D_o]}{\partial u}$$

Parameters $\alpha_t, \alpha_o, \beta_t, \beta_o$ are fitting parameters. To compute transition probabilities, we truncate and discretize the continuous probability distributions in equally-sized intervals, pick the middle point of each interval as the discretization value and then integrate the probability distribution across the interval, to determine how likely the system transitions to that discretized value.[2]

Wind-Energy Availability. We created a stochastic model of the available wind energy starting from measured data collected from the wind farm at Lake Benton, Minnesota, USA [170]. The goal is to take forecast values into account, while also considering the intrinsic inaccuracies of these predictions. First, we compute the (discrete) empirical probability distribution μ_W of a training set of collected wind-energy data. Second, we divide a new set of data in T_2-slots, and consider the average value for each new T_2-slot as the *forecast* energy value. We then scale μ_W to have such expected value $\mathbb{E}[W_k]$, thus obtaining μ_{W_k}. Finally, we compute the ellipsoidal Sets (2.7) $\mathcal{F}'^a_s$ (collected in $\mathcal{F}'$) to represent uncertainty in the transition probability between two discretized wind-energy levels in two consecutive T_2-slots. Transition frequencies are computed by counting observed transitions in the training set of data. Further, using classical results from statistics [125], we can compute the value of parameter β^a_s from Set (2.1.3.3) corresponding to a desired confidence level C_L in the measurements. In particular, $0 \leq C_L \leq 1$ and

$$C_L = 1 - cdf_{\chi^2_d}\left(2 * (\beta^a_{s,max} - \beta^a_s)\right)$$

where $cdf_{\chi^2_d}$ is the cumulative density function of the Chi-squared distribution with d degrees of freedom (d is equal to the number of bins used to discretize W). As explained in Section 2.1.3, the

[2]For simplicity, we assume that the distributions of D_t and D_o are known with certainty. In fact, these distributions are usually estimated much more accurately than the wind forecast, by analyzing the history of measured data [76]. Nevertheless, adding uncertainty models also for these distributions can indeed give further insight in the system dynamics and it is left as future work.

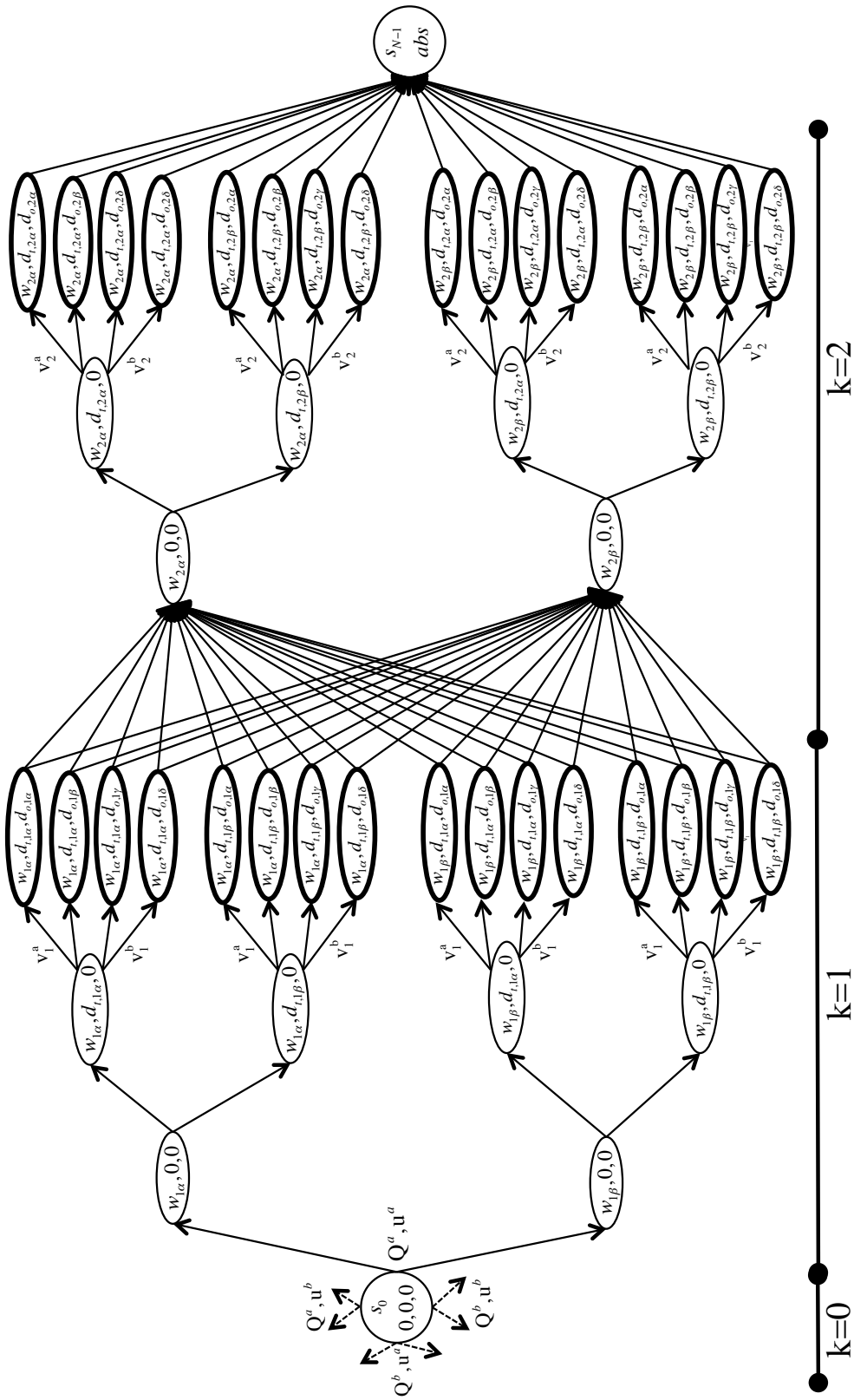


Figure 7.3: The figure shows $\mathcal{M}_E$ for $K = 2$ and two discretization levels for each quantity. We show the full state graph in the Ellipsoidal-MDP related to decision (Q^a, u^a). Similar state graphs are present also for the other decision pairs $(Q^a, u^b), (Q^b, u^a), (Q^b, u^b)$, as hinted by the dashed arrows departing from s_0.

sets $\mathcal{F}_s'^a$ computed using this procedure are second-order approximations of *likelihood* estimators of the wind-energy availability. Moreover, an increasing value of parameter β_s^a, which sets the uncertainty level in wind forecasting, can be used in the sequence of decision epochs to model the fact that forecast farther-away in time are less accurate.

In order to account for system performances, we provide the states with thick circles in Figure 7.2 (bottom) with three reward structures. These structures express the profit and risk of the energy aggregator and the QoS for the users. We choose those states because the quantities $D_{t,k}, D_{o,k}, W_k$ are all fully observable in them, thus allowing the evaluation of the system performances. We set:

$$r_{s,k}^{Profit}[\$] = ud_{t,k} + v_k d_{o,k} - (c_p\Delta_k + c_1(q-\Delta_k))\mathbf{1}_{\Delta_k \geq 0}^A - (c_1 q - c_2\Delta_k)\mathbf{1}_{\Delta_k<0}^B \tag{7.3a}$$

$$r_{s,k}^{LoL}[\text{MWh}] = max(0, X\mathbb{E}[W_k] + Y\mathbb{E}[D_{t,k} + D_{o,k}] - \Delta_k) \tag{7.3b}$$

$$r_{s,k}^{Quality}[\text{MWh}] = d_{t,k} + d_{o,k} \tag{7.3c}$$

with $\Delta_k = w_k + q - d_{t,k} - d_{o,k}$ representing the surplus of supply on demand, X and Y defined in the following, and $\mathbf{1}$ the indicator function. Reward (7.3a) subtracts purchasing costs to the aggregator revenue to compute the net profit. Indicator $\mathbf{1}^A$ ($\mathbf{1}^B$) corresponds to the scenario when the sum of day-ahead purchased baseline energy and of wind energy is sufficient (insufficient) to cover the demand. In the latter case, fast-start energy needs to be purchased in real-time. Reward (7.3b) computes the Loss of Load (LoL). In practical scenarios, the amount of fast-start energy available in real-time is limited. Often this limit is computed with the formula $FS \leq X\mathbb{E}[W] + Y\mathbb{E}[D_t + D_o]$ (e.g., $X = 3\%, Y = 10\%$) [53]. If $\Delta + FS < 0$ the service incurs in a LoL, with potentially risky consequences. Reward (7.3c) accounts for user demand incentivized by energy pricing.

Finally, we mark all states with $\Delta + FS < 0$ with the label *risk*, and use label *abs* for the absorbing state, so $\Omega = \{risk, abs\}$.

The optimal strategy $\sigma^* = (u^*, Q^*, v_k^*), 1 \leq k \leq K$ is the solution of the constrained optimization problem:

$$\begin{aligned} \mathbb{W}_{s_0}^* = & \max_{Q,u} \min_{\mathbf{f_s^a} \in \mathcal{F}_s'^a} \mathbb{E}_W^{\sigma,\mathbf{f_s^a}} \mathbb{E}_{D_t}^{\sigma} \max_{v_k} \mathbb{E}_{D_o}^{\sigma} rew_{r^{Profit}}(\pi, K) \\ & s.t.\ \mathcal{M}_E', \sigma^* \models_{Nat} \phi \quad \text{where:} \\ & \phi = R_{\leq EENS_M}^{r^{LoL}}[\boldsymbol{\mathcal{C}}\ abs] \wedge R_{\geq QoS_m}^{r^{Quality}}[\boldsymbol{\mathcal{C}}\ abs] \wedge P_{\geq 1-LoLP_M}[\neg risk\ \boldsymbol{\mathcal{U}}\ abs] \end{aligned} \tag{7.4}$$

In Problem (7.4), we maximize the expected value of the aggregator profit $\mathbb{W}_{s_0}^*$ under the worst-case resolution of uncertainty in the wind-energy forecast by summing the instantaneous state rewards r_s^{Profit} along the paths $\pi \in \Pi_{fin}$ of K steps of $\mathcal{M}_E'$, corresponding to the K T_2-slots. By replicating the decision process for each of the K T_2-slots while building model $\mathcal{M}_E'$, every K-step execution path traverses all decision epochs, so the computed reward, as introduced in Definition 2.10, will account for the sum of the contributions of each decision epoch.

Moreover, according to the semantics defined in Table 2.2, the PCTL specification ϕ constrains the expected aggregator risk and user QoS across the decision horizon. $EENS_M$ is the desired maximum value of Expected Energy Not Served, $LoLP_M$ is the maximum allowed value of Loss of

Load Probability (these two properties limit the risk for the aggregator), and QoS_m is the minimum value of QoS that needs to be guaranteed to the users.

Remark 7.2. *While in the current formulation of the problem the cost function maximizes the economic profit of the energy aggregator, any other* social welfare *cost function (as the ones considered by Bouffard et al. [29] and Papavasiliou et al. [133]) can be considered with minimal changes to the optimization problem. Indeed, this can be done by equipping the Ellipsoidal-MDP model of the system with an appropriate reward function to account for the different metric to be optimized.*

7.4 Experimental Results

In this section, we present experimental results about the performance of the energy aggregator when operating following the optimal control strategy generated by solving Problem (7.4). The goals of this section fall into two main categories. First, we aim to characterize the performance of the strategy-synthesis algorithm and give insight about its functionality. Second, we will apply the algorithm more specifically to the energy pricing and purchasing problem and present results on:

- the analysis of the impact of uncertainties in the forecast of the availability of wind energy on the economic trade-offs that drive the decision process of the energy aggregator and;
- the performance of the energy aggregator when operating under different control strategies, to show that the strategy synthesized using the proposed approach can achieve better performances *on average* than other solutions presented in the literature.

The constrained optimization problems used to synthesize optimal control strategies were solved using the convex solver Gurobi [68]. Experimental runtime data were obtained on a 2.4 GHz Intel Xeon machine with 32GB of RAM.

To simplify the syntax in the rest of the section, we define the following quantities:

$Profit$	:=	$\mathbb{W}^{*}_{s_0}$
$EENS$	:=	$R^{r^{LoL},\, \sigma^*,\, max}_{s_0}[\boldsymbol{\mathcal{C}}\ abs]$
QoS	:=	$R^{r^{Quality},\, \sigma^*,\, min}_{s_0}[\boldsymbol{\mathcal{C}}\ abs]$
$1 - LoLP$	:=	$P^{\sigma^*,\, min}_{s_0}[\neg risk\ \boldsymbol{\mathcal{U}}\ abs]$

where the min and max operators refer to the action range of nature Nat. As defined in Section 2.2, these quantities represent the *quantitative* values of rewards and satisfaction probability, which will then be compared to the corresponding thresholds ($EENS_M,\ QoS_m, LoLP_M$) in Problem (7.4) to determine the satisfaction of ϕ, i.e., whether $\mathcal{M}'_E, \sigma^* \models_{Nat} \phi$.

In the following experiments, we will normalize the $Profit$ to the maximum computed profit value for each set of experiments, labeled as $Profit_M$, to ease the interpretation of the results. Moreover, we set $T_1 = 1$h, $T_2 = 30$min so $K = 2$, and consider two pricing options both for traditional (u^a and u^b) and opportunistic users (v^a and v^b). If not otherwise stated, we will use

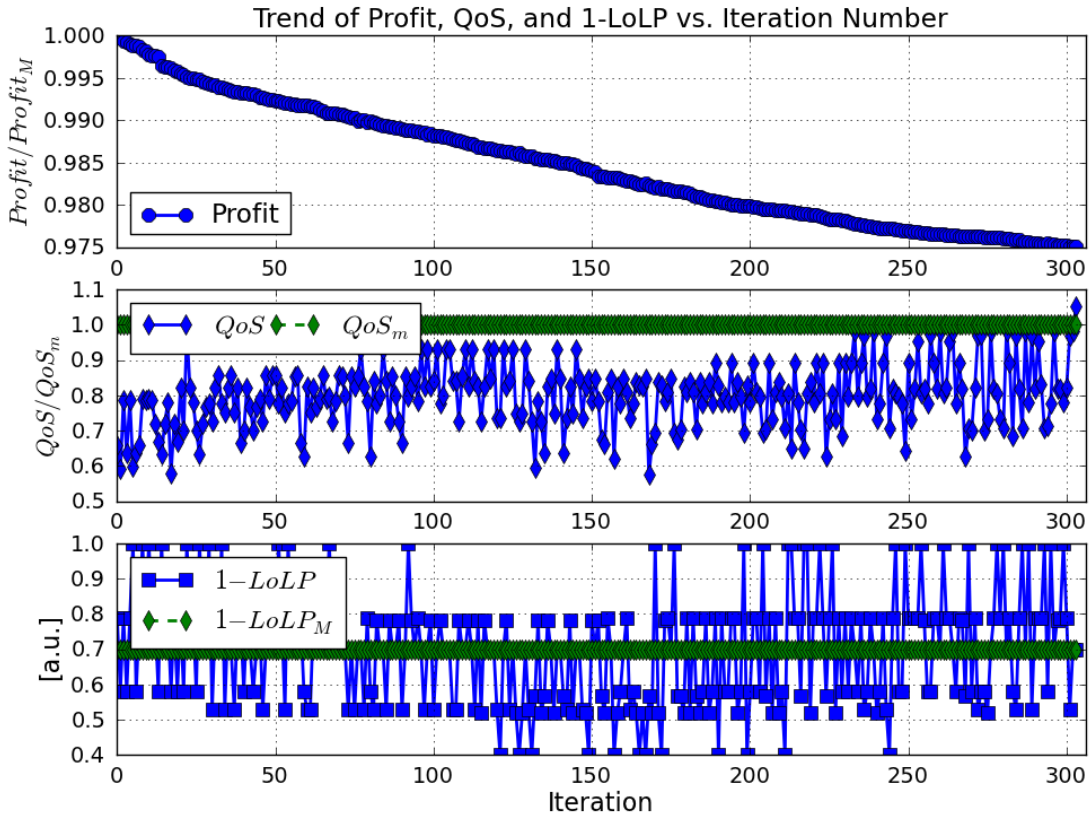


Figure 7.4: The figure shows the trend of the expected performances of the system at the different iterations of the strategy-synthesis algorithm in terms of aggregator profit (top), Quality of Service (QoS) for the users (middle) and Loss of Load Probability (LoLP) (bottom). At each iteration, these performances are evaluated by the verification engine, to assess whether the control strategy explored by the optimization engine satisfies the system specifications.

$C_L = 90\%$. Furthermore, we will discretize the wind energy W in 5 bins, and traditional D_t, opportunistic D_o demands and baseline supply Q in 2 bins. Finally, we set:

$$\begin{aligned} QoS_m &= 80\% \textstyle\sum_k \mathbb{E}[D_{t,k} + D_{o,k}] \\ LoLP_M &= 10\% \\ EENS_M &= 5\% \textstyle\sum_k \mathbb{E}[W_k + q] \end{aligned}$$

The other parameter values were taken from references [66, 76].

In Figure 7.4, we show the trend of the expected system performances as a function of the iteration of the synthesis algorithm. The energy aggregator $Profit$ monotonically decreases until the proposed candidate strategy σ_c meets all specifications. We note that $1 - LoLP$ and QoS ($EENS$, not shown, has a trend similar to $1 - LoLP$) instead vary non-monotonically. Intuitively,

Table 7.2: Performance Analysis

W bins	5	10	15	20
$Profit$	1	0.98	0.97	0.965
$1 - LoLP$	0.99	0.99	0.99	0.99
$EENS$	0.98	0.98	0.98	0.98
QoS	1.01	1.01	1.01	1.01
Runtime	144s	400s	1368s	3289s
#Iter.	223	53	547	332
$N + T$	1343	2719	4115	5591
#MD Strat. (I)	4096	4.2e6	4.3e9	4.4e12

this is because the $Profit$ can be increased in at least two different ways, as described in the following.

- In the first approach, the energy aggregator purchases less baseline energy Q. This strategy thus relies more on wind energy to cover the demand and it reduces the costs associated to the purchasing of non-renewable energy. On the other hand, this approach results in a higher risk of power unbalance because the fast-start energy might not be enough to compensate for the under-dispatch of baseline energy. Iterations associated to such an approach will thus result in a high value of $LoLP$ and $EENS$.

- In a second approach, the energy aggregator increases the energy price v for opportunistic users. This strategy has the effect of forcing more users to decrease their power consumption, but it does not reduce the aggregator profit because each of the remaining users pays a higher price. While advantageous for the energy aggregator, it is apparent that this approach is *unfair* to the users and it thus results in a reduction of the value of QoS.

The optimal strategy that satisfies all specifications will strike a balance between the two approaches just described. By ranking strategies by expected $Profit$, our algorithm is capable of selecting such optimal strategy despite the complex parameter interdependences of the model under analysis.

In Table 7.2, we compare synthesis results while varying the number of discretization bins for W (all values are normalized to the corresponding target specification to ease the comparison). First, we note that the expected system performances do not substantially vary by changing the number of bins, thus supporting our choice of 5 bins in the other experiments. Second, we collect runtime results for problem of increasing size, where we use $N + T$, i.e., the sum of the number of states and of the number of transitions in the Ellipsoidal-MDP, as a proxy of the model size. Results show that the algorithm can handle in reasonable time problems more than 10 times larger than the ones analyzed by Lahijanian et al. [103], who presented the only other algorithm proposed in the literature capable of accepting arbitrary PCTL formulas (further, we remind from Chapter 3 that this algorithm was not complete and it does not accept state-transition uncertainty sets). We believe that this improvement in the scalability of the algorithm is mainly due to the effective

decomposition of the constrained optimization problem into an unconstrained optimization step and into a verification step, both exploiting decades of advances in mixed-integer and non-linear optimization engines [68] and in verification algorithms [26, 99, 139]. Third, we observe that the total number of MD strategies I:

$$I = |\mathcal{A}_{s_0}| \times |\mathcal{A}_{s_1}| \times \cdots \times |\mathcal{A}_{s_N}| = O(M^N)$$

grows indeed exponentially with the number of states to be controlled, as expected. As a consequence, the approach of verifying all the I MD strategies $\sigma \in \Sigma^{MD}$, which is always available in an NP-complete problem and is an alternative to the approach proposed in Chapter 6, is not practical on problems of even medium size. In fact, such an exponential increase of the search space motivates further future work to find techniques to prune more rapidly non-interesting re-

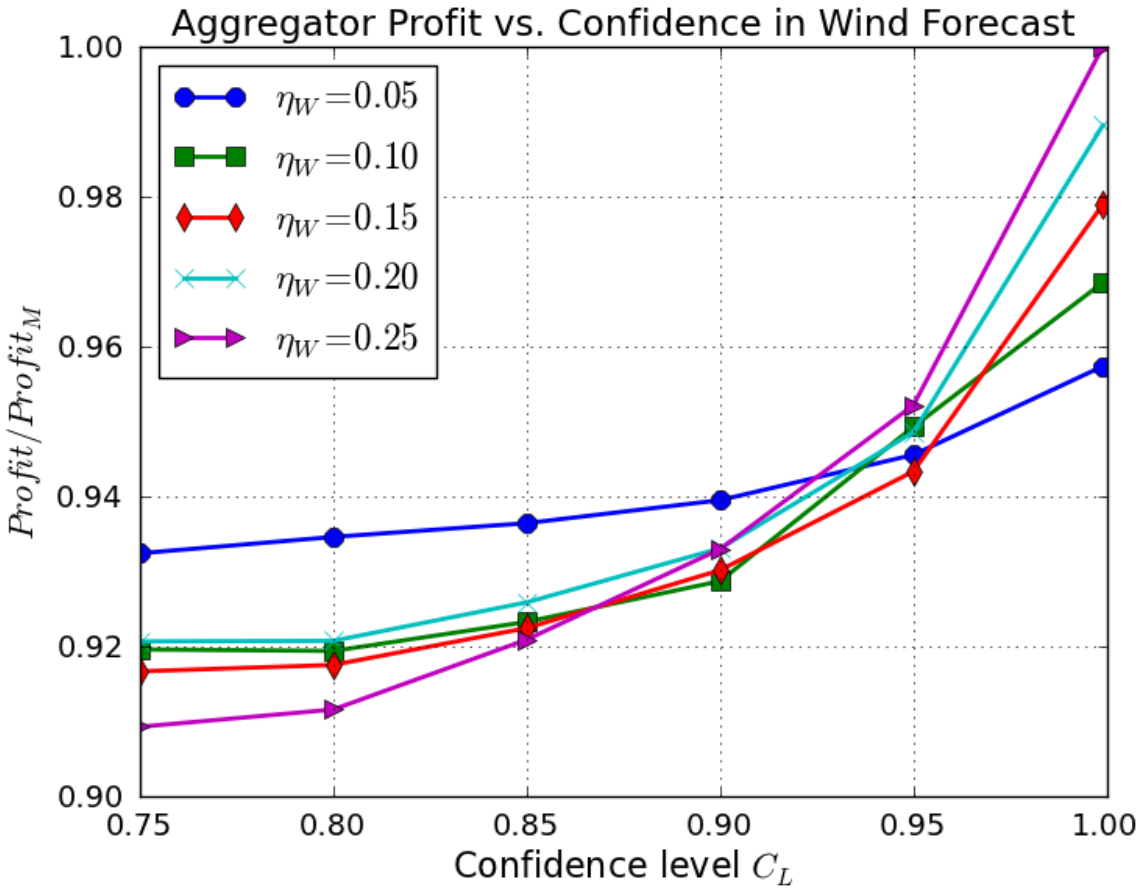


Figure 7.5: The figure shows the trend of the expected economic profit for the energy aggregator as a function of the confidence level C_L in the forecast of wind availability. Each curve is associated to a different value of wind penetration η_W.

gions of the search space. In particular, this can be done by generating shorter *conflict clauses* in the verification engine.

In Figure 7.5, we study the effect of different confidence levels C_L in the wind-energy forecast on the expected $Profit$ of the aggregator, while keeping all constraints constant and only sweeping the value of wind penetration η_W, with:

$$\eta_W = \sum_{k=1}^{K} \frac{\mathbb{E}[W_k]}{\mathbb{E}[W_k + q]}$$

where we do not include at the denominator the expected value of the fast-start energy $\mathbb{E}[f]$, because such energy is used only in emergency, if the forecast of the availability of renewable energy was wrong. For high values of C_L, higher profits can be expected for increasing η_W, since wind energy is assumed free. This analysis, the only one available using strategy-synthesis algorithms that cannot capture modeling uncertainties, would thus suggest the energy aggregator to rely more heavily on wind energy. On the other hand, our approach shows that, for low values of C_L, higher wind penetration amplifies more the effect of uncertainties on the expected performances of the aggregator. In fact, for $C_L \leq 90\%$, the trend of expected profits is even reversed, and low wind penetration guarantees higher profits. This is indeed to be expected, since a more conservative strategy only relying on non-renewable supplies guarantees fewer energy wastes and a lower risk of power unbalance when there is little information about the availability of renewable energy. The energy aggregator can use these curves to better select the mixture of energy supplies to be employed, and to assess the return of investment in employing more accurate (and expensive) forecast techniques to better predict what to expect by using renewable sources.

Finally, in Figure 7.6 we compare results with two other formulations for energy pricing proposed in the literature.

- He et al. [76] solve the optimization problem without enforcing any constraint on the risk of power unbalance and on the QoS for users, i.e., they just optimize the energy aggregator profit.
- Varaiya et al. [167] set limits only on the acceptable LoLP. In fact, their approach is not trivially extendable to reward properties expressed using the $\boldsymbol{R}$ operator. Moreover, they solve optimally (in fact, analytically) only for $LoLP = 0$, i.e., they enforce the risk of power unbalance to be null.

Comparison is done by solving the three different strategy-synthesis formulations and then running Monte Carlo simulations (1000 runs) of the controlled system on test data (different from the training ones) to evaluate its performance ($Profit$, $EENS$ and QoS). This can be done by executing the induced Convex-MC for K steps, collecting the observed system performances for each run, and then taking the mean value of each performance across runs. We repeat such process while varying the wind penetration level η_W, to assess the impact of renewable sources on the synthesized strategies. The values of EENS and QoS are normalized to the thresholds of the corresponding specification, so, ideally, the reported values should all be equal to 1.

As expected, the unconstrained strategy by He et al. [76] has higher $Profit$ (up to 5%) compared to our approach, but it also has up to 12% more $EENS$ and 10% less QoS, since these performances are not constrained at optimization time. The strategy by Varaiya et al. [167] guarantees null $EENS$, but it has up to 6% lower $Profit$ (due to over-constraining $EENS$) and 10% less QoS (which is left unconstrained). Moreover, we note that system performance tend to be similar among all approaches for low levels of wind penetration, while discrepancies increase when the aggregator portfolio relies more heavily on renewables. This is indeed expected, since all approaches produce similar results when strategies rely mainly on the deterministic availability of non-renewable energy. On the other hand, when the stochastic behavior of the system is enhanced by relying more on wind energy, the importance of constraining system performance at optimiza-

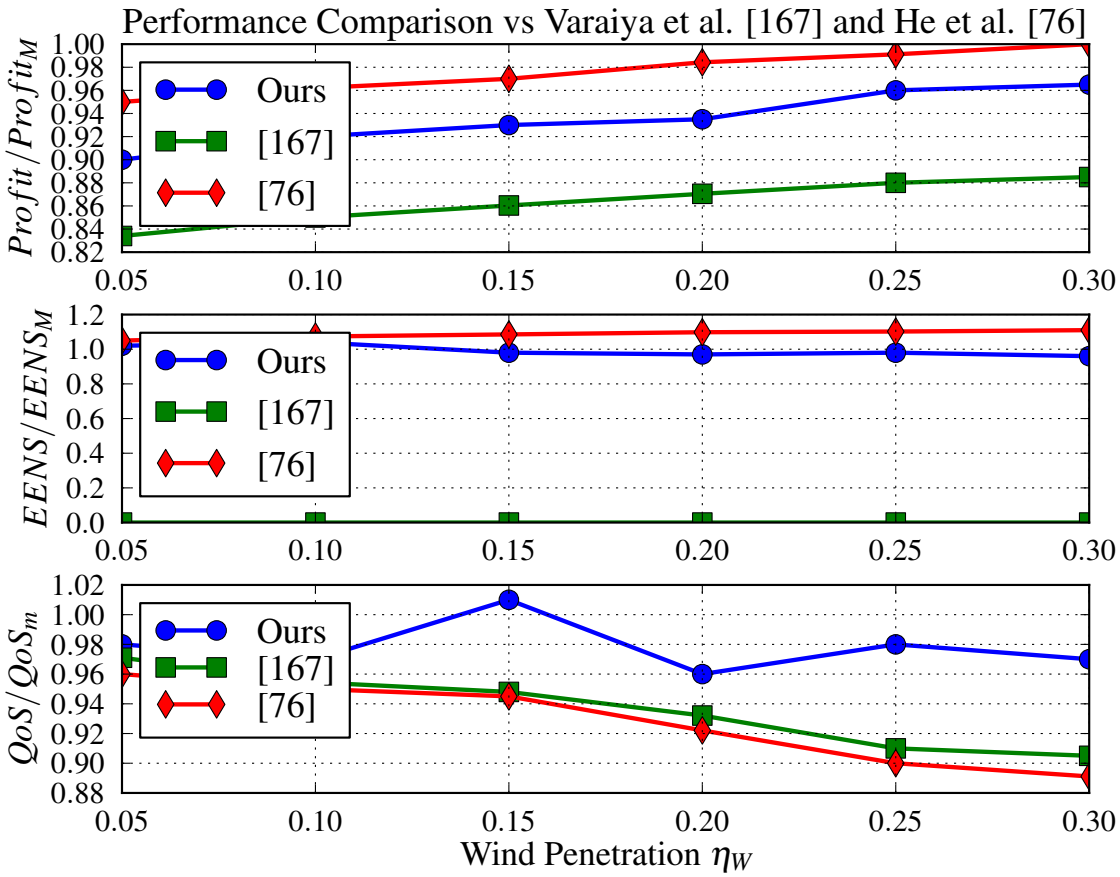


Figure 7.6: The figure shows a comparison via Monte Carlo simulation of the performances of the energy aggregator when operating following strategies synthesized using the proposed approach or using two alternative approaches presented in the literature. We show the energy aggregator profit at the top, the Expected Energy Not-Served (EENS) in the middle, and the Quality of Service (QoS) for the users at the bottom.

tion time becomes more apparent. Results show that the aggregator performances obtained using the proposed approach remain quite close to the target specification thresholds across all values of wind penetration.

Remark 7.3. *We note that the algorithmic runtime may increase exponentially as we tighten the specification thresholds ($QoS_m,\ LoLP_M,\ EENS_M$), since it becomes increasingly more difficult to find a solution within the exponentially-sized search space. Nevertheless, the chosen values for these thresholds were tight enough to improve the quality of alternative energy pricing strategies proposed in the literature, while maintaining the runtime acceptable for this application.*

Chapter 8

Conclusions and Future Work

In this chapter, we summarize the main contributions presented in the dissertation and highlight new promising directions of research to be pursed as future work.

8.1 Conclusions

The complexity of today's embedded systems has reached unprecedented levels. Formal techniques to assist the design and verification of these systems are needed to shorten design iterations and guarantee correct functionality and satisfactory performance early in the development cycle. These formal methods often rely on deriving a probabilistic model of the underlying system, hence the formal guarantees they provide are only as good as the estimation of the process dynamics. In a real setting, these estimations are affected by *uncertainties* due for example to unmodeled dynamics, measurement errors or approximations of the real system by mathematical models. It is thus fundamental to mathematically capture the uncertainties in the modeling process in order to produce sensible analysis results. Moreover, fast algorithmic runtime is a necessary attribute for these design and verification techniques, so that they can successfully process systems of large size, as required by today's real-world applications.

In this dissertation, we have presented a comprehensive framework to allow the modeling, verification and control of stochastic systems in the presence of uncertainties in the modeling process. In particular, we presented time efficient algorithms to model check and optimally control these systems, and we applied these techniques to two real-world case studies, to show the feasibility of our approach.

We started by introducing the model of Convex-Markov Decision Processes (Convex-MDPs), i.e., MDPs whose state transition probabilities are only known to lie within convex uncertainty sets. This formalism is able to model non-determinism and stochasticity, as regular MDPs, and it is furthermore capable of mathematically capturing uncertainties in the estimation of the state-transition

probabilities of the model. As such, Convex-MDPs represent a highly expressive formalism to model the behavior of the wide class of systems whose dynamics are not fully known or specified. Noticeably any system containing a physical component belongs to this category, so Convex-MDPs have the potential of becoming a premiere framework to model the behavior of embedded systems.

We then addressed the problem of model checking properties of Convex-MDPs expressed in Probabilistic Computation Tree Logic (PCTL). PCTL allows querying a wide variety of *quantitative* properties of a system, e.g., *"What is the maximum probability that the system will eventually send an acknowledgment if it receives a request?"*. Such properties are fundamental for the analysis of systems that cannot be proven error-free under all circumstances but whose behavior is still acceptable as long as the probability of failure is sufficiently low.

Using results on strong duality for convex programs, we proved that the PCTL model-checking problem for Convex-MDPs is decidable in P for the fragment of PCTL without operators with a finite time horizon. For the entire PCTL syntax, the algorithmic complexity just increases to pseudo-polynomial in the maximum value of time horizon. This result allowed us to lower the previously known complexity upper bound for the problem of PCTL model checking for Convex-MDPs from co-NP to P, and it is valid for a wide class of non-linear convex uncertainty models, including the ellipsoidal, likelihood and entropy models, which are commonly used to capture the stochastic behavior of cyber-physical systems.

Furthermore, the new result on theoretical complexity allowed us to develop the first polynomial-time algorithm for the PCTL model checking of Convex-MDPs. An experimental analysis of the algorithmic runtime showed that the proposed algorithm does indeed scale to the analysis of large systems and it is faster than the state-of-the-art model checker PRISM for some problem instances.

We used the developed algorithm to analyze the behavior of human drivers while performing complex maneuvers in a variety of environmental conditions, e.g., with or without an obstacle on the road, and distracted or not while driving. Such analysis is useful in several applications, including personalized correction of driving misbehaviors, personalized computation of car insurance rate, and, ultimately, real-time automated driving assistance in semi-autonomous cars. We trained a Convex Markov Chain (Convex-MC) model of the performance of a driver with data collected using a car simulator. We then used the model-checking algorithm to estimate the driver behavior while performing a complex maneuver, e.g., how likely he or she would successfully complete a double turn while being distracted by a text message. Results show that personalized assessment of the driving performance is indeed possible. As an example, it is possible to automatically determine if a driver tends to break too often, and then intervene to correct this misbehavior.

We finally addressed the problem of synthesizing control strategies for Convex-MDPs, with the goal of maximizing a given system performance while guaranteeing that the system execution satisfies a specification expressed in PCTL for all resolutions of uncertainty in the state-transition probabilities of the model. In other words, we analyzed techniques to solve a *game* between a controller, which aims to maximize the system performance, and the modeling uncertainties, which are interpreted as adversarial and which try to cause the system to fail.

We first proved that the strategy-synthesis problem is in the complexity class NP-complete also when convex uncertainty sets are added to the model. This result shows that adding uncertainties does not make the problem more complex than the version without uncertainties, which is also NP-complete.

We then cast the synthesis problem into a constrained optimization problem, where the constraint is represented by the PCTL specification and the system performance has to be optimized. To solve the problem, we presented the first sound and complete synthesis algorithm capable of processing specifications expressed using the full PCTL syntax. The algorithm avoids a blind strategy enumeration by ranking the available strategies in order of optimality, i.e., it iteratively determines the strategy that maximizes the system performance among those that have not been explored yet, and checks whether such a strategy satisfies the PCTL specification. The first strategy that satisfies the specification is thus guaranteed to be also the solution of the constrained optimization problem.

We applied the synthesis algorithm to the problem of determining optimal energy pricing and purchasing strategies to be adopted by a for-profit energy aggregator whose portfolio of energy supplies includes renewable sources, e.g., wind. The synthesis problem aims to maximize the economic profit of the energy aggregator while minimizing the risk of power unbalance within the portfolio and guaranteeing the desired Quality-of-Service (QoS) level for the service subscribers. Uncertainties in the model creation stem from the difficulties in predicting the availability of the renewable sources of energy, and were statistically characterized using measured data from a windmill farm. Results showed that the energy aggregator risks and user QoS can be both effectively constrained at design time, and that more accurate predictions of the expected economic profit can be obtained with respect to state of the art solutions, by taking the uncertainty in the availability of the renewable sources into consideration.

8.2 Future Work

We believe that the results presented in this dissertation have opened up the path to numerous further research directions. These span the theoretical, algorithmic and application aspects of the presented work.

From the theoretical side, it would be interesting to extend the proposed results to consider also different frameworks to model stochastic systems and to analyze other formal logics to express system properties.

While Markov Decision Processes (MDPs) have been widely used in the literature in a variety of applications, system designers would benefit from a wider availability of modeling options. In general, the results discussed in this dissertation best apply to the modeling of systems that integrate a computation (cyber) component (e.g., a controller) and a physical component (e.g., a plant). The extension of such results to the analysis of Continuous-Time Markov Chains (CTMCs) [159], Probabilistic Timed Automata (PTA) [149] and Hybrid Automata (HA) [78], all widely used formalisms for the analysis of cyber-physical systems, would thus represent a promising next step of research. In fact, we have shown in Chapter 4 that the proposed approach can already be applied

to the analysis of PTA (and CTMCs) by appropriately converting these models into MDPs through a discretization of the time evolution of the system dynamics. On the other hand, specialized techniques for the analysis of these systems might be able to achieve higher accuracy in the results and shorter runtime, so they might be worth exploring.

We have shown in Chapter 2 that Probabilistic Computation Tree Logic (PCTL) can express a wide variety of *quantitative* safety and liveness properties of a stochastic system. On the other hand, it cannot express arbitrary liveness properties and enforce arbitrary fairness conditions. It would thus be interesting to extend the proposed techniques to other more expressive formal logics, like Probabilistic Branching Time Logic (PBTL) [19] and ω-PCTL [38]. These logics have already been studied in the context of the analysis of models with no uncertainties, but more work needs to be developed to improve the theoretical complexity of the model-checking and strategy-synthesis problems in the presence of modeling uncertainties.

Overall, the suggested strategy to be pursued to extend the proposed results should start by formulating the model-checking or the strategy-synthesis problems on models with no uncertainties in terms of a linear program (LP) or mixed-integer linear program (MILP). It would then be possible to apply the dual transformations exemplified in Chapter 4 (for model checking) and Chapter 6 (for strategy synthesis) to include uncertainties. In this way, adding uncertainties would not increase the complexity of the model-checking and control problems with respect to the analogous problems for models without uncertainties. For example, we conjecture that properties of Convex-MDPs expressed with the full Probabilistic Linear Temporal Logic (PLTL) can be model checked in time polynomial in the size of the model and double-exponential in the size of the formula by starting from the LP formulation of the analogous problem for MDPs presented by Baier et al. [18].

New directions for research are present also from the algorithmic side.

First, conditions to relax the *rectangular uncertainty* assumption, i.e., the requirement for the state-transition probability distributions to be independent from one another among different states, can be investigated. Indeed, such an assumption is not fundamental to derive the convex programming formulations of the model-checking and strategy-synthesis algorithms, and it was made to ease the exposition of the material. In fact, as a less stringent condition on the functional relation between state-transition probability distributions of different states, we can simply require that such a functional relation maintains the convexity of the final mathematical programming formulation of the verification and control problems. Although we conjecture that this condition cannot be satisfied for arbitrary non-linear functional relations among state-transition probability distributions, it would be worth exploring scenarios in which a specific functional relation (e.g., a linear combination) has a concrete *physical* meaning for the application under analysis, and re-derive the convex programming formulations in such a scenario. The main advantage of such a construction would be to decrease the action range of the adversarial nature on the system, thus increasing the modeling expressivity and resulting in a less conservative and more realistic analysis of the system behavior.

Specifically related to the strategy-synthesis algorithm presented in Chapter 6, it would be important to investigate techniques to generate more concise constraints to prove the failure of the verification step of the synthesis loop, in order to prune more effectively the search space for the

optimization engine. The approach proposed in this dissertation is capable of discarding only one candidate strategy per iteration. On the other hand, it is reasonable to expect that in some scenarios a few bad choices of actions in key states of the Convex-MDP might be already enough to prevent the satisfaction of the PCTL specification, no matter what is the action assignment for the other states in the model. Detecting such a core reason of failure in meeting the specifications would allow to prune more effectively the search space, with exponential speed-ups in the algorithmic runtime. In fact, techniques to generate compact reasons of failure of conjunctions of mixed-integer convex clauses have already been proposed in the literature [71, 126]. More work still needs to be done to adapt such techniques to the specific structure of the strategy-synthesis problem.

The quest towards achieving shorter and shorter algorithmic running time is never ending. In fact, an exciting new realm of *real-time* applications for the verification and control algorithms could be explored, if it was possible to solve the optimization problems in a matter of milliseconds or less. While such a runtime can hardly be achieved with a software implementation of the convex programming solver, a *hardware* implementation of the solver can achieve substantial speed-ups thanks to its inherent parallelization. Analog circuits to solve linear programs have indeed already been demonstrated [162], and are capable of solving linear programs with a fixed structure in a matter of microseconds. Although a new circuit implementation would be required for any new *instance* of the optimization problem, in many applications, noticeably the real-time control of physical systems, the same problem instance needs to be solved over and over again during the system execution while only varying the problem *parameters*, to determine at each time step the control signals to be actuated. The migration of the convex problems presented in this dissertation to a hardware implementation thus appears as a promising solution to extend the applicability of the proposed verification and optimal control approaches.

Finally, and most importantly, the relevance of the presented work will be tested by applying the proposed framework to the analysis and design of complex real-world applications. In this dissertation, we have already analyzed the performance of a human driver, and synthesized optimal energy pricing and purchasing strategies to be adopted by an energy aggregator whose portfolio includes renewables, but plenty further fields could potentially benefit from the presented results. In the following, we list a few of them.

The topic of improving the energy efficiency of commercial and residential buildings, which account for a large percentage of the total primary energy consumption in all developed and developing countries, has received a lot of attention in the last decade. In fact, this task is considered of outmost importance in the development agenda of several governments of Western countries [87]. The roadmap to enable the realization of these *green* buildings foresees the deployment of sensor and actuator networks within the buildings. These networks are capable of sensing internal (e.g., the presence of people) and external (e.g., the weather conditions) changes and promptly react (e.g., by changing the HVAC settings), to optimize energy efficiency without reducing the perceived comfort. A number of challenges need to be faced in order to guarantee the correct functionality of these solutions, spanning the whole stack of the OSI layers [84, 110, 116, 140]. The analysis and control framework proposed in this dissertation can be applied to assess and control the performance of these networks by capturing the behavior of the digital controllers, of the

analog sensors and actuators and of the uncertain physical environment of the building.

The topic of energy management has been widely investigated also at the level of a single integrated circuit. In fact, nowadays Systems-on-Chip (SoCs) integrate an unprecedented amount of functionality and are ubiquitously present around us, enabling the new computation revolution. Given the explosion of both the number of mobile electronic devices, and of the number of servers to support the computational *"cloud"*, it is fundamental to maximize the energy efficiency of electronic systems both to increase their battery life and to reduce operation costs. Energy consumption can be reduced in SoCs by partitioning the integrated circuit in multiple "energy-domains" and by providing to each domain only the energy that is strictly necessary to maintain correct functionality, a technique called Dynamic Voltage and Frequency Scaling (DVFS). While simple in principle, such an approach presents enormous challenges to be fully exploited and it is an active topic of research in the community [47, 88, 104, 135]. The modeling framework presented in this dissertation can be used to verify and control the functionality of SoCs adopting DVFS. The model state can capture snapshots of the data flow within the SoC during execution. The actions available at each state can represent the operating voltages and frequencies for each energy domain in the system, and a reward structure can capture the total consumed energy associated to choosing a given action. The strategy-synthesis algorithm can then be used to find the DVFS strategy that minimizes energy consumption, while the model-checking algorithm can be applied to verify the correct transmission of data packets across different energy domains.

The verification of the model of the performance of a human driver presented in Chapter 5 represents just the first step towards the more ambitious goal of designing and verifying the control system for autonomous and semi-autonomous cars. Autonomous and semi-autonomous car driving is recently receiving very high attention in both the academic and industrial research communities [111]. In fact, several car companies, including Volvo, Mercedes, BMW, GM, Toyota, are already equipping their high-tier cars with technology capable of automatically performing simple tasks like *distance keeping* or *lane changing*. Moreover, Google has recently started the first on-the-road experimentations of the Google Self-Driving Car [117], a fully autonomous car capable of driving itself to the desired destination, and four states in the United States (Nevada, Michigan, Florida and California) have already authorized testing of autonomous vehicles on public roads. The challenges presented by this project are unprecedented and they will involve solutions from a heterogeneous set of disciplines, ranging from mechanical engineering and control theory to embedded software development [57, 109]. Abstraction techniques to represent the complex model dynamics will play a key role in this effort, and the capability of formally capturing the uncertainties in the modeling process will be fundamental, given the number of undefined environmental conditions at design time (e.g., the surroundings of the car while moving in a high-traffic city road). The application of the techniques developed in this dissertation to autonomous cars thus offer several opportunities. For example, the strategy-synthesis algorithm could be run in real-time in a receding-horizon fashion to optimally control the navigation of the autonomous vehicle while robustly accounting for uncertainties in the prediction of the future environmental dynamics. Furthermore, the model-checking algorithm could be run on the outputs of the other controllers composing the system to guarantee that the composition of the different control strategies successfully achieves the desired goals (e.g., reaching the destination while maintaining the passengers

safe).

In conclusion, the fields of verification and optimal control of stochastic systems have attracted a lot of attention in the last decade, but we believe that they have not reached yet full maturity. Many opportunities exist for further research and it appears that this is the perfect time to work on these topics and produce results that can indeed shape the research directions of the community in the years to come.

Bibliography

[1] Alberto Puggelli's home page on the website of the EECS Department at the University of California, Berkeley. Online: `http://www.eecs.berkeley.edu/~puggelli/`.

[2] The PRISM Model Checker. Online: `http://www.prismmodelchecker.org/`.

[3] J. Abate and W. Whitt. "Transient Behavior of the M/M/l Queue: Starting at the Origin". In: *Queueing Systems* 2.1 (1987), pp. 41–65.

[4] N. Ahmed, E. de Visser, T. Shaw, R. Parasuraman, A. Mohammed-Ameen, and M. Campbell. "A Look at Probabilistic Gaussian Process, Bayes Net, and Classifier Models for Prediction and Verification of Human Supervisory Performance". In: *Formal Verification and Modeling in Human-Machine Systems*. AAAI Spring Symposium Series. Mar. 2014.

[5] L. de Alfaro. "Formal Verification of Probabilistic Systems". Ph.D. Dissertation. Stanford University, 1997.

[6] L. de Alfaro and T. A. Henzinger. "Concurrent omega-Regular Games". In: *Proceedings. of the 15th Annual IEEE Symposium on Logic in Computer Science (LICS)*. June 2000, pp. 141–154.

[7] L. de Alfaro and R. Majumdar. "Quantitative Solution of omega-Regular Games". In: *Journal of Computer and System Sciences* 68.2 (2004), pp. 374–397.

[8] D. F. Anderson and T. G. Kurtz. "Continuous Time Markov Chain Models for Chemical Reaction Networks". In: *Design and Analysis of Biomolecular Circuits*. Ed. by H. Koeppl, G. Setti, M. di Bernardo, and D. Densmore. Springer New York, 2011, pp. 3–42.

[9] J. R. Anderson. "ACT: A Simple Theory of Complex Cognition". In: *American Psychologist,* 51.4 (Apr. 1996), pp. 355–365.

[10] S. J. Anderson, S. C. Peters, T. E. Pilutti, and K. Iagnemma. "Design and Development of an Optimal-Control-Based Framework for Trajectory Planning, Threat Assessment, and Semi-Autonomous Control of Passenger Vehicles in Hazard Avoidance Scenarios". In: *Robotics Research*. Ed. by C. Pradalier, R. Siegwart, and G. Hirzinger. Vol. 70. Springer Tracts in Advanced Robotics. Springer Berlin Heidelberg, 2011, pp. 39–54.

[11] A. Andreychenko, L. Mikeev, D. Spieler, and V. Wolf. "Parameter Identification for Markov Models of Biochemical Reactions". In: *Computer Aided Verification (CAV)*. Ed. by G. Gopalakrishnan and S. Qadeer. Vol. 6806. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2011, pp. 83–98.

[12] G. Aoude, B. Luders, K. Lee, D. Levine, and J. How. "Threat Assessment Design for Driver Assistance System at Intersections". In: *Proceedings of the 13th International IEEE Conference on Intelligent Transportation Systems (ITSC)*. 2010, pp. 1855–1862.

[13] J. Aspnes and M. Herlihy. "Fast Randomized Consensus Using Shared Memory". In: *Journal of Algorithms* 11.3 (1990), pp. 441–461.

[14] A. Aziz, K. Sanwal, V. Singhal, and B. Brayton. "Verifying Continuous Time Markov Chains". In: *Computer Aided Verification (CAV)*. Springer-Verlag, 1996, pp. 269–276.

[15] C. Baier. "On Algorithmic Verification Methods for Probabilistic Systems". PhD thesis. Universität Mannheim, Mannheim, Germany, 1998.

[16] C. Baier, M. Größer, M. Leucker, B. Bollig, and F. Ciesinski. "Controller Synthesis for Probabilistic Systems". In: *Exploring New Frontiers of Theoretical Informatics*. Vol. 155. Springer US, 2004, pp. 493–506.

[17] C. Baier, B. Haverkort, H. Hermanns, and J.-P. Katoen. "Model Checking Continuous-Time Markov Chains by Transient Analysis". In: *Computer Aided Verification (CAV)*. Ed. by E. A. Emerson and A. P. Sistla. Vol. 1855. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2000, pp. 358–372.

[18] C. Baier and J. -P. Katoen. *Principles of Model Checking (Representation and Mind Series)*. The MIT Press, 2008.

[19] C. Baier and M. Kwiatkowska. "Model Checking for a Probabilistic Branching Time Logic with Fairness". In: *Distributed Computing* 11 (1998), pp. 125–155.

[20] D. Basacik and A. Stevens. "Scoping Study of Driver Distraction". In: *Transport Research Laboratory. Road Safety Research Report* 95 (2008).

[21] R. E. Bellman. *Dynamic Programming*. Princeton, NJ: Princeton University Press, 1957.

[22] M. Benedikt, R. Lenhardt, and J. Worrell. "LTL Model Checking of Interval Markov Chains". In: *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*. Ed. by N. Piterman and S. A. Smolka. Vol. 7795. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2013, pp. 32–46.

[23] A. Ben-Tal and A. Nemirovski. "Robust Solutions of Uncertain Linear Programs". In: *Operations Research Letters* 25.1 (1999), pp. 1–13.

[24] D. Bertsekas. *Dynamic Programming and Optimal Control*. Athena Scientific, 2011.

[25] D. Bertsimas, E. Litvinov, X. A Sun, J. Zhao, and T. Zheng. "Adaptive Robust Optimization for the Security Constrained Unit Commitment Problem". In: *IEEE Transactions on Power Systems* 28.1 (Feb. 2013), pp. 52–63.

[26] A. Bianco and L. Alfaro. "Model Checking of Probabilistic and Nondeterministic Systems". In: *Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*. Ed. by P. S. Thiagarajan. Vol. 1026. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 1995, pp. 499–513.

[27] M. L. Bolton, E. J. Bass, and R. I. Siminiceanu. "Using Formal Verification to Evaluate Human-Automation Interaction: A Review". In: *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 43.3 (May 2013), pp. 488–503.

[28] S. Borenstein and S. Holland. "On the Efficiency of Competitive Electricity Markets with Time-Invariant Retail Prices". In: *RAND Journal of Economics* 36.3 (Autumn 2005), pp. 469–493.

[29] F. Bouffard and F. D. Galiana. "Stochastic Security for Operations Planning with Significant Wind Power Generation". In: *IEEE Transaction on Power Systems*. Vol. 23. 2. May 2008, pp. 306–316.

[30] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge, MA: Cambridge University Press, 2004.

[31] T. Brazdil, V. Brozek, V. Forejt, and A. Kučera. "Stochastic Games with Branching-Time Winning Objectives". In: *21st Annual IEEE Symposium on Logic in Computer Science (LICS)*. Aug. 2006, pp. 349–358.

[32] K. Brokish and J. Kirtley. "Pitfalls of Modeling Wind Power Using Markov Chains". In: *IEEE/PES Power Systems Conference and Exposition (PSCE)*. Mar. 2009, pp. 1–6.

[33] C. Cacciabue. *Modelling Driver Behaviour in Automotive Environments*. Critical Issues in Driver Interactions with Intelligent Transport Systems. Springer, 2007.

[34] P. Carpentier, G. Cohen, J.-C. Culioli, and A. Renaud. "Stochastic Optimization of Unit Commitment: A New Decomposition Framework". In: *IEEE Transactions on Power Systems* 11.2 (May 1996), pp. 1067–1073.

[35] *CarSim Mechanical Simulation.* `http://www.carsim.com`.

[36] K. Chatterjee and T. A. Henzinger. "Simple Stochastic Parity Games". In: *Proceedings of the 17th Workshop on Logic in Computer Science (LICS)*. Vol. 2803. LNCS. Springer, 2003, pp. 100–113.

[37] K. Chatterjee, M. Jurdziński, and T. A. Henzinger. "Quantitative Stochastic Parity Games". In: *Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. Society for Industrial and Applied Mathematics, 2004, pp. 121–130.

[38] K. Chatterjee, K. Sen, and T. A. Henzinger. "Model-Checking ω-Regular Properties of Interval Markov Chains". In: *Foundations of Software Science and Computational Structures (FOSSACS)*. Ed. by R. Amadio. Vol. 4962. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2008, pp. 302–317.

[39] S. Cheshire, B. Adoba, and E. Gutterman. "Dynamic Configuration of IPv4 Link Local Addresses". Available from `http://www.ietf.org/rfc/rfc3927.txt`.

[40] E. M. Clarke and E. A. Emerson. "Design and Synthesis of Synchronization Skeletons Using Branching Time Temporal Logic". In: *Logics of Programs*. Ed. by D. Kozen. Vol. 131. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 1982, pp. 52–71.

[41] E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.

[42] E. Coelingh, L. Jakobsson, H. Lind, and M. Lindman. "Collision Warning with Auto Brake: a Real-Life Safety Perspective". In: *Proceedings of the 20th International Technical Conference on the Enhanced Safety of Vehicles (ESV)*. June 2007.

[43] A. Condon. "On Algorithms for Simple Stochastic Games". In: *Advances in Computational Complexity Theory, volume 13 of DIMACS Series in Discrete Mathematics and Theoretical Computer Science*. American Mathematical Society, 1993, pp. 51–73.

[44] A. Condon. "The Complexity of Stochastic Games". In: *Information and Computation* 96 (1992), pp. 203–224.

[45] E. M. Constantinescu, V. M. Zavala, M. Rocklin, S. Lee, and M. Anitescu. "A Computational Framework for Uncertainty Quantification and Stochastic Optimization in Unit Commitment with Wind Power Generation". In: *IEEE Transactions on Power Systems* 26.1 (Feb. 2011), pp. 431–441.

[46] C. Courcoubetis and M. Yannakakis. "The Complexity of Probabilistic Verification". In: *Journal of ACM* 42.4 (July 1995), pp. 857–907.

[47] J. Crossley, A. Puggelli, H. -P. Le, B. Yang, R. Nancollas, K. Jung, L. Kong, N. Narevsky, Y. Lu, N. Sutardja, E. J. An, A. L. Sangiovanni-Vincentelli, and E. Alon. "BAG: A Designer-Oriented Integrated Framework for the Development of AMS Circuit Generators". In: *2013 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. Nov. 2013, pp. 74–81.

[48] A. D'Innocenzo, A. Abate, and J.-P. Katoen. "Robust PCTL Model Checking". In: *Proceedings of the 15th ACM International Conference on Hybrid Systems: Computation and Control (HSCC)*. Beijing, China: ACM, 2012, pp. 275–286.

[49] B. Donmez, C. Nehme, and M.L. Cummings. "Modeling Workload Impact in Multiple Unmanned Vehicle Supervisory Control". In: *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans* 40.6 (Nov. 2010), pp. 1180–1190.

[50] K. Draeger, V. Forejt, M. Kwiatkowska, D. Parker, and M. Ujma. "Permissive Controller Synthesis for Probabilistic Systems". In: *Proc. 20th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*. Vol. 8413. LNCS. Springer, Mar. 2014, pp. 531–546.

[51] M. Duflot, L. Fribourg, and C. Picaronny. "Randomized Dining Philosophers without Fairness Assumption". In: *Distributed Computing* 17.1 (2004), pp. 65–76.

[52] J. Dupacova, N. Growe-Kuska, and W. Römisch. "Scenario Reduction in Stochastic Programming: An Approach Using Probability Metrics". In: *Mathematical Programming* 95 (2003), pp. 493–511.

[53] E. Ela. *Operating Reserves and Variable Generation*. NREL Technical Report. [Online]: `http://www.nrel.gov/docs/fy11osti/51978.pdf`. Aug. 2011.

[54] E. A. Emerson and J. Y. Halpern. ""Sometimes" and "Not Never" Revisited: On Branching Versus Linear Time Temporal Logic". In: *Journal of the ACM* 33.1 (Jan. 1986), pp. 151–178.

[55] *ENERNOC*. `http://www.enernoc.com/`.

[56] G. E. Fainekos, S. G. Loizou, and G. J. Pappas. "Translating Temporal Logic to Controller Specifications". In: *Proceedings of the 45th Conference on Decision and Control (CDC)*. Dec. 2006, pp. 899–904.

[57] P. Falcone, F. Borrelli, J. Asgari, H.E. Tseng, and D. Hrovat. "Predictive Active Steering Control for Autonomous Vehicle Systems". In: *IEEE Transactions on Control Systems Technology* 15.3 (May 2007), pp. 566–580.

[58] *Federal Energy Regulatory Commission (FERC)*. `http://www.ferc.gov/`.

[59] J. Filar and K. Vrieze. *Competitive Markov Decision Processes*. New York, NY, USA: Springer-Verlag New York, Inc., 1996.

[60] V. Forejt, M. Kwiatkowska, G. Norman, and D. Parker. "Automated Verification Techniques for Probabilistic Systems". In: *Formal Methods for Eternal Networked Software Systems (SFM)*. Ed. by M. Bernardo and V. Issarny. Vol. 6659. LNCS. Springer, June 2011, pp. 53–113.

[61] R. Fuller and E. Farrell. "Operation Life-Saver Assessment." In: *Project OLA* RS 459 (2001).

[62] C. E. García, D. M. Prett, and Morari M. "Model Predictive Control: Theory and Practice - A Survey". In: *Automatica* 25.3 (1989), pp. 335–348.

[63] J. Gill. "An Entropy Measure of Uncertainty in Vote Choice". In: *Electoral Studies* 24 (2005), pp. 371–392.

[64] *GoodEnergy*. `http://www.goodenergy.com/`.

[65] A. Gore. *An Inconvenient Truth*. New York: Rodale, 2006.

[66] C. Grigg, P. Wong, P. Albrecht, R. Allan, M. Bhavaraju, R. Billinton, Q. Chen, C. Fong, S. Haddad, S. Kuruganty, W. Li, R. Mukerji, D. Patton, N. Rau, D. Reppen, A Schneider, M. Shahidehpour, and C. Singh. "The IEEE Reliability Test System-1996. A Report Prepared by the Reliability Test System Task Force of the Application of Probability Methods Subcommittee". In: *IEEE Transactions on Power Systems* 14.3 (Aug. 1999), pp. 1010–1020.

[67] N. Growe-Kuska, K. C. Kiwiel, M. P. Nowak, W. Römisch, and I. Wegner. "Power Management in a Hydro-Thermal System Under Uncertainty by Lagrangian Relaxation". In: *Volumes in Mathematics and Its Applications*. Vol. 128. IMA. New York: Springer-Verlag, 2002, pp. 39–70.

[68] *Gurobi Optimizer*. [Online]: `http://www.gurobi.com/`.

[69] E. M. Hahn, T. Han, and L. Zhang. "Synthesis for PCTL in Parametric Markov Decision Processes". In: *NASA Formal Methods*. Ed. by M. Bobaru, K. Havelund, G. J. Holzmann, and R. Joshi. Vol. 6617. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2011, pp. 146–161.

[70] E. Hahn, H. Hermanns, B. Wachter, and L. Zhang. "INFAMY: An Infinite-State Markov Model Checker". In: *Computer Aided Verification (CAV)*. Ed. by A. Bouajjani and O. Maler. Vol. 5643. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2009, pp. 641–647.

[71] C. Hang, P. Manolios, and V. Papavasileiou. "Synthesizing Cyber-Physical Architectural Models with Real-Time Constraints". In: *Computer Aided Verification (CAV)*. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 441–456.

[72] H. Hansson and B. Jonsson. "A Logic for Reasoning About Time and Reliability". In: *Formal Aspects of Computing* 6.5 (1994), pp. 512–535.

[73] J. A. Hartigan and M. A. Wong. "Algorithm AS 136: A K-Means Clustering Algorithm". In: *Journal of the Royal Statistical Society.* 28.1 (1979), pp. 100–108.

[74] A. Hartmanns and H. Hermanns. "A Modest Approach to Checking Probabilistic Timed Automata". In: *Proceedings of the 6^{th} International Conference on the Quantitative Evaluation of Systems (QEST)*. Sept. 2009, pp. 187–196.

[75] V. Hashemi, H. Hatefi, and J. Krcál. "Probabilistic Bisimulations for PCTL Model Checking of Interval MDPs". In: *Proceedings of the 1^{st} International Workshop on Synthesis of Continuous Parameters (SynCoP)*. Apr. 2014, pp. 19–33.

[76] M. He, S. Murugesan, and J. Zhang. "Multiple Timescale Dispatch and Scheduling for Stochastic Reliability in Smart Grids with Wind Generation Integration". In: *Proceedings of the 30^{th} IEEE International Conference on Computer Communications (INFOCOM)*. 2011, pp. 461–465.

[77] H. Heitsch and W. Römisch. "Scenario Reduction in Stochastic Programming". In: *Computational Optimization and Applications* 24 (2003), pp. 187–206.

[78] T. A Henzinger. "The Theory of Hybrid Automata". In: *Proceedings of the 11^{th} Annual IEEE Symposium on Logic in Computer Science (LICS)*. July 1996, pp. 278–292.

[79] H. Hermanns, J.-P. Katoen, J. Meyer-Kayser, and M. Siegle. "ETMCC: Model Checking Performability Properties of Markov Chains." In: *DSN*. IEEE Computer Society, 2003, pp. 673–.

[80] H. Hermanns, J.-P. Katoen, J. Meyer-Kayser, and M. Siegle. "Towards Model Checking Stochastic Process Algebra". In: *Integrated Formal Methods*. Ed. by W. Grieskamp, T. Santen, and B. Stoddart. Vol. 1945. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2000, pp. 420–439.

[81] R. A. Howard. *Dynamic Programming and Markov Processes*. Cambridge, MA: Technology Press of Massachusetts Institute of Technology, 1960.

[82] *International Energy Agency, World Energy Outlook 2009*. [Online]: `http://www.worldenergyoutlook.org/docs/weo2009/WEO2009_es_english.pdf`.

[83] L. James. *Road Rage and Aggressive Driving: Steering Clear of Highway Warfare*. Prometheus Books, 2000.

[84] X. Jiang, S. Dawson-Haggerty, P. Dutta, and D. Culler. "Design and Implementation of a High-Fidelity AC Metering Network". In: *Proceedings of the International Conference on Information Processing in Sensor Networks (IPSN)*. IEEE. 2009, pp. 253–264.

[85] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra. "Planning and Acting in Partially Observable Stochastic Domains". In: *Artificial Intelligence* 101.12 (1998), pp. 99–134.

[86] J.-P. Katoen, I. S. Zapreev, E. M. Hahn, H. Hermanns, and D. N. Jansen. "The Ins and Outs of the Probabilistic Model Checker MRMC". In: *Performance Evaluation* 68.2 (Feb. 2011), pp. 90–104.

[87] C. J. Kibert. *Sustainable Construction: Green Building Design and Delivery*. Hoboken, N.J. : John Wiley & Sons, 2013.

[88] W. Kim, M. S. Gupta, G. Y. Wei, and D. Brooks. "System Level Analysis of Fast, Per-Core DVFS Using On-Chip Switching Regulators". In: *Proceedings of the 14th IEEE International Symposium on High Performance Computer Architecture (HPCA)*. Feb. 2008, pp. 123–134.

[89] B. Kirby. "Spinning Reserve from Responsive Loads". In: (2003).

[90] S. Klauer. "The Impact of Driver Inattention on Near-Crash/Crash Risk: An Analysis Using the 100-Car Naturalistic Driving Study". In: *National Highway Traffic Safety Administration* (2006).

[91] U. G. Knight. *Power Systems in Emergencies: From Contingency Planning to Crisis Management*. Chichester, UK: John Wiley & Sons, 2001.

[92] S. Koch, Mathieu J. L., and Callaway D. S. "Modeling and Control of Aggregated Heterogeneous Thermostatically Controlled Loads for Ancillary Services". In: *Proceedings of the 15th Power Systems Computation Conference (PSCC)*. Aug. 2011.

[93] I. O. Kozine and L. V. Utkin. "Interval-Valued Finite Markov Chains". In: *Reliable Computing* 8.2 (2002), pp. 97–113.

[94] V. Kreinovich, A. Neumaier, and G. Xiang. *Towards a Combination of Interval and Ellipsoid Uncertainty*. Tech. rep. UTEP-CS-07-42b. Department of Computer Science, UT-El Paso, 2008.

[95] H. Kress-Gazit, D. C. Conner, H. Choset, A. A. Rizzi, and G. J. Pappas. "Courteous Cars". In: *IEEE Robotics Automation Magazine* 15.1 (Mar. 2008), pp. 30–38.

[96] A. Kučera and O. Stražovský. "On the Controller Synthesis for Finite-State Markov Decision Processes". In: *Proceedings of the 25th Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*. Vol. 3821. LNCS. Springer Berlin Heidelberg, 2005, pp. 541–552.

[97] N. Kuge, T. Yamamura, O. Shimoyama, and A. Liu. "A Driver Behavior Recognition Method Based on a Driver Model Framework". In: *Modelling Driver Behaviour in Automotive Environments*. Ed. by P. C. Cacciabue. Warrendale, PA: Society of Automotive Engineers, 1998, pp. 3–25.

[98] M. Kwiatkowska. "Quantitative Verification: Models, Techniques and Tools". In: *The 6^{th} Joint Meeting on European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering: Companion Papers*. ESEC-FSE companion '07. New York, NY, USA: ACM, 2007, pp. 449–458.

[99] M. Kwiatkowska, G. Norman, and D. Parker. "PRISM 4.0: Verification of Probabilistic Real-Time Systems". In: *Computer Aided Verification (CAV)*. Ed. by G. Gopalakrishnan and S. Qadeer. Vol. 6806. LNCS. Springer, July 2011, pp. 585–591.

[100] M. Kwiatkowska, G. Norman, D. Parker, and J. Sproston. "Performance Analysis of Probabilistic Timed Automata Using Digital Clocks". In: *Formal Modeling and Analysis of Timed Systems (FORMATS)*. Ed. by K. G. Larsen and P. Niebert. Vol. 2791. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2004, pp. 105–120.

[101] M. Kwiatkowska, G. Norman, and R. Segala. "Automated Verification of a Randomized Distributed Consensus Protocol Using Cadence SMV and PRISM". In: *Computer Aided Verification (CAV)*. Ed. by G. Berry, H. Comon, and A. Finkel. Vol. 2102. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2001, pp. 194–206.

[102] M. Kwiatkowska and D. Parker. "Automated Verification and Strategy Synthesis for Probabilistic Systems". In: *Proc. 11^{th} International Symposyum on Automated Technology for Verification and Analysis (ATVA)*. Vol. 8172. LNCS. Springer, 2013, pp. 5–22.

[103] M. Lahijanian, S. B. Andersson, and C. Belta. "Temporal Logic Motion Planning and Control with Probabilistic Satisfaction Guarantees". In: *IEEE Transactions on Robotics* 28.2 (2012), pp. 396–409.

[104] H. P. Le, S. R. Sanders, and E. Alon. "Design Techniques for Fully Integrated Switched-Capacitor DC-DC Converters". In: *IEEE Journal of Solid-State Circuits* 46.9 (Sept. 2011), pp. 2120–2131.

[105] X. Le and M. D. Ilić. "Model Predictive Dispatch in Electric Energy Systems with Intermittent Resources". In: *IEEE International Conference on Systems, Man and Cybernetics (SMC)*. Oct. 2008, pp. 42–47.

[106] J. F. Lehman, J. Laird, and P. Rosenbloom. "A Gentle Introduction to Soar, an Architecture for Human Cognition". In: *Invitation to Cognitive Science*. Ed. by S. Sternberg and D. Scarborough. MIT Press, 1996.

[107] D. Lehmann and M. Rabin. "On the Advantage of Free Choice: A Symmetric and Fully Distributed Solution to the Dining Philosophers Problem". In: *Proceedings of the 8^{th} Annual ACM Symposium on Principles of Programming Languages (POPL)*. 1981, pp. 133–138.

[108] E. Lehmann and G. Casella. *Theory of Point Estimation*. Springer-Verlag, New York, 1998.

[109] J. Levinson, J. Askeland, J. Becker, J. Dolson, D. Held, S. Kammel, J. Z. Kolter, D. Langer, O. Pink, V. Pratt, M. Sokolsky, G. Stanek, D. Stavens, A. Teichman, M. Werling, and S. Thrun. "Towards Fully Autonomous Driving: Systems and Algorithms". In: *IEEE Intelligent Vehicles Symposium (IV)*. June 2011, pp. 163–168.

[110] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, and D. Culler. "TinyOS: An Operating System for Sensor Networks". In: *Ambient intelligence*. Springer, 2005, pp. 115–148.

[111] W. Li, D. Sadigh, S. S. Sastry, and S. A. Seshia. "Synthesis for Human-in-the-Loop Control Systems". In: *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*. Ed. by E. Ábrahám and K. Havelund. Vol. 8413. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2014, pp. 470–484.

[112] C.-C. Lin, H. Peng, and J. W. Grizzle. "A Stochastic Control Strategy for Hybrid Electric Vehicles". In: *Proceedings of the American Control Conference (ACC)*. Vol. 5. June 2004, pp. 4710–4715.

[113] *Linking Driving Behavior to Automobile Accidents and Insurance Rates.* `http://www.progressive.com/Content/pdf/newsroom/snapshot_report_final_070812.pdf`.

[114] S. G. Loizou and K. J. Kyriakopoulos. "Automatic Synthesis of Multiagent Motion Tasks Based on LTL Specifications". In: *Proceedings of the 43rd IEEE Conference on Decision and Control (CDC)*. 2004, pp. 153–158.

[115] N. Lu, D. P. Chassin, and S. E. Widergren. "Modeling Uncertainties in Aggregated Thermostatically Controlled Loads Using a State Queueing Model". In: *IEEE Transactions on Power Systems* 20.2 (May 2005), pp. 725–733.

[116] M. Maasoumy, Q. Zhu, C. Li, F. Meggers, and A. L. Sangiovanni-Vincentelli. "Co-Design of Control Algorithm and Embedded Platform for Building HVAC Systems". In: *Proceedings of the ACM/IEEE International Conference on Cyber-Physical Systems (ICCPS)*. Apr. 2013, pp. 61–70.

[117] J. Markoff. "Google Cars Drive Themselves, in Traffic". In: *The New York Times* 10 (2010), A1.

[118] A. McCallum, D. Freitag, and F. C. N. Pereira. "Maximum Entropy Markov Models for Information Extraction and Segmentation". In: *Proceedings of the 17th International Conference on Machine Learning (ICML)*. ICML '00. Morgan Kaufmann Publishers Inc., 2000, pp. 591–598.

[119] *Microsoft Kinect.* `http://www.xbox.com/en-US/KINECT`.

[120] G. E. Monahan. "A Survey of Partially Observable Markov Decision Processes: Theory, Models, and Algorithms". In: *Management Science* 28.1 (Jan. 1982), pp. 1–16.

[121] J. M. Morales, A. J. Conejo, and J. Perez-Ruiz. "Economic Valuation of Reserves in Power Systems with High Penetration of Wind Power". In: *IEEE Transactions on Power Systems* 24.2 (May 2009), pp. 900–910.

[122] *MOSEK*. http://www.mosek.com.

[123] T. Mount, L. Anderson, J. Cardell, A. Lamadrid, S. Maneevitjit, B. Thomas, and R. Zimmerman. "The Economic Implications of Adding Wind Capacity to a Bulk Power Transmission Network Applied Economics and Management". In: *Technical Report*. Cornell University, Ithaca, NY, 2008.

[124] Y. Nesterov and A. Nemirovskii. *Interior-Point Polynomial Algorithms in Convex Programming*. Society for Industrial and Applied Mathematics, 1994.

[125] A. Nilim and L. El Ghaoui. "Robust Control of Markov Decision Processes with Uncertain Transition Matrices". In: *Operations Research* 53.5 (2005), pp. 780–798.

[126] P. Nuzzo, A. Puggelli, S. A. Seshia, and A. L. Sangiovanni-Vincentelli. "CalCS: SMT Solving for Non-linear Convex Constraints". In: *Proceedings of the Conference on Formal Methods in Computer-Aided Design (FMCAD)*. FMCAD Inc., 2010, pp. 71–80.

[127] *Opower*. `http://opower.com/`.

[128] S. Owicki and L. Lamport. "Proving Liveness Properties of Concurrent Programs". In: *ACM Transactions of Programming Languages and Systems* 4.3 (July 1982), pp. 455–495.

[129] M. Panou, E. Bekiaris, and V. Papakostopoulos. "Modelling Driver Behaviour in European Union and International Projects". In: *Modelling Driver Behaviour in Automotive Environments*. Ed. by P. C. Cacciabue. Springer London, 2007, pp. 3–25.

[130] A. Papavasiliou and S. S. Oren. "Large-Scale Integration of Deferrable Demand and Renewable Energy Sources". In: *IEEE Transactions on Power Systems* 29.1 (Jan. 2014), pp. 489–499.

[131] A. Papavasiliou and S. S. Oren. "Multiarea Stochastic Unit Commitment for High Wind Penetration in a Transmission Constrained Network". In: *Operations Research* 61.3 (2013), pp. 578–592.

[132] A. Papavasiliou and S. S. Oren. "Supplying Renewable Energy to Deferrable Loads: Algorithms and Economic Analysis". In: *IEEE Power and Energy Society General Meeting (PES)*. July 2010, pp. 1–8.

[133] A. Papavasiliou, S. S. Oren, and R. P. O'Neill. "Reserve Requirements for Wind Power Integration: A Scenario-Based Stochastic Programming Framework". In: *IEEE Transactions on Power Systems* 26.4 (Nov. 2011), pp. 2197–2206.

[134] A. Pentland and A. Lin. "Modeling and Prediction of Human Behavior". In: *Neural Computation* 11 (1995), pp. 229–242.

[135] P. Pillai and K. G. Shin. "Real-Time Dynamic Voltage Scaling for Low-Power Embedded Operating Systems". In: *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP)*. ACM, 2001, pp. 89–102.

[136] N. Piterman, A. Pnueli, and Y. Sa'ar. "Synthesis of Reactive(1) Designs". In: *Verification, Model Checking, and Abstract Interpretation*. Ed. by E. A. Emerson and K. S. Namjoshi. Vol. 3855. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2006, pp. 364–380.

[137] A. Pnueli. "The Temporal Logic of Programs". In: *Proceedings of the 18th Annual Symposium on the Foundations of Computer Science (FOCS)*. Oct. 1977, pp. 46–57.

[138] K. Porter and J. Rogers. "Survey of Variable Generation Forecasting in the West". In: *NREL Subcontract Report SR-5500-54457*. Apr. 2012, p. 56.

[139] A. Puggelli, W. Li, A. L. Sangiovanni-Vincentelli, and S. A. Seshia. "Polynomial-Time Verification of PCTL Properties of MDPs with Convex Uncertainties". In: *Computer Aided Verification (CAV)*. Ed. by N. Sharygina and H. Veith. Vol. 8044. Lecture Notes in Computer Science. Springer Berlin Heidelberg, July 2013, pp. 527–542.

[140] A. Puggelli, M. M. R. Mozumdar, L. Lavagno, and A. L. Sangiovanni-Vincentelli. *Routing-Aware Design of Indoor Wireless Sensor Networks Using an Interactive Tool*. Dec. 2013.

[141] A. Puggelli, A. L. Sangiovanni-Vincentelli, and S. A. Seshia. "Robust Strategy Synthesis for Probabilistic Systems Applied to Risk-Limiting Renewable-Energy Pricing". In: *Proceedings of the ACM/IEEE International Conference on Embedded Software (EMSOFT)*. Oct. 2014.

[142] M. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, 1994.

[143] J. Rasmussen. *Information Processing and Human-Machine Interaction: An Approach to Cognitive Engineering*. New York, NY, USA: Elsevier Science Inc., 1986.

[144] M. Regan, J. Lee, and K. Young. *Driver distraction: Theory, effects, and mitigation*. CRC Press, 2008.

[145] P. A. Ruiz, R. C. Philbrick, and P. W. Sauer. "Wind Power Day-Ahead Uncertainty Management through Stochastic UC Policies". In: *Power Systems Conference and Exposition*. Mar. 2009, pp. 1–9.

[146] S. M. Ryan, R. J.-B. Wets, D. L. Woodruff, C. Silva-Monroy, and J.-P. Watson. "Toward Scalable, Parallel Progressive Hedging for Stochastic Unit Commitment". In: *IEEE Power and Energy Society General Meeting (PES)*. July 2013, pp. 1–5.

[147] D. Sadigh, K. Driggs-Campbell, A. Puggelli, W. Li, V. Shia, R. Bajcsy, A. L. Sangiovanni-Vincentelli, S. S. Sastry, and S. A. Seshia. "Data-Driven Probabilistic Modeling and Verification of Human Driver Behavior". In: *Formal Verification and Modeling in Human-Machine Systems*. AAAI Spring Symposium Series. Mar. 2014. URL: http://www.aaai.org/ocs/index.php/SSS/SSS14/paper/view/7749/7759.

[148] D. Sadigh, E. S. Kim, S. Coogan, S. S. Sastry, and S. A. Seshia. "A Learning Based Approach to Control Synthesis of Markov Decision Processes for Linear Temporal Logic Specifications". In: *To appear in Proceedings of the 53rd IEEE Annual Conference on Decision and Control (CDC)*. 2014.

[149] R. Segala. "Modeling and Verification of Randomized Distributed Real-Time Systems". PhD thesis. Massachusetts Institute of Technology, Cambridge, MA, 1995.

[150] K. Sen, M. Viswanathan, and G. Agha. "Model-Checking Markov Chains in the Presence of Uncertainties". In: *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*. Ed. by Holger Hermanns and Jens Palsberg. Vol. 3920. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2006, pp. 394–410.

[151] K. Sen, M. Viswanathan, and G. Agha. "VESTA: A Statistical Model-Checker and Analyzer for Probabilistic Systems". In: *Proceedings of the 2nd International Conference on the Quantitative Evaluation of Systems (QEST)*. Sept. 2005, pp. 251–252.

[152] R. F. Serfozo. "An Equivalence between Continuous and Discrete Time Markov Decision Processes". In: *Operations Research* 27.3 (May 1979), pp. 616–620.

[153] V. A Shia, Y. Gao, R. Vasudevan, K. Driggs-Campbell, T. Lin, F. Borrelli, and R. Bajcsy. "Semiautonomous Vehicular Control Using Driver Modeling". In: *IEEE Transactions on Intelligent Transportation Systems* PP.99 (2014), pp. 1–14.

[154] T. Shiina and J. R. Birge. "Stochastic Unit Commitment Problem". In: *International Transactions in Operational Research* 11.95 (2004), pp. 19–32.

[155] R. Sioshansi. "Evaluating the Impacts of Real-Time Pricing on the Cost and Value of Wind Generation". In: *IEEE Transactions on Power Systems* 25.2 (May 2010), pp. 741–748.

[156] R. Sioshansi. "OR Forum: Modeling the Impacts of Electricity Tariffs on Plug-In Hybrid Electric Vehicle Charging, Costs, and Emissions". In: *Operations Research* 60.3 (2012), pp. 506–516.

[157] R. Sioshansi and W. Short. "Evaluating the Impacts of Real Time Pricing on the Usage of Wind Power Generation". In: *IEEE Transactions on Power Systems* 24.2 (May 2009), pp. 516–524.

[158] E. J. Sondik. "The Optimal Control of Partially Observable Markov Processes." PhD thesis. Stanford University, Stanford, CA, 1971.

[159] W. Stewart. "Introduction to the Numerical Solution of Markov Chains". In: *Princeton University Press* (1994).

[160] S. Stoft. *Power System Economics: Designing Markets for Electricity*. New York, NY: John Wiley & Sons, 2002.

[161] S. Takriti, J. R. Birge, and E. Long. "A Stochastic Model for the Unit Commitment Problem". In: *IEEE Transactions on Power Systems* 11.3 (Aug. 1996), pp. 1497–1508.

[162] D. Tank and J. J. Hopfield. "Simple 'Neural' Optimization Networks: An A/D Converter, Signal Decision Circuit, and a Linear Programming Circuit". In: *IEEE Transactions on Circuits and Systems* 33.5 (May 1986), pp. 533–541.

[163] W. Thomas. "Languages, Automata, and Logic". In: *Handbook of Formal Languages*. Springer, 1996, pp. 389–455.

[164] A. Tuohy, P. Meibom, E. Denny, and M. O'Malley. "Unit Commitment for Systems with High Wind Penetration". In: *IEEE Transactions on Power Systems* 24.2 (May 2009), pp. 592–601.

[165] C. Urmson et al. "Autonomous Driving in Urban Environments: Boss and the Urban Challenge". In: *Journal of Field Robotics* 25.8 (2008), pp. 425–466.

[166] P. Varaiya. "Smart Cars on Smart Roads: Problems of Control". In: *IEEE Transactions on Automatic Control* 38.2 (Feb. 1993), pp. 195–207.

[167] P. P. Varaiya, F. F. Wu, and J. W. Bialek. "Smart Operation of Smart Grid: Risk-Limiting Dispatch". In: *Proceedings of the IEEE* 99.1 (2011), pp. 40–57.

[168] M.Y. Vardi. "Automatic Verification of Probabilistic Concurrent Finite State Programs". In: *Proceedings of the 26th Annual Symposium on the Foundations of Computer Science (FOCS)*. Oct. 1985, pp. 327–338.

[169] R. Vasudevan, V. Shia, G. Yiqi, R. Cervera-Navarro, R. Bajcsy, and F. Borrelli. "Safe Semi-Autonomous Control with Enhanced Driver Modeling". In: *American Control Conference (ACC)*. June 2012, pp. 2896–2903.

[170] Y. H Wan. *Long-term Wind Power Variability*. NREL Technical Report TP-5500-53637. [Online]: `http://www.nrel.gov/docs/fy12osti/53637.pdf`. Jan. 2012.

[171] J. Wang, M. Shahidehpour, and Z. Li. "Security-Constrained Unit Commitment with Volatile Wind Power Generation". In: *IEEE Transactions on Power Systems* 23.3 (Aug. 2008), pp. 1319–1327.

[172] E. Wolff, U. Topcu, and R. Murray. "Robust Control of Uncertain Markov Decision Processes with Temporal Logic Specifications". In: *International Conference on Decision and Control (CDC)*. 2012, pp. 3372–3379.

[173] H. L. S. Younes. "Ymer: A Statistical Model Checker". In: *Computer Aided Verification (CAV)*. Ed. by Kousha Etessami and SriramK. Rajamani. Vol. 3576. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2005, pp. 429–433.

[174] H. L. S. Younes and R. G. Simmons. "Probabilistic Verification of Discrete Event Systems Using Acceptance Sampling". In: *Computer Aided Verification (CAV)*. Ed. by E. Brinksma and K. G. Larsen. Vol. 2404. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2002, pp. 223–235.

[175] J. Zhang, J. D. Fuller, and S. Elhedhli. "A Stochastic Programming Model for a Day-Ahead Electricity Market with Real-Time Reserve Shortage Pricing". In: *IEEE Transactions on Power Systems* 25.2 (May 2010), pp. 703–713.

Appendix A

Example of the LP Formulation to Verify the Unbounded Until Operator

This appendix reports the full LP formulation that was used to verify property $\phi = \boldsymbol{P}_{\geq 0.4}[\ \vartheta\, \boldsymbol{\mathcal{U}}\, \omega\ ]$ on the example in Figure 2.3. Problem (4.19) written with the data of the model has 19 variables and 11 constraints. Note that variables are constrained by default to be non-negative in most publicly-available LP solvers, so the LP formulation only reports variables that should instead be treated as unconstrained (labeled as "free").

$$
\begin{aligned}
\max_{\mathbf{x},\lambda^a_{1,s},\boldsymbol{\lambda^a_{2,s}},\boldsymbol{\lambda^a_{3,s}}} \quad & x_0 + x_3 \qquad (A.1)\\
\text{Subject to: } & x_2 = 1\\
& x_1 = 0\\
& x_0 \leq \lambda^a_{1,s_0} + 0.6\lambda^a_{2,s_0s_1} + 0.2\lambda^a_{2,s_0s_2} - 0.8\lambda^a_{3,s_0s_1} - 0.5\lambda^a_{3,s_0s_2}\\
& x_1 - \lambda^a_{1,s_0} + \lambda^a_{3,s_0s_1} - \lambda^a_{2,s_0s_1} = 0\\
& x_2 - \lambda^a_{1,s_0} + \lambda^a_{3,s_0s_2} - \lambda^a_{2,s_0s_2} = 0\\
& x_0 \leq x_3\\
& x_3 \leq \lambda^a_{1,s_3} + 0.1\lambda^a_{2,s_3s_0} + 0.5\lambda^a_{2,s_3s_1} + 0.3\lambda^a_{2,s_3s_2} - 0.5\lambda^a_{3,s_3s_0} - 0.8\lambda^a_{3,s_3s_1} - 0.4\lambda^a_{3,s_3s_2}\\
& x_0 - \lambda^a_{1,s_3} + \lambda^a_{3,s_3s_0} - \lambda^a_{2,s_3s_0} = 0\\
& x_1 - \lambda^a_{1,s_3} + \lambda^a_{3,s_3s_1} - \lambda^a_{2,s_3s_1} = 0\\
& x_2 - \lambda^a_{1,s_3} + \lambda^a_{3,s_3s_2} - \lambda^a_{2,s_3s_2} = 0\\
& x_3 \leq \lambda^b_{1,s_3} + 0.3\lambda^b_{2,s_3s_3} + 0.4\lambda^b_{2,s_3s_2} - 0.7\lambda^b_{3,s_3s_3} - 0.6\lambda^b_{3,s_3s_2}\\
& x_2 - \lambda^b_{1,s_3} + \lambda^b_{3,s_3s_2} - \lambda^b_{2,s_3s_2} = 0\\
& x_3 - \lambda^b_{1,s_3} + \lambda^b_{3,s_3s_3} - \lambda^b_{2,s_3s_3} = 0\\
\text{Free: } & \lambda^a_{1,s_0}, \lambda^a_{1,s_3}, \lambda^b_{1,s_3}
\end{aligned}
$$

Appendix B

Proof of Convergence of the Contraction Mapping

In this appendix, we give details about the proof of Proposition 4.4.

To keep the appendix self-contained, we re-define:

Definition B.1. ***Contraction.*** *Let (B, d) be a metric space and $g : B \to B$. Function g is a* contraction *if there is a real number θ, $0 \leq \theta < 1$, such that:*

$$d\left(g(u), g(v)\right) \leq \theta d(u, v) \quad \forall u, v \in B \tag{B.1}$$

In the following, we will use:

Proposition B.1. ***Contraction Mapping.*** *Let (B, d) be a complete metric space and $g : B \to B$ a contraction. Then there exists a unique point $x^* \in B$ such that:*

$$g(x^*) = x^*$$

Additionally, if $x \in B$, then:

$$\lim_{k \to +\infty} g^k(x) = x^*$$

We use the mapping $g = G$ defined as:

$$\mathbf{x}^i = G^i(\mathbf{x}^{i-1}) = \begin{cases} 0;\ 1; & \forall s \in S^{no};\ \forall s \in S^{yes}; \\ 0; & \forall s \in S^{?} \wedge i = 0; \\ \max\limits_{a \in \mathcal{A}(s)} \max\limits_{\mathbf{f}_s^a \in \mathcal{F}_s^a} \sum\limits_{s'} f_{ss'}^a x_{s'}^{i-1} & \forall s \in S^{?} \wedge i > 0 \end{cases} \tag{B.2}$$

where $S^{yes} \stackrel{def}{=} Sat\left(\boldsymbol{P}_{\geq 1}[\phi_1\ \boldsymbol{\mathcal{U}}\ \phi_2]\right)$, $S^{no} \stackrel{def}{=} Sat\left(\boldsymbol{P}_{\leq 0}[\phi_1\ \boldsymbol{\mathcal{U}}\ \phi_2]\right)$ and $S^{?} = S \setminus (S^{no} \cup S^{yes})$.

To simplify notation in the proof, we use the weighted maximum norm $\| \ . \ \|_{\mathbf{w}}$ of a vector $v \in \mathbb{R}^N$ defined as:

$$\| \ . \ \|_{\mathbf{w}} = \max_{i=1 \cdots N} \frac{|\ v_i\ |}{w_i} \tag{B.3}$$

where w_i is the scalar weight associated to each element of v.

The results presented in the following are actually valid for any vector $\mathbf{w} \mid w_i > 0, \forall i \in \mathbb{R}^N$, since a contraction drives the residual norm to 0, which is independent of $\mathbf{w}$. It is thus correct to use the infinity norm, i.e., $\mathbf{w} = \mathbf{1}$, in Section 4.2.3.2.

We can now state:

Proposition B.2. *Mapping G is a contraction over the metric space $(\mathbb{R}^N, \| \, . \, \|_{\mathbf{w}})$.*

Proof. The proof follows closely the ones in references [15], [24] (Vol. II, Section 2.4) and [172]. Those proofs refer to the control settings, where the optimal *strategy* can be selected. Hence, the contraction needs to hold for only one of the available strategies, i.e., the optimal one (existential quantification). Conversely, in the verification settings, the contraction needs to hold across all available *adversaries*, because we consider the worst-case resolution of non-determinism (universal quantification).

Further, as in the work by Wolff et al. [172], we quantify across all nature behaviors. This quantification is possible thanks to Assumption 2.2, which guarantees that if the probability of a transition is zero (non-zero) for at least one probability distribution in the uncertainty set, then it is zero (non-zero) for all probability distributions in the set.

For the sake of brevity, in the following we will only consider the calculation of $P_s^{min}[\psi]$, but the same reasoning applies also for the maximization problem.

We start from partitioning the state space S into the three subsets S^{yes}, S^{no} and $S^{?}$. Since the probabilities $P_s^{min}[\psi]$ are fixed in all states $s \in S^{yes} \cup S^{no}$ ($P_s^{min}[\psi] = 1$ if $s \in S^{yes}$ and $P_s^{min}[\psi] = 0$ if $s \in S^{no}$), we do not need to consider these states explicitly. In particular, we perform the following transformation on the Convex-MDP underlying graph. We collapse the set S^{yes} into one terminal state t, and eliminate all states $s \in S^{no}$ from the graph. This transformation is fundamental, together with Assumption 2.2, to guarantee that all possible adversaries Adv are *proper* in the transformed Convex-MDP, i.e., any adversary almost certainly reaches the terminal state t for all transition matrices $F^a \in \mathcal{F}^a, \forall a \in A$ [24].

We will now work with the new state space $S^{\dagger} = S^{?} \cup \{t\}$, and, for simplicity, we redefine $N = |S^{\dagger}|$. We further partition $S^{\dagger}$ as follows. Let $S_1 = \{t\}$ and for $q = 2, 3, \cdots$ compute:

$$S_q = \{s \in S^{\dagger} \mid s \notin S_1 \cup \cdots \cup S_{q-1}, \min_{a \in \mathcal{A}(s)} \max_{s' \in S_1 \cup \cdots \cup S_{q-1}} \min_{f_s^a \in \mathcal{F}_s^a} f_{ss'}^a > 0\}$$

Let r be the largest integer such that S_r is nonempty. Since all adversaries are *proper*, we are guaranteed that $\cup_{q=1}^{r} S_q = S^{\dagger}$.

We now need to choose weights $w_s, \forall s \in S^{\dagger}$ such that G is a contraction with respect to $\|.\|_{\mathbf{w}}$. First, we take the i^{th} component w_i to be the same for states s_i in the same set S_q. Then we set $w_i = y_q$ if $s_i \in S_q$, where $y_1, \cdots, y_r$ are scalars satisfying:

$$1 = y_1 < y_2 < \cdots < y_r$$

Further, let:

$$\xi = \min_{q=2,\cdots,r} \min_{a \in A} \min_{s \in S_q} \min_{\mathbf{f}_s^a \in \mathcal{F}_s^a} \sum_{s' \in S_1 \cup \cdots \cup S_{q-1}} f_{ss'}^a$$

with, by Assumption 2.2, $0 < \xi \leq 1$.

The rest of the proof goes as follows. First, we will show that if we can find $y_2, \cdots, y_r$ such that, for $q = 2, \cdots, r$:

$$\frac{y_r}{y_q}(1-\xi) + \frac{y_{q-1}}{y_q} \leq \theta$$

for some $\theta < 1$, then G is a contraction. Second, we will prove that such values always exist. We begin by defining:

$$G_s(\mathbf{x}) = \min_{a \in \mathcal{A}(s)} \min_{\mathbf{f}_s^a \in \mathcal{F}_s^a} \mathbf{x}^T \mathbf{f}_s^a$$
$$G_s^a(\mathbf{x}) = \min_{\mathbf{f}_s^a \in \mathcal{F}_s^a} \mathbf{x}^T \mathbf{f}_s^a$$

i.e., the element of the output of mapping G corresponding to state $s \in S^\dagger$, when mapping G is applied to vector $\mathbf{x} \in \mathbb{R}^N$, and the same element when mapping G is evaluated only at the fixed action $a \in \mathcal{A}(s)$.

For all vectors $\mathbf{v}, \mathbf{u} \in \mathbb{R}^N$, we determine $a \in \mathcal{A}(s)$ such that:

$$a = \operatorname*{argmin}_{\mathcal{A}(s)} G_s(u)$$

We can thus write for all $s \in S^\dagger$:

$$\begin{aligned} G_s(\mathbf{v}) - G_s(\mathbf{u}) &= G_s(\mathbf{v}) - G_s^a(\mathbf{u}) \\ &\leq G_s^a(\mathbf{v}) - G_s^a(\mathbf{u}) \\ &= \textstyle\sum_{s'} \left(V_{ss'}^a v_{s'} - U_{ss'}^a u_{s'}\right) \\ &\leq \textstyle\sum_{s'} M_{ss'}^a \left(v_{s'} - u_{s'}\right) \end{aligned}$$

where:

$$\mathbf{V}_s^a = \operatorname*{argmin}_{\mathbf{f}_s^a \in \mathcal{F}_s^a} \mathbf{v}^T \mathbf{f}_s^a$$
$$\mathbf{U}_s^a = \operatorname*{argmin}_{\mathbf{f}_s^a \in \mathcal{F}_s^a} \mathbf{u}^T \mathbf{f}_s^a$$
$$M_{ss'}^a = \max\left\{V_{ss'}^a \left(v_{s'} - u_{s'}\right), U_{ss'}^a \left(v_{s'} - u_{s'}\right)\right\}$$

Let $q(s)$ be a function such that state s belongs to the set $S_{q(s)}$. Then, for any constant c:

$$\|\mathbf{v} - \mathbf{u}\|_{\mathbf{w}} \Rightarrow v_s - u_s \leq c y_{q(s)}, \quad \forall s \in S^\dagger$$

We can thus write $\forall s \in S_q$ and $q = 1, \cdots, r$:

$$\begin{aligned}
\frac{G_s(v) - G_s(u)}{c y_{q(s)}} &\leq \frac{1}{y_{q(s)}} \sum_{s' \in S^\dagger} M^a_{ss'} y_{q(s')} \\
&\leq \frac{y_{q(s)-1}}{y_{q(s)}} \sum_{s' \in S_1 \cup \cdots \cup S_{q(s)-1}} M^a_{ss'} \\
&+ \frac{y_r}{y_{q(s)}} \sum_{s' \in S_{q(s)} \cup \cdots \cup S_r} M^a_{ss'} \\
&= \left(\frac{y_{q(s)-1}}{y_{q(s)}} - \frac{y_r}{y_{q(s)}} \right) \sum_{s' \in S_1 \cup \cdots \cup S_{q(s)-1}} M^a_{ss'} \\
&+ \frac{y_r}{y_{q(s)}} \leq \left(\frac{y_{q(s)-1}}{y_{q(s)}} - \frac{y_r}{y_{q(s)}} \right) \xi + \frac{y_r}{y_{q(s)}} \leq \theta
\end{aligned}$$

We have thus proved that:

$$\frac{G_s(v) - G_s(u)}{w_i} \leq c\theta$$

for an arbitrary state $s \in S^\dagger$. Taking the maximum over $S^\dagger$, we get:

$$\|G(v) - G(u)\|_{\mathbf{w}} \leq c\theta, \ \forall \mathbf{u}, \mathbf{v} \in \mathbb{R}^N \text{ s.t. } \|\mathbf{v} - \mathbf{u}\| \leq c$$

so G is a contraction over the metric space $(\mathbb{R}^N, \| \ . \ \|_{\mathbf{w}})$, and:

$$\theta = \max_{1 \leq q \leq r} \frac{y_r}{y_q}(1 - \xi) + \frac{y_{q-1}}{y_q} \tag{B.4}$$

is the corresponding contraction factor.

Finally, we constructively prove by induction that it is always possible to find scalars $y_1, \cdots, y_r$ such that the above assumptions hold. As the base case, we set $y_0 = 0, y_1 = 1$. At the induction step, we suppose that $y_2, \cdots, y_q$ have already been determined. If $\xi = 1$, we set $y_{q+1} = y_q + 1$. If $\xi < 1$, we set $y_{q+1} = \frac{1}{2}(y_q + m_q)$ where:

$$m_q = \min_{1 \leq i \leq q} \left\{ y_i + \frac{\xi}{1 - \xi}(y_i - y_{i-1}) \right\}$$

With these choices, we are guaranteed that:

$$m_{q+1} = \min \left\{ m_q, y_i + \frac{\xi}{1 - \xi}(y_i - y_{i-1}) \right\}$$

so, by induction, we have that $y_q < y_{q+1} < m_{q+1}$, and we can construct the required sequence. $\square$

Appendix C

Python Implementation of the Verification Algorithm

We give details about the software implementation of the model-checking algorithm. In particular, we present the code implementation in Python, due to its higher readability. The full Python implementation can be found at [1]. The Java implementation of the code will instead be available open source within the PRISM Model Checker distribution.

As explained in Chapter 4, the tool exploits class inheritance to maximize the reuse of code across the different flavors of Convex-MDPs. In particular, as shown in Figure C.1, the classes for Interval/Ellipsoidal/Likelihood/Entropy-MDPs all inherit from a common Convex-MDP class. The Convex-MDP class contains the code to model check every operator in the PCTL logic. The subclasses instead only implement the specific routines to determine the worst-case resolution of uncertainty for the corresponding uncertainty model, i.e., they solve the inner optimization problems in Equation (4.9)-(4.12) on the left, reported below for convenience.

$$P_s^{max}[\psi] = \max_{a \in \mathcal{A}(s)} \max_{\mathbf{f}_s^a \in \mathcal{F}_s^a} P_s(a, \mathbf{f}_s^a)[\psi] \quad \Rightarrow \quad P_s^{max}[\psi] \stackrel{?}{\leq} p$$

$$P_s^{min}[\psi] = \min_{a \in \mathcal{A}(s)} \min_{\mathbf{f}_s^a \in \mathcal{F}_s^a} P_s(a, \mathbf{f}_s^a)[\psi] \quad \Rightarrow \quad P_s^{min}[\psi] \stackrel{?}{\geq} p$$

$$\mathbb{E}_s^{max}[\rho] = \max_{a \in \mathcal{A}(s)} \max_{\mathbf{f}_s^a \in \mathcal{F}_s^a} \mathbb{E}_s(a, \mathbf{f}_s^a)[\rho] \quad \Rightarrow \quad \mathbb{E}_s^{max}[\rho] \stackrel{?}{\leq} v$$

$$\mathbb{E}_s^{min}[\rho] = \min_{a \in \mathcal{A}(s)} \min_{\mathbf{f}_s^a \in \mathcal{F}_s^a} \mathbb{E}_s(a, \mathbf{f}_s^a)[\rho] \quad \Rightarrow \quad \mathbb{E}_s^{min}[\rho] \stackrel{?}{\geq} v$$

To ease the comprehension of the code, we first introduce the data structures used to store the model data. In particular, we will show an example of the content of these data structures for the simple Interval-MDP introduced in Chapter 2 and reported below in Figure C.2 for convenience.

The data structure *fastIdx* is a list (across states) of lists (across actions) of lists of next state

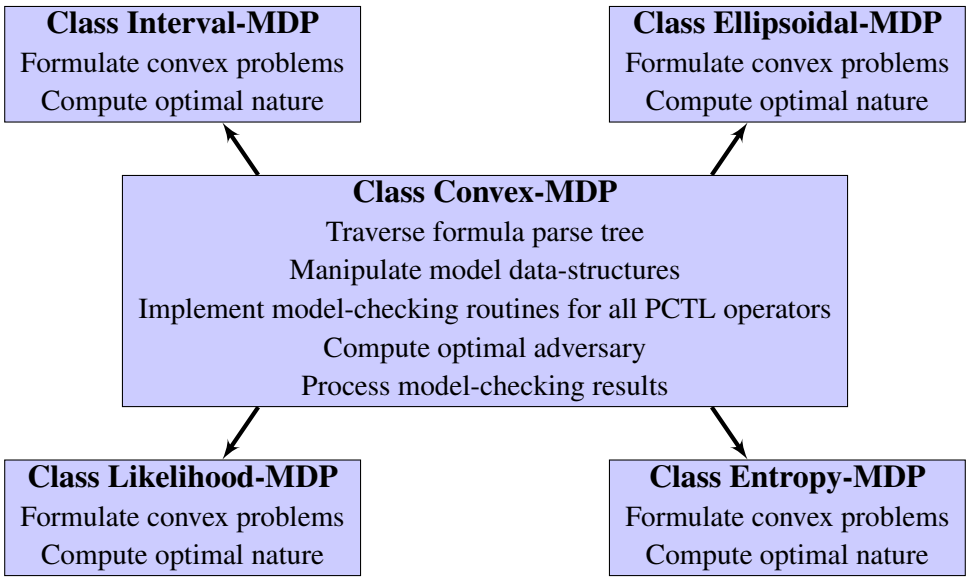


Figure C.1: Class inheritance diagram for the Python implementation of the verification algorithm. Each block represents a class and the arrow points to the class which inherits. For each class, we also list the implemented functionality.

indices. For the Interval-MDP in the example, *fastIdx* contains:

$$
\begin{array}{rl}
 & [\\
\text{/* State } s_0\text{, action a and b*/} & [[1,2],[3]], \\
\text{/* State } s_1\text{, action a*/} & [[1,2]], \\
\text{/* State } s_2\text{, action a*/} & [[1]], \\
\text{/* State } s_3\text{, action a and b*/} & [[0,1,2],[2,3]] \\
 &]
\end{array}
$$

The data structure *par1* is a list (across states) of lists (across actions) of lists (across next states) of uncertain transition probabilities. For example, for the interval model of uncertainty, the transitions probabilities are stored as a list containing the list of lower bounds and the list of upper bounds on the transition probabilities. If a transition is not associated to an uncertainty set, only the *certain* transition probability is reported. For the Interval-MDP in the example, *par1* contains:

$$
\begin{array}{rl}
 & [\\
\text{/* State } s_0\text{, action a and b*/} & [[[0.6,0.2],[0.8,0.5]],[[1.0]]], \\
\text{/* State } s_1\text{, action a*/} & [[[0.5,0.5]]], \\
\text{/* State } s_2\text{, action a*/} & [[[1.0]]], \\
\text{/* State } s_3\text{, action a and b*/} & [[[0.1,0.5,0.3],[0.5,0.8,0.4]],[[0.4,0.3],[0.6,0.7]]] \\
 &]
\end{array}
$$

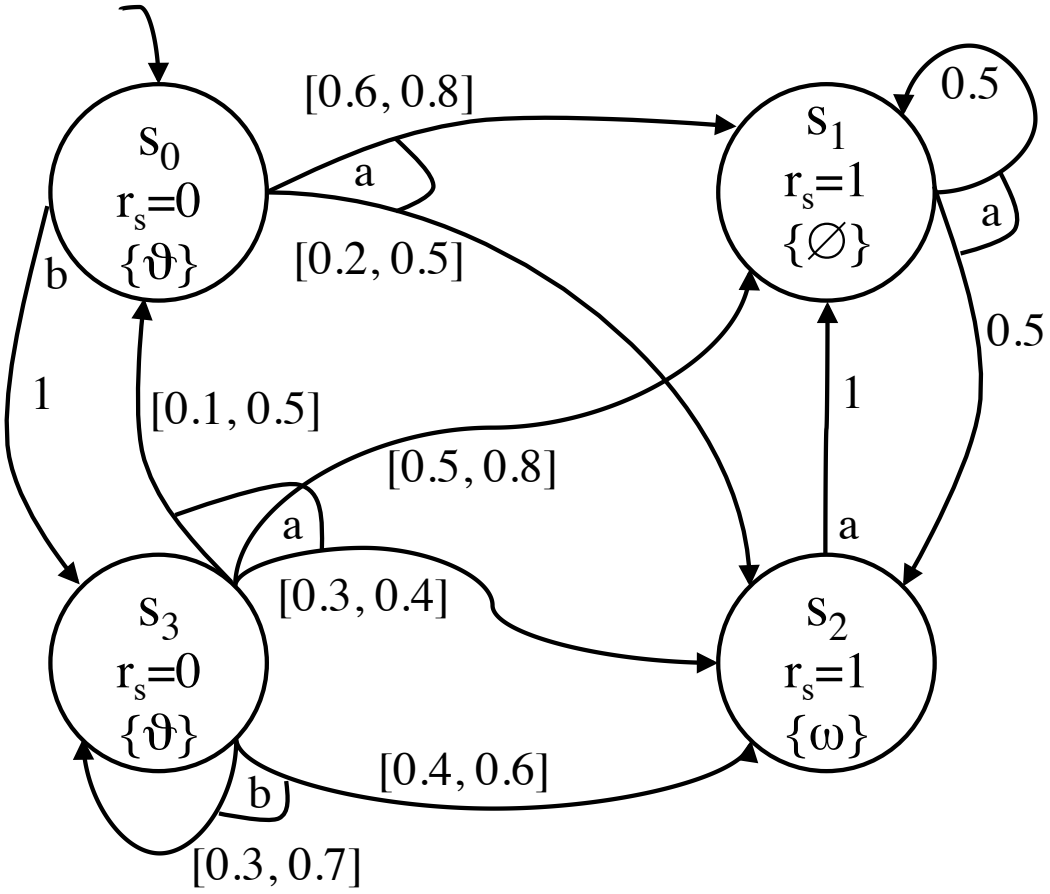


Figure C.2: Example of a Convex-MDP $\mathcal{M}_{\mathcal{C}}$. In particular, $S = \{s_0 \cdots s_3\}$, $S_0 = \{s_0\}$, $A = \{a, b\}$, $\Omega = \{\omega, \vartheta\}$, $\mathcal{A} : \{s_1, s_2\} \rightarrow \{a\}$; $\{s_0, s_3\} \rightarrow \{a, b\}$, $L : \{s_0, s_3\} \rightarrow \vartheta$; $\{s_2\} \rightarrow \omega$. Uncertainty in transition probabilities is captured in the intervals shown next to each transition. For example, $\mathcal{F}^a_{s_0} = \{\mathbf{f}^a_{s_0} \in \mathbb{R}^N \mid [0, 0.6, 0.2, 0] \leq \mathbf{f}^a_{s_0} \leq [0, 0.8, 0.5, 0], \sum_{s' \in S} f^a_{ss'} = 1\}$. The reward structure r associated to $\mathcal{M}_{\mathcal{C}}$ is as follows: $r_s : \{s_0, s_3\} \rightarrow 0$; $\{s_1, s_2\} \rightarrow 1$, $r_a : \{(s_0, b), (s_2, a)\} \rightarrow 0$; $\{(s_0, a), (s_3, a), (s_1, a)\} \rightarrow 1$; $\{(s_3, b)\} \rightarrow 2$. The values of r_a are not shown in the figure to avoid clutter.

In the following, we first report the commented code for the implementation of the model-checking routines for the temporal operators of the PCTL logic (*Next*, *Bounded Until* and the Value Iteration routine for *Unbounded Until*). Only the routine computing the maximum satisfaction probability is reported for brevity. The routines for the verification of the reward operators follow a similar pattern. We then give details about the procedures to solve the inner optimization problems (both maximization and minimization) for the interval and ellipsoidal models of uncertainty. We conclude by reporting the routines to formulate the monolithic instances of the convex problems used to model check the *Unbounded Until* operator using the CP procedure, as presented in Section 4.2.3.1.

Next Operator

```
# Routine to compute Pmax[Xphi], i.e., the probability of reaching a state
    satisfying phi in one step.
# Class: Convex-MDP
# Inputs:
# list[int] yesStates: list of indices of the states that satisfy phi
# Outputs:
# list[double] x_new: list of satisfaction probabilities for each state
def computeNextMax(self, yesStates):
    # Retrieve the indices of the states reachable in one step for all states
    # and actions (fastIdx) and the information about the convex uncertainty
    # set (par1)
    fastIdx, par1 = self.computeUncertainties()

    # Initialize the result list
    x_new = [0]*self.nStates
    for s in yesStates:
        x_new[s] = 1

    # Save a local copy of the result list
    x = x_new[:]
    # For each state (s), iterate through all the actions to determine
    # the maximum satisfaction probability.
    # The list fastIdx stores, for each state, a list (idxs) of lists (idx) of
    # next state indices, each associated to an action (a) available at the
    # state.
    # For each action, solve the inner optimization problem and the pick the
    # highest value (max) across all actions.
    # Note: if the transition is deterministic, i.e., there is only one next
    # state available (len(idx)==1), do not call the inner optimization
    # problem and just assign the satisfaction probability associated
    # to the next state.
    for s, idxs in enumerate(fastIdx):
            x_new[s] = max([x[idx[0]] if len(idx)==1 \
                else self.computeCurrentMax([x[i] for i in idx], par1[s][a]) \
                for a, idx in enumerate(idxs)])

    return x_new
```

Bounded Until Operator

```
# Routine to compute Pmax[phi1 U<=k phi2], i.e., the probability of reaching a
# state satisfying phi2 within k steps while only visiting states
# satisfying phi1.
# Class: Convex-MDP
# Inputs:
# list[int] yesStatesList: list of indices of the precomputed states
    satisfying property phi2
# list[int] maybeStatesList: list of indices of the precomputed states for
    which the satisfaction probability is unknown
# int k: maximum bound on the number of allowable steps
# Outputs:
# list[double] x_new: list of satisfaction probabilities for each state of the
    model
def computeBUntilMax(self, yesStatesList, maybeStatesList, steps):
    # Retrieve the indices of the states reachable in one step for all the
    # states in maybeStatesList (fastIdxUntil) and the information about
    # the convex uncertainty set (par1)
    # Note: fastIdxUntil is just a reduced version of fastIdx where only the
    # lists corresponding to states in maybeStatesList are saved.
    fastIdxUntil, par1 = self.computeUncertaintiesUntil(maybeStatesList)

    # Initialize the result list
    x_new = [0]*self.nStates
    for s in yesStatesList:
        x_new[s] = 1

    # If there is no state in maybeStatesList, return the satisfaction
    # probabilities
    if not maybeStatesList:
        return x_new

    # Initialize the iteration
    step = 0
    # Iterate until the number of states has reached the bound
    while(step < k):
        # Store a local copy of the satisfaction probabilities computed at the
        # previous iteration step
        x = x_new[:]
        # Iterate through all states in maybeStatesList
        for s, idxs in enumerate(fastIdxUntil):
            # For each state (s), iterate through all the actions to
            # determine the maximum satisfaction probability.
            # The list fastIdxUntil stores, for each state, a list (idxs) of
            # lists (idx) of next state indices, each associated to an
            # action (a) available at the state.
            # For each action, solve the inner optimization problem
            # and the pick the highest value (max) across all actions.
            # Note: if the transition is deterministic, i.e., there is only
            # one next state available (len(idx)==1), do not call the inner
```

```
47              # optimization problem and just assign the satisfaction
48              # probability associated to the next state.
49              x_new[maybeStatesList[s]] = max([x[idx[0]] if len(idx)==1 \
50                  else self.computeCurrentMax([x[i] for i in idx], par1[s][a]) \
51                  for a, idx in enumerate(idxs)])
52
53          # Increment the iteration count
54          step += 1
55
56      return x_new
```

Unbounded Until Operator - Value Iteration Procedure

```
# Routine to compute Pmax[phi1 U phi2], i.e., the probability of reaching a
# state satisfying phi2 while only visiting states satisfying phi1.
# Class: Convex-MDP
# Inputs:
# list[int] yesStatesList: list of indices of the precomputed states
    satisfying property phi2
# list[int] maybeStatesList: list of indices of the precomputed states for
    which the satisfaction probability is unknown
# Outputs:
# list[double] x_new: list of satisfaction probabilities for each state of the
    model
def computeUntilMaxVI(self, yesStatesList, maybeStatesList):
    # Retrieve the indices of the states reachable in one step for all the
    # states in maybeStatesList (fastIdxUntil) and the information about
    # the convex uncertainty set (par1)
    # Note: fastIdxUntil is just a reduced version of fastIdx where only the
    # lists corresponding to states in maybeStatesList are saved.
    fastIdxUntil, par1 = self.computeUncertaintiesUntil(maybeStatesList)

    # Initialize the result list
    x_new = [0]*self.nStates
    for s in yesStatesList:
        x_new[s] = 1

    # If there is no state in maybeStatesList, return the satisfaction
    # probabilities
    if not maybeStatesList:
        return x_new

    # Iterate until reaching the fixed point
    delta = 1e-3
    epsilon = 1e-6
    current = -1.
    while(1):
        x = x_new[:]
        # For each state (s), iterate through all the actions to
        # determine the maximum satisfaction probability.
        # The list fastIdxUntil stores, for each state, a list (idxs) of
        # lists (idx) of next state indices, each associated to an
        # action (a) available at the state.
        # For each action, solve the inner optimization problem
        # and the pick the highest value (max) across all actions.
        # Note: if the transition is deterministic, i.e., there is only
        # one next state available (len(idx)==1), do not call the inner
        # optimization problem and just assign the satisfaction
        # probability associated to the next state
        for s, idxs in enumerate(fastIdxUntil):
            x_new[maybeStatesList[s]] = max([x_new[idx[0]] if len(idx)==1 \
             else self.computeCurrentMax([x_new[i] for i in idx], par1[s][a]) \
```

```
47              for a, idx in enumerate(idxs)])
48
49          # Stopping criterion
50          # Relative tolerance condition across all the states in
51          # maybeStatesList
52          # STOP, if no probability in the initial states has changed by more
53          # than epsilon after decreasing delta by 100x
54          # Note: the computation of the absolute value of the difference
55          # between the results of two consecutive iterations is not required,
56          #because satisfaction probabilities increase monotonically
57          if(delta > max([(x_new[s]-x[s])/x_new[s] for s in maybeStatesList])):
58              approx = int(max([x_new[s] for s in self.initState])*1/epsilon)
59              if approx == current:
60                  if done:
61                      break
62                  if not done:
63                      delta /= 10.
64                      done = True
65              else:
66                  current = approx
67                  done = False
68                  delta /= 10.
69
70      return x_new
```

Procedure to solve the inner convex problem for the interval model of uncertainty.
This procedure solve the following convex optimization problem (in the maximization case):

$$\nu_s^a(\mathbf{x}) = \max_{\mathbf{f_s^a} \in \mathcal{F}_s^a} (\mathbf{x^T f_s^a})$$

where $\nu_s^a(\mathbf{x})$ is the candidate satisfaction probability in state s if action a is chosen, $\mathbf{x} \in \mathbb{R}^N$ is a vector containing the current estimations of the satisfaction probabilities of the next-states reachable from s if choice a is taken and $\mathcal{F}_s^a$ is the convex uncertainty set associated to this transition.

If there is no uncertainty in the transition, the set $\mathcal{F}_s^a$ degenerates to a point $\mathbf{f_s^a} \in \mathbb{R}^N$. The solution of the problem can then be trivially computed by a vector-vector multiplication:

$$\nu_s^a(\mathbf{x}) = \mathbf{x^T f_s^a}$$

In the non-degenerated case, the uncertainty set is defined as:

$$\mathcal{F}_s^a = \{\mathbf{f}_s^a \in \mathbb{R}^N \mid \mathbf{0} \leq \underline{\mathbf{f}}_s^a \leq \mathbf{f}_s^a \leq \overline{\mathbf{f}}_s^a \leq \mathbf{1}, \mathbf{1}^T \mathbf{f}_s^a = 1\}$$

i.e., it is a subset of $\mathbb{R}^N$ delimited by the intervals associated to the probabilities of transitioning to the next states, and further intersected with the probability simplex. Such intervals represent a range of possible weights to be associated to the values of the known vector $\mathbf{x}$. In order to maximize the value of $\nu_s^a(\mathbf{x})$, we would like to give maximum weight to the largest element of $\mathbf{x}$. This is not immediately possible, though, because we need first to guarantee that all the lower bounds on the probability to transition to each of the next states get satisfied. After satisfying all lower bounds, we can start processing the elements of $\mathbf{x}$ from the largest to the smallest and assign maximum weight, within the upper bound of each interval, to the elements of $\mathbf{x}$ in descending order of magnitude. This iteration has to stop when the sum of the assigned weights reaches the limit of 1, enforced by the probability simplex.

Finally, we note that, for the minimization case, all the reasoning above remains valid, but the elements of $\mathbf{x}$ are processed in ascending order of magnitude to minimize the value of $\nu_s^a(\mathbf{x})$. The code implementing this procedure is reported below.

```
# Routine to compute: nu=max_{f in F}(f*x)
# where:
# nu is the candidate satisfaction probability of state s if action a is chosen
# F in the interval uncertainty set associated to state s and action a,
# f a realization of uncertainty, and
# x is the vector of (temporarily computed) satisfaction probabilities of the next states reachable from s if action a is chosen

# Class: Interval-MDP
# Inputs:
# list[double] x: list of of (temporarily computed) satisfaction probabilities of the next states reachable from s if action a is chosen
# list[list[double]] F: list of lists of doubles containing lower and upper bounds of the interval ranges
# Outputs:
# double nu: is the candidate satisfaction probability of state s if action a is chosen
def computeCurrentMax(self, x, F):
    # First, check whether the transition is actually uncertain, i.e.,
    # whether F contains lower and upper bounds or not.
    if len(F) == 2:
        # Uncertainty, assign lower/upper bounds to lists iL/iU
        iL = F[0]
        iU = F[1]
    else:
        # No uncertainty, just perform a vector-vector multiplication
        return sum([v[0]*v[1] for v in zip(x, F[0])])

    # Beginning of the convex optimization
    # Sort the indices of the elements of vector x from the one pointing to
    # the largest element to the one pointing to the smallest.
    # Note: set "reverse = False" to solve the minimization problem
    idxSort = [v[0] for v in sorted(enumerate(x), key=lambda v:v[1], reverse = True)]

    # Make sure that the lower bounds in all the transition probabilities
    # get satisfied
    nu = sum(map(lambda v, w: v*w, x, iL))
    # totP stores how much weight we can still assign
    # to the elements of x after all lower bounds
    # are satisfied
    totP = 1. - sum(iL)

    # Process the elements of x in descending order of magnitude
    for i in idxSort:
      # Compute the size of the interval
        delta = iU[i] - iL[i]
        # If the size of the interval is smaller
        # then the weight that is still available...
```

```
45          if(delta<totP):
46              # ...assign maximum weight within
47              # the upper bound of the interval to the current element
48              # of x, and update the value of the remaining
49              # weight...
50              nu += delta*x[i]
51              totP -= delta
52          else:
53              # ...otherwise, assign all the weight that is
54              # left to the current element of x
55              # and break the loop
56              nu += totP*x[i]
57              break
58
59      return nu
60
```

Procedure to solve the inner convex problem for the ellipsoidal model of uncertainty.
We aim again to solve the following convex optimization problem (in the maximization case):

$$\nu_s^a = \max_{\mathbf{f_s^a} \in \mathcal{F}_s^a} (\mathbf{x^T f_s^a}) \tag{C.1}$$

where the uncertainty set $\mathcal{F}_s^a$ is obtained by intersecting the probability simplex with the ellipsoidal region defined in Equation 2.5. In particular, we rewrite Problem (C.1) in canonical primal form:

$$\begin{aligned} \nu_s^a = \max_{\mathbf{f_s^a}} \; & \mathbf{x}^T \mathbf{f_s^a} \\ \text{s.t. } & \mathbf{1}^T \mathbf{f_s^a} = 1 \\ & \sum_{s'} \frac{(f_{ss'} - h_{ss'})^2}{h_{ss'}} \leq (\mathcal{K}_s^a)^2 \\ & \mathbf{f_s^a} \geq \mathbf{0} \end{aligned} \tag{C.2}$$

We report now a dual formulation of Problem (C.2) that we experimentally found to be solvable in a shorter time than the primal formulation. Intuitively, the faster performance is achieved because this formulation allows to write the analytical expression of the dual solution d_s^a of the inner problem as a function of the state probabilities $\mathbf{x}$. The analytical expression can then be rapidly evaluated using the known values of vector $\mathbf{x}$. This material is inspired by results by Nilim and El Ghaoui [125]. In the following, we drop the superscript a and subscript s to increase readability.

The Lagrangian operator associated to Problem (C.2) reads:

$$\mathcal{L}(\mathbf{x}, \mathbf{f}, \mu, \boldsymbol{\xi}, \eta) = \mathbf{x}^T \mathbf{f} + \mu(1 - \mathbf{1}^T \mathbf{f}) + \boldsymbol{\xi}^T \mathbf{f} + \eta \left(\mathcal{K}^2 - \sum_{s'} \frac{(f_{s'} - h_{s'})^2}{h_{s'}} \right) \tag{C.3}$$

The primal optimal value can be computed as:

$$\nu(\mathbf{x}) = \max_{\mathbf{f}} \min_{\mu, \boldsymbol{\xi} \geq \mathbf{0}, \eta \geq 0} \mathcal{L}(\mathbf{x}, \mathbf{f}, \mu, \boldsymbol{\xi}, \eta) \tag{C.4}$$

By the *minimax* theorem [30], we obtain an upper bound on the value of $\nu(\mathbf{x})$ by inverting the "max" and "min" operators:

$$d(\mathbf{x}) = \min_{\mu, \boldsymbol{\xi} \geq \mathbf{0}, \eta \geq 0} \max_{\mathbf{f}} \mathcal{L}(\mathbf{x}, \mathbf{f}, \mu, \boldsymbol{\xi}, \eta) \tag{C.5}$$

Problem (C.2) satisfies Slater's condition [30] for any non-trivial uncertainty set, so strong duality holds and $\nu(\mathbf{x}) = d(\mathbf{x})$. We can thus solve Problem (C.5) instead of Problem (C.2) while preserving the soundness and completeness of the model-checking procedure. As a first step, we compute the dual function $g(\mathbf{x}, \mu, \boldsymbol{\xi}, \eta)$ by solving the inner problem in Equation (C.5), i.e., we aim to compute:

$$g(\mathbf{x}, \mu, \boldsymbol{\xi}, \eta) = \max_{\mathbf{f}} \mathcal{L}(\mathbf{x}, \mathbf{f}, \mu, \boldsymbol{\xi}, \eta) \tag{C.6}$$

We can solve Problem (C.6) by setting the gradient of the Lagrangian to zero, and solving for the optimal primal solution $\mathbf{f}^*$:

$$\begin{cases} \frac{\partial \mathcal{L}}{\partial f_{s_0}} = & x_0 - \mu + \xi_0 - \frac{2\eta}{h_{s_0}}(f_{s_0} - h_{s_0}) & = 0 & & f^*_{s_0} = & \frac{h_{s_0}}{2\eta}(x_0 - \mu + \xi_0) + h_{s_0} \\ \frac{\partial \mathcal{L}}{\partial f_{s_1}} = & x_1 - \mu + \xi_1 - \frac{2\eta}{h_{s_1}}(f_{s_1} - h_{s_1}) & = 0 & \Rightarrow & f^*_{s_1} = & \frac{h_{s_1}}{2\eta}(x_1 - \mu + \xi_1) + h_{s_1} \\ \dots & & & & \dots & \\ \frac{\partial \mathcal{L}}{\partial f_{s_N}} = & x_N - \mu + \xi_N - \frac{2\eta}{h_{s_N}}(f_{s_N} - h_{s_N}) & = 0 & & f^*_{s_N} = & \frac{h_{s_N}}{2\eta}(x_N - \mu + \xi_N) + h_{s_N} \end{cases} \tag{C.7}$$

Substituting $\mathbf{f}^*$ back into Problem C.6, we obtain the dual function:

$$g(\mathbf{x}, \mu, \boldsymbol{\xi} \geq 0, \eta \geq 0) = \mu + \eta\mathcal{K}^2 + \sum_{s'}(h_{s'}(x_{s'} - \mu + \xi_{s'})) + \frac{1}{4\eta}\sum_{s'}(h_{s'}(x_{s'} - \mu + \xi_{s'})^2) \tag{C.8}$$

We can now compute d solving the dual problem:

$$d(\mathbf{x}) = \min_{\mu, \boldsymbol{\xi} \geq \mathbf{0}, \eta \geq 0} g(\mathbf{x}, \mu, \boldsymbol{\xi}, \eta) \tag{C.9}$$

Before further proceeding, we note that, for monotonicity reasons, we can trivially set $\boldsymbol{\xi}^* = 0$. We thus aim to solve the following optimization problem:

$$d(\mathbf{x}) = \min_{\mu, \eta \geq 0} g(\mathbf{x}, \mu, \eta) = \min_{\mu, \eta \geq 0} \mu + \eta\mathcal{K}^2 + \sum_{s'}(h_{s'}(x_{s'} - \mu)) + \frac{1}{4\eta}\sum_{s'}(h_{s'}(x_{s'} - \mu)^2) \tag{C.10}$$

We first set the partial derivatives of the dual function g to zero and compute the optimal dual solution (μ^*, η^*). Formally:

$$\begin{cases} \frac{\partial g}{\partial \mu} = & 1 - \frac{2\mathbf{h}}{4\eta}(\mathbf{x} - \mu\mathbf{1})^T - \mathbf{1}^T\mathbf{h} & = 0 & & \mu^*(\mathbf{x}) = & \sum_{s'} h_{s'}x_{s'} \\ & & & \Rightarrow & & \\ \frac{\partial g}{\partial \eta} = & \mathcal{K}^2 - \frac{\mathbf{h}(\mathbf{x} - \mu\mathbf{1})^{2T}}{4\eta^2} & = 0 & & \eta^*(\mathbf{x}) = & \frac{1}{2\mathcal{K}}\sqrt{\sum_{s'} h_{s'}(x_{s'} - \mu)^2} \end{cases} \tag{C.11}$$

The optimal value can then be computed as:

$$d(\mathbf{x}) = g(\mu^*(\mathbf{x}), \eta^*(\mathbf{x}))$$

Finally, we report the primal and dual formulations to solve the analogous minimization problem. In the primal Problem (C.2), we change the optimization operator from "max" to "min":

$$\begin{aligned} \nu^a_s = \min_{\mathbf{f^a_s}} \; & \mathbf{x}^T\mathbf{f^a_s} \\ \text{s.t. } & \mathbf{1}^T\mathbf{f^a_s} = 1 \\ & \sum_{s'} \frac{(f_{ss'} - h_{ss'})^2}{h_{ss'}} \leq (\mathcal{K}^a_s)^2 \\ & \mathbf{f^a_s} \geq \mathbf{0} \end{aligned}$$

Following the same steps presented above, we obtain the corresponding dual problem (again we dropped subscripts and superscripts):

$$d(\mathbf{x}) = \min_{\mu,\eta\geq 0} g(\mathbf{x},\mu,\eta) = \max_{\mu,\eta\geq 0} \mu - \eta\mathcal{K}^2 + \sum_{s'}(h_{s'}(x_{s'}-\mu)) - \frac{1}{4\eta}\sum_{s'}(h_{s'}(x_{s'}-\mu)^2)$$

which admits the optimal solution:

$$\begin{cases} \frac{\partial g}{\partial \mu} = & 1 + \frac{2\mathbf{h}}{4\eta}(\mathbf{x}-\mu\mathbf{1})^T - \mathbf{1}^T\mathbf{h} & = 0 \\ \frac{\partial g}{\partial \eta} = & -\mathcal{K}^2 + \frac{\mathbf{h}(\mathbf{x}-\mu\mathbf{1})^{2T}}{4\eta^2} & = 0 \end{cases} \Rightarrow \begin{cases} \mu^*(\mathbf{x}) = & \sum_{s'} h_{s'}x_{s'} \\ \eta^*(\mathbf{x}) = & \frac{1}{2\mathcal{K}}\sqrt{\sum_{s'} h_{s'}(x_{s'}-\mu)^2} \end{cases}$$

And, again:

$$d(\mathbf{x}) = g(\mu^*(\mathbf{x}),\eta^*(\mathbf{x}))$$

In the following, we report the Python routines used to solve the problems formulated above, both for the maximization and minimization cases.

```
# Routine to compute: d=g(mu^{*}, eta^{*})

# Class: Ellipsoidal-MDP
# Inputs:
# list[double] x: list of of (temporarily computed) satisfaction probabilities
    of the next states reachable from s if action a is chosen
# list[list[double]] F: list of lists of doubles containing the experimental
    transition frequencies and the uncertainty parameter K
# Outputs:
# double d: (by duality equal to nu) is the candidate satisfaction probability
    of state s if action a is chosen
def computeCurrentMax(self, x, F):
    # First, check whether the transition is actually uncertain, i.e.,
    # whether F contains lower and upper bounds or not.
    if len(F) == 2:
        # Uncertainty, assign experimental transition frequencies
        # to list h and the uncertainty parameter K
        h = F[0]
        K = F[1][0]
    else:
        # No uncertainty, just perform a vector-vector multiplication
        return sum([v[0]*v[1] for v in zip(x, F[0])])

    # Determine the index of the element of x with the highest
    # magnitude
    idx = max(xrange(len(v)), key=v.__getitem__)
    Ks = K*K
    # Perform an initial sanity check to determine
    # whether the quadratic constraint is binding
    # or not. If such a constraint is not binding
    # just give maximum weight to the element of
    # x with highest magnitude
    hi = h[idx]
    ihi = 1.-hi
    if ihi+ihi*ihi/hi <= Ks:
        # Quadratic constraint is not binding
        return x[idx]

    # Computation of the optimal value of mu
    mu = 0.
    for i, hi in enumerate(h):
        mu += hi*x[i]
    # Computation of the term under the square
    # root sign in the expression of the optimal eta
    s = 0.
    for i, hi in enumerate(h):
        inn = x[i] - mu
        s += hi*inn*inn
    # If the optimal eta is null, just give all the weight
    # to the element of x with highest value
```

```
48      if not s:
49          return x[idx]
50      # Computation of the optimal value of eta
51      eta = 1./(2.*K)*(s**0.5)
52
53      # Evaluation of the optimal value of function g
54      return eta*Ks+mu+s/(4*eta)
```

```
# Routine to compute: d=g(mu^{*}, eta^{*})

# Class: Ellipsoidal-MDP
# Inputs:
# list[double] x: list of of (temporarily computed) satisfaction probabilities
    of the next states reachable from s if action a is chosen
# list[list[double]] F: list of lists of doubles containing the experimental
    transition frequencies and the uncertainty parameter K
# Outputs:
# double d: (by duality equal to nu) is the candidate satisfaction probability
    of state s if action a is chosen
def computeCurrentMin(self, v, N):
# First, check whether the transition is actually uncertain, i.e.,
    # whether F contains lower and upper bounds or not.
    if len(F) == 2:
        # Uncertainty, assign experimental transition frequencies
        # to list h and the uncertainty parameter K
        h = F[0]
        K = F[1][0]
    else:
        # No uncertainty, just perform a vector-vector multiplication
        return sum([v[0]*v[1] for v in zip(x, F[0])])

    # Determine the index of the element of x with the lowest
    # magnitude
    idx = min(xrange(len(v)), key=v.__getitem__)
    Ks = K*K
    # Perform an initial sanity check to determine
    # whether the quadratic constraint is binding
    # or not. If such a constraint is not binding
    # just give maximum weight to the element of
    # x with highest magnitude
    hi = h[idx]
    ihi = 1.-hi
    if ihi+ihi*ihi/hi <= Ks:
        # Quadratic constraint is not binding
        return x[idx]

    # Computation of the optimal value of mu
    mu = 0.
    for i, hi in enumerate(h):
        mu += hi*x[i]
    # Computation of the term under the square
    # root sign in the expression of the optimal eta
    s = 0.
    for i, hi in enumerate(h):
        inn = x[i] - mu
        s += hi*inn*inn
    # If the optimal eta is null, just give all the weight
    # to the element of x with highest value
```

```
48      if not s:
49          return x[idx]
50      # Computation of the optimal value of eta
51      eta = 1./(2.*K)*(s**0.5)
52
53      # Evaluation of the optimal value of function g
54      return -nu*ks+mu-s/(4*nu)
```

Finally, we present the routines to generate the Convex Programming formulations of Problem (4.18) to model check the *Unbounded Until* operator, both for the interval and the ellipsoidal models of uncertainty. The generated convex programs can the be solved using MOSEK [122].

```
# Routine to generate the CP formulation of the convex problem to model-check
# the Unbounded Until operator, i.e., to compute the satisfaction
# probabilities Pmax/min[phi1 U phi2]

# Class: Interval-MDP
# Inputs:
# list[int] yesStatesList: list of indices of the precomputed states satisfying property phi2
# list[int] maybeStatesList: list of indices of the precomputed states for which the satisfaction probability is unknown
# list[int] noStatesList: list of indices of the precomputed states for which the satisfaction probability is 0
# boolean MIN: generates the formulation to compute the minimum (maximum) satisfaction probabilities if true (false)
# Outputs:
# list[double] x_new: list of satisfaction probabilities for each state of the model
def computeCP(self, yesStatesList, maybeStatesList, noStatesList, MIN):
    # Retrieve the indices of the states reachable in one step for all the
    # states in maybeStatesList (fastIdxUntil) and the information about
    # the convex uncertainty set (par1)
    # Note: fastIdxUntil is just a reduced version of fastIdx where only the
    # lists corresponding to states in maybeStatesList are saved.
    fastIdxUntil, par1 = self.computeUncertaintiesUntil(maybeStatesList)

    # Initialize the result list
    x_new = [0]*self.nStates
    for s in yesStatesList:
        x_new[s] = 1

    # If there is no state in maybeStatesList, return the satisfaction
    # probabilities
    if not maybeStatesList:
        return x_new

    # Write the cost function min/max the sum of the satisfaction probabilities
    # of the states in the maybeStatesList
    objective = '[objective \%s \'obj\']\n'\% \
       ('maximize ' if MIN else 'minimize ')
    objective += 'x'.join(map(lambda s: 'x'+str(s), maybeStatesList)) + '\n'
    objective += '[/objective]\n'

    # Declare the variables x containing the satisfaction probabilities
    # of the states in the maybeStatesList
    allVariables = '[variables]\n'
```

```
41      allVariables += ' '.join(map(lambda s: 'x'+str(s), maybeStatesList))
42
43      # Initialize the strings containing the constraints and variable bound
44      # declarations.
45      # For the variable bounds, we will first set all variables to be positive
46      # and then overwrite this command to set the required variables to be
47      # unbounded
48      constraints = '[constraints]\n'
49      bounds = '[bounds]\n'
50      bounds += '[b] 0 <= * [/b]\n'
51
52      # Declare the list containing the unbounded variables
53      frees = []
54      # Initialize the counters to store the model statistics
55      # Passing these values to MOSEK help the solver to allocate memory
56      # more efficiently.
57      varCount = 0 # Number of variables
58      consCount = 0 # Number of constraints
59      ANZCount = 0 # Number of non-zeros in the problem tableau
60      # Start writing the constraints
61      # Iterate through all the states s in maybeStatesList
62      for s, idxs in enumerate(fastIdxUntil):
63          # Iterate through each action a associated to the state
64          for a, idx in enumerate(idxs):
65              # Process certain transitions
66              if len(par1[s][a]) == 1:
67                  # Write constraint x_s - f*x <= 0
68                  # or  x_s - f*x >= 0
69                  # for the min/max problem, respectively
70                  variables = ''
71                  RHS = 0.
72                  variables += 'x' + str(maybeStatesList[s])
73                  ANZCount += 1
74                  # Process one transition to a next state per time
75                  for inext, next in enumerate(idx):
76                      if next in noStatesList:
77                          # Nothing to be done, since x_s=0 if s in noStatesList
78                          continue
79                      if next in yesStatesList:
80                          # Just modify the value of the RHS, since x_s=1
81                          # if s in yesStatesList
82                          RHS += par1[s][a][0][inext]
83                      else:
84                          # Add the term f_ss' * x to the LHS
85                          variables += str(-par1[s][a][0][inext]) \
86                           + ' x' + str(next)
87                          ANZCount += 1
88                  # Add the constraint and increment counter
89                  constraints += '\t[con \'c\%d\'] '\%(consCount) + variables \
90                   + (' <= ' if MIN else ' >= ') + str(RHS) + ' [/con]\n'
```

```
91                  consCount += 1
92              else:
93                  # Process uncertain transition
94                  # First constraint in the form
95                  # x_s - lambda_1 - iL * lambda_2 + iU * lambda_3 <= 0
96                  # or x_s - lambda_1 + iL * lambda_2 - iU * lambda_3 >= 0
97                  # for the min/max problem, respectively
98                  variables = ''
99                  variables += 'x' + str(maybeStatesList[s])
100                 # Declare lambda_1 as unbounded variable
101                 frees.append('l1_' + str(maybeStatesList[s]) + '_' + str(a))
102                 variables += '-' + frees[-1] + ' '
103                 allVariables += ' ' + frees[-1]
104                 varCount += 1
105                 # Process one lower bound per time
106                 for inext, low in enumerate(par1[s][a][0]):
107                     varCount += 1
108                     variables += (str(-low) if MIN else '+'+str(low)) + \
109                      ' l2_' + str(maybeStatesList[s]) + \
110                      '_' + str(a) + '_' + str(inext) + ' '
111                     allVariables += ' ' + ' l2_' + str(maybeStatesList[s]) + \
112                      '_' + str(a) + '_' + str(inext)
113                 # Process one upper bound per time
114                 for inext, up in enumerate(par1[s][a][1]):
115                     varCount += 1
116                     variables += ('+'+str(up) if MIN else str(-up)) + \
117                      ' l3_' + str(maybeStatesList[s]) + \
118                      '_' + str(a) + '_' + str(inext) + ' '
119                     allVariables += ' ' + ' l3_' + str(maybeStatesList[s]) + \
120                      '_' + str(a) + '_' + str(inext)
121                 # Write the constraint and update the model statistics
122                 constraints += '\t[con \'c\%d\'] '\%(consCount) + variables \
123                  + (' <= ' if MIN else ' >= ') + '0 [/con]\n'
124                 ANZCount += 2 + 2*len(par1[s][a][0])
125                 consCount += 1
126                 # Second constraint in the form
127                 # x_s - lambda_2 + lambda_3 - lambda_1 = 0
128                 # or x_s + lambda_2 - lambda_3 - lambda_1 = 0
129                 # for the min/max problem, respectively
130                 # Add one constraint for each transition to a next state
131                 for inext, next in enumerate(idx):
132                     variables = ''
133                     RHS = '0'
134                     if next in noStatesList:
135                         # Nothing to be done, since x_s=0 if s in noStatesList
136                         pass
137                     elif next in yesStatesList:
138                         # Just modify the value of the RHS, since x_s=1
139                         # if s in yesStatesList
140                         RHS = '-1'
```

```
                    else:
                        # Add the term x_s' to the LHS
                        variables = 'x' + str(next)
                        ANZCount += 1
                    # Add the rest of the variables to the LHS
                    variables += ' -' + frees[-1] + (' +13_' if MIN else \
                     ' -13_') + str(maybeStatesList[s]) + '_' + str(a) + '_' \
                      + str(inext) + (' -12_' if MIN else ' +12_') + \
                     str(maybeStatesList[s])+'_'+str(a) +'_'+ str(inext) + ' '
                    # Add the constraint
                    constraints += '\t[con \'c\%d\'] '\%(consCount) \
                     + variables + '=' + RHS + ' [/con]\n'
                    consCount += 1
                    ANZCount += 3

    # If there are unbounded variables, add them to the bound declarations
    if frees:
        bounds += '[b] ' + ','.join(frees) + ' free [/b]\n'

    # Terminate the string containing the variable, constraint and bound
    # declarations
    allVariables += '[/variables]\n'
    constraints += '[/constraints]\n'
    bounds += '[/bounds]\n'

    # Write all the string containing the model statistics
    hints = '[hints]\n'
    hints += '\t[hint NUMVAR] \%d [/hint]\n'\%(varCount+len(maybeStatesList))
    hints += '\t[hint NUMCON] \%d [/hint]\n'\%(consCount)
    hints += '\t[hint NUMANZ] \%d [/hint]\n'\%(ANZCount)
    hints += '[/hints]\n'

    # Write the file containing the problem formulation
    f = open('LP.opf', 'w')
    f.write(hints)
    f.write(allVariables)
    f.write(objective)
    f.write(constraints)
    f.write(bounds)
    f.close()

    # Call MOSEK to solve the convex problem
    call(['mosek', 'LP.opf'])

    # Read the solution file and retrieve the optimal solution
    r = open('LP.sol', 'r')
    ss = r.read()
    res = re.findall('x[0-9]+[ \t]+[A-Z]+[ \t]+[0-9]\.[0-9]+e[+-][0-9]+', ss)
    r.close()
    for r in res:
```

```
191             sliced = r.split()
192             x_new[int(sliced[0][1:])] = float(sliced[-1])
193 
194     return x_new
```

```
# Routine to generate the CP formulation of the convex problem to model-check
# the Unbounded Until operator, i.e., to compute the satisfaction
# probabilities Pmax/min[phi1 U phi2]

# Class: Ellipsoidal-MDP
# Inputs:
# list[int] yesStatesList: list of indices of the precomputed states
    satisfying property phi2
# list[int] maybeStatesList: list of indices of the precomputed states for
    which the satisfaction probability is unknown
# list[int] noStatesList: list of indices of the precomputed states for which
    the satisfaction probability is 0
# boolean MIN: generates the formulation to compute the minimum (maximum)
    satisfaction probabilities if true (false)
# Outputs:
# list[double] x_new: list of satisfaction probabilities for each state of the
    model
def computeCP(self, yesStatesList, maybeStatesList, noStatesList, MIN):
    # Retrieve the indices of the states reachable in one step for all the
    # states in maybeStatesList (fastIdxUntil) and the information about
    # the convex uncertainty set (par1)
    # Note: fastIdxUntil is just a reduced version of fastIdx where only the
    # lists corresponding to states in maybeStatesList are saved.
    fastIdxUntil, par1 = self.computeUncertaintiesUntil(maybeStatesList)

    # Initialize the result list
    x_new = [0]*self.nStates
    for s in yesStatesList:
        x_new[s] = 1

    # If there is no state in maybeStatesList, return the satisfaction
    # probabilities
    if not maybeStatesList:
        return x_new

    # Write the cost function min/max the sum of the satisfaction
    probabilities
    # of the states in the maybeStatesList
    objective = '[objective \%s \'obj\']\n'\% \
       ('maximize ' if MIN else 'minimize ')
    objective += '+'.join(map(lambda s: 'x'+str(s), maybeStatesList)) + '\n'
    objective += '[/objective]\n'

    # Declare the variables x containing the satisfaction probabilities
    # of the states in the maybeStatesList
    allVariables = '[variables]\n'
    allVariables += ' '.join(map(lambda s: 'x'+str(s), maybeStatesList))

    # Initialize the strings containing the constraints and variable bound
    # declarations.
```

```
45      # For the variable bounds, we will first set all variables to be positive
46      # and then overwrite this command to set the required variables to be
47      # unbounded
48      constraints = '[constraints]\n'
49      bounds = '[bounds]\n'
50      bounds += '[b] 0 <= * [/b]\n'
51
52      # Declare the list containing the unbounded variables
53      frees = []
54      # Initialize the counters to store the model statistics
55      # Passing these values to MOSEK help the solver to allocate memory
56      # more efficiently.
57      varCount = 0 # Number of variables
58      consCount = 0 # Number of constraints
59      ANZCount = 0 # Number of non-zeros in the problem tableau
60      # Start writing the constraints
61      # Iterate through all the states s in maybeStatesList
62      for s, idxs in enumerate(fastIdxUntil):
63          # Iterate through each action a associated to the state
64          for a, idx in enumerate(idxs):
65              # Process certain transitions
66              if len(par1[s][a]) == 1:
67                  # Write constraint x_s - f*x <= 0
68                  # or  x_s - f*x >= 0
69                  # for the min/max problem, respectively
70                  variables = ''
71                  RHS = 0.
72                  variables += 'x' + str(maybeStatesList[s])
73                  ANZCount += 1
74                  # Process one transition to a next state per time
75                  for inext, next in enumerate(idx):
76                      if next in noStatesList:
77                          # Nothing to be done, since x_s=0 if s in noStatesList
78                          continue
79                      if next in yesStatesList:
80                          # Just modify the value of the RHS, since x_s=1
81                          # if s in yesStatesList
82                          RHS += par1[s][a][0][inext]
83                      else:
84                          # Add the term f_ss' * x to the LHS
85                          variables += str(-par1[s][a][0][inext]) \
86                           + ' x' + str(next)
87                          ANZCount += 1
88                  # Add the constraint and increment counter
89                  constraints += '\t[con \'c\%d\'] '\%(consCount) + variables \
90                   + (' <= ' if MIN else ' >= ') + str(RHS) + ' [/con]\n'
91                  consCount += 1
92              else:
93                  # Process uncertain transition
94                  # First constraint in the form
```

```
95                  # x_s - lambda_1 +  lambda_2 + h * E * lambda_3 <= 0
96                  # or x_s - lambda_1 - lambda_2 - h * E * lambda_3 >= 0
97                  # for the min/max problem, respectively
98                  variables = ''
99                  variables += 'x' + str(maybeStatesList[s])
100                 l1 = 'l1_' + str(maybeStatesList[s]) + '_' + str(a)
101                 # Declare lambda_1 as unbounded variable
102                 frees.append(l1)
103                 variables += '-' + l1 + ' '
104                 allVariables += ' ' + l1
105                 varCount += 1
106                 l2 = 'l2_' + str(maybeStatesList[s]) + '_' + str(a)
107                 variables += (' +' if MIN else ' -') + l2
108                 allVariables += ' ' + l2
109                 varCount += 1
110                 # Compute the inverse of the uncertainty parameter K
111                 # for convenience
112                 oneoverks = 1./par1[s][a][1][0]
113                 # Process one transition to a next state per time
114                 for inext, next in enumerate(par1[s][a][0]):
115                     l3 = 'l3_' + str(maybeStatesList[s]) + '_' + str(a) + \
116                      '_' + str(inext)
117                     variables += ('+' if MIN else '-') \
118                      + str(oneoverks*next**0.5) + ' ' + l3 + ' '
119                     allVariables += ' ' + l3
120                     # Declare lambda_3 as unbounded variable
121                     frees.append(l3)
122                 # Update model statistics and write constraint
123                 varCount += len(par1[s][a][0])
124                 ANZCount += 3 + len(par1[s][a][0])
125                 constraints += '\t[con \'c\%d\'] '\%(consCount) +variables+ \
126                  (' <= ' if MIN else ' >= ') + '0 [/con]\n'
127                 consCount += 1
128                 # Second constraint in the form
129                 # x_s - lambda_1 + E * lambda_3 = 0
130                 # or x_s - lambda_1 - E * lambda_3 = 0
131                 # for the min/max problem, respectively
132                 # Add one constraint for each transition to a next state
133                 for inext, next in enumerate(idx):
134                     variables = ''
135                     RHS = '0'
136                     if next in noStatesList:
137                         # Nothing to be done, since x_s=0 if s in noStatesList
138                         pass
139                     elif next in yesStatesList:
140                         # Just modify the value of the RHS, since x_s=1
141                         # if s in yesStatesList
142                         RHS = '-1'
143                     else:
144                         # Add the term x_s' to the LHS
```

```
145                         variables = 'x' + str(next)
146                         ANZCount += 1
147                     # Add the rest of the variables to the LHS
148                     variables += ' -' + l1 + (' +' if MIN else ' -') + \
149                      str(oneoverks/par1[s][a][0][inext]**0.5) + \
150                      ' l3_' + str(maybeStatesList[s]) +'_'+ str(a) +'_'  \
151                      + str(inext)
152                     # Add the constraint
153                     constraints += '\t[con \'c\%d\'] '\%(consCount) +
154                      variables + '=' + RHS + ' [/con]\n'
155                     consCount += 1
156                     ANZCount += 2
157                 # Quadratic conic constraint
158                 # Adding this constraint as a conic bound substantially
159                 # simplifies the solution of the SOCP
160                 # Constraint in the form norm(lambda_3) <= lambda_2
161                 bounds += '[cone quad] ' + l2 + ',' + \
162                     ','.join(['l3_' + str(maybeStatesList[s]) +'_'+ str(a) + \
163                     '_'+str(inext) for inext in range(len(par1[s][a][0]))]) \
164                      + ' [/cone]\n'
165
166     # If there are unbounded variables, add them to the bound declarations
167     if frees:
168         bounds += '[b] ' + ','.join(frees) + ' free [/b]\n'
169
170     # Terminate the string containing the variable, constraint and bound
171     # declarations
172     allVariables += '[/variables]\n'
173     constraints += '[/constraints]\n'
174     bounds += '[/bounds]\n'
175
176     # Write all the string containing the model statistics
177     hints = '[hints]\n'
178     hints += '\t[hint NUMVAR] \%d [/hint]\n'\%(varCount+len(maybeStatesList))
179     hints += '\t[hint NUMCON] \%d [/hint]\n'\%(consCount)
180     hints += '\t[hint NUMANZ] \%d [/hint]\n'\%(ANZCount)
181     hints += '[/hints]\n'
182
183     # Write the file containing the problem formulation
184     f = open('SOCP.opf', 'w')
185     f.write(hints)
186     f.write(allVariables)
187     f.write(objective)
188     f.write(constraints)
189     f.write(bounds)
190     f.close()
191
192     # Call MOSEK to solve the convex problem
193     call(['mosek', 'SOCP.opf'])
194
```

```
195     # Read the solution file and retrieve the optimal solution
196     r = open('SOCP.sol', 'r')
197     ss = r.read()
198     res = re.findall('x[0-9]+[ \t]+[A-Z]+[ \t]+[0-9]\.[0-9]+e[+-][0-9]+', ss)
199     r.close()
200     for r in res:
201         sliced = r.split()
202         x_new[int(sliced[0][1:])] = float(sliced[-1])
203 
204     return x_new
```

Appendix D

Generation of the MIQCP Formulation of the Strategy Synthesis Problem in Python

In this appendix, we first report the Python code used to formulate Problem (6.11) starting from the model data of an Ellipsoidal-MDP. We then list the full formulation of the MIQCP problem generated to synthesize the optimal control strategy for the Ellipsoidal-MDP in Figure 2.5.

```
# Routine to generate the MIQCP formulation of the optimization engine
# in the first iteration of the strategy synthesis algorithm for the
# ellipsoidal model of uncertainty

# Class: Ellipsoidal-MDP
# Inputs:
# boolean MIN: generates the formulation to compute the minimum (maximum)
    satisfaction probabilities if true (false)
# rewardName: name of the reward structure to compute the total expected
    reward on
# Outputs:
# GRB.model model: Gurobi model of the MIQCP optimization problem
def generateCoreProblem(self, MIN, rewardName):
        # Retrieve the indices of the states reachable in one step for all the
        # states in maybeStatesList (fastIdx), the information about
        # the convex uncertainty set (par1), and the reward structure for
        # state rewards (rewStates) and
        # action rewards (rewTrans)
        # For more details on the underlying data-structures, see
        # Appendix C
        fastIdx, par1, rewStates, rewTrans = \
         self.computeUncertaintiesAndRewards(rewardName)

        # Instantiate an empty Gurobi model
        model = Model("qcp")
        # Create a dictionary to store the total expected reward variables,
        # i.e., variables x in the formulation in Chapter 6
        rewVars = {}
```

```
27          # Associate a variable to each state
28          for i in range(self.nStates):
29              rewVars['x'+str(i)] = model.addVar(name='x'+str(i))
30          initVars = []
31          # Explicitly enumerate the reward variables associated
32          # to the initial states
33          for i in self.initState:
34              initVars.append('x'+str(i))
35
36          # Store the declared variables in the model
37          # (required by the Gurobi API)
38          model.update()
39
40          # Create dictionaries for the slack, dual and integer variables
41          slackVars = {}
42          dualVars = {}
43          intVars = {}
44
45          # Write objective function
46          # min/max  of the sum of the reward variables
47          # across the initial states
48          obj = quicksum([rewVars[key] for key in initVars])
49          model.setObjective(obj, GRB.MINIMIZE if MIN else GRB.MAXIMIZE)
50
51          # Initialize data structures to collect model statistics
52          ic = 0 # number of integer constraints
53          ics = [] # names of the integer constraint
54          lc = 0 # number of linear constraints
55          lcs = [] # names of the linear constraints
56          qc = 0 # number of quadratic constraints
57          qcs = [] # names of the quadratic constraints names
58          frees = [] # Unbounded variables
59
60          # Compute  "big'' B
61          # Retrieve the reward values associated to all transitions
62          flat = []
63          map(flat.extend, [self.fastTranRewards[i_idx][i_i] for i_idx, idx in \
64           enumerate(self.fastIdx) for i_i, i in enumerate(idx)])
65          maxTranReward = max(flat)
66          self.bigB = self.nStates*(max(rewStates)+maxTranReward)
67
68          # Determine the absorbing states, i.e., states with only one action
69          # and a self-loop transition of probability 1
70          absorbing = [s for s, idxs in enumerate(fastIdx) if len(idxs) == 1 \
71           and len(idxs[0]) == 1 and idxs[0][0] == s]
72          # Enforce the reward in these states to be zero
73          # (these constraints help the convergence of the numerical solver
74          for s in absorbing:
75              model.addConstr(rewVars['x'+str(s)] == 0, 'lc'+str(lc))
76              lc += 1
```

```
77
78          # Write constraints
79          # Process one state at a time
80          for s, idxs in enumerate(fastIdx):
81              # If s is absorbing, nothing to be done
82              if s in absorbing:
83                  continue
84              # States with multiple actions associated are
85              # treated differently. The first step is thus
86              # to determine if the state has multiple actions
87              # associated to it or not
88              multActions = True
89              if len(idxs) == 1:
90              # only one action
91                   # No extra constraint to add
92                   multActions = False
93
94              # Processing states with multiple actions
95              # Add extra constraints specific to states
96              # with multiple actions
97              else:
98                  # Store the suffix used to uniquify variable names
99                  sufs = []
100                 # Iterate through all actions associated to a state
101                 for a in range(len(idxs)):
102                     suf = str(s) + '_' + str(a)
103                     # Add the slack and integer variables to the model
104                     slackVars['l' + suf] = model.addVar(name='l' + suf)
105                     slackVars['n' + suf] = model.addVar(name='n' + suf)
106                     intVars['z' + suf] = model.addVar(vtype=GRB.BINARY, \
107                      name='z' + suf)
108                     sufs.append(suf)
109                 model.update()
110                 # Add the integer constraint to enforce only one action to be
111                 # selected among those available
112                 model.addConstr(quicksum([intVars['z'+suf] for suf in sufs]) \
113                  == len(idxs) - 1, 'ic'+str(ic))
114                 ics.append('ic'+str(ic))
115                 ic += 1
116                 # Iterate through all the slack variables and add
117                 # the constraints to set the slacks to zero if the
118                 # action is selected
119                 for suf in sufs:
120                     model.addConstr(slackVars['n' + suf] - \
121                      self.bigB*intVars['z'+suf] <= 0, 'ic'+str(ic))
122                     ics.append('ic'+str(ic))
123                     ic += 1
124                     model.addConstr(slackVars['l' + suf] - \
125                      self.bigB*intVars['z'+suf] <= 0, 'ic'+str(ic))
126                     ics.append('ic'+str(ic))
```

```
                    ic += 1

            # Now add constraints to compute the total expected reward for
            # every state
            for a, idx in enumerate(idxs):
                suf = str(s) + '_' + str(a)
                # Certain transitions
                if len(par1[s][a]) == 1:
                    # Add a constraint in the form
                    # x_s - f*x - l + n = r_s + r_s_a
                    # Note: add slack variables only to constraints
                    # for states with multiple actions
                    model.addConstr(rewVars['x'+str(s)] + quicksum( \
                     [-par1[s][a][0][inext]*rewVars['x'+str(next)] for inext,\
                     next in enumerate(idx)]) + (-slackVars['l'+suf]+ \
                     slackVars['n'+suf] if multActions else 0.) == \
                     rewStates[s] + rewTrans[s][a][0], 'lc'+str(lc))
                    lcs.append('lc'+str(lc))
                    lc += 1
                else: # Uncertain transition
                    # Add slack variables to the model
                    dualVars['la1'+suf] = model.addVar(name = 'la1'+suf)
                    # Declare unbounded variable
                    frees.append(dualVars['la1'+suf])
                    dualVars['la2'+suf] = model.addVar(name = 'la2'+suf)
                    for next in range(len(idx)):
                        dualVars['la3'+suf+str(next)] = model.addVar(name \
                         = 'la3'+suf+str(next))
                    model.update()

                    # Compute the inverse of the uncertainty parameter K
                    # for convenience
                    oneoverks = 1./par1[s][a][1][0]
                    # First constraint in the form
                    # x_s - lam_1 -  lam_2  - h*E*lam_3 - l + n = r_s + r_s_a
                    # x_s - lam_1 +  lam_2  + h*E*lam_3 - l + n = r_s + r_s_a
                    # For the min/max problem respectively
                    model.addConstr(rewVars['x'+str(s)] -dualVars['la1'+suf]\
                     + (-1. if MIN else +1.)*dualVars['la2'+suf] \
                     + quicksum([(-1 if MIN else +1)*oneoverks*next**0.5 \
                      *dualVars['la3'+suf+str(inext)] for inext, next in \
                      enumerate(par1[s][a][0])]) + (-slackVars['l'+suf]+ \
                      slackVars['n'+suf] if multActions else 0.) == \
                      rewStates[s] + rewTrans[s][a][0], 'lc'+str(lc))
                    lcs.append('lc'+str(lc))
                    lc += 1
                    # Second constraint in the form
                    # x_s - lambda_1 - E * lambda_3 = 0
                    # x_s - lambda_1 + E * lambda_3 = 0
                    # For the min/max problem respectively
```

```
177                     for inext, next in enumerate(idx):
178                         model.addConstr(rewVars['x'+str(next)] - \
179                          dualVars['la1'+suf] + (-1 if MIN else +1)* \
180                          oneoverks/par1[s][a][0][inext]**0.5* \
181                          dualVars['la3'+suf+str(inext)] == 0, 'lc'+str(lc))
182                         lcs.append('lc'+str(lc))
183                         lc += 1
184                         # Third constraint (quadratic) in the form
185                         # norm(lambda_3) <= lambda_2
186                         model.addQConstr(quicksum([dualVars['la3'+suf \
187                          +str(inext)]*dualVars['la3'+suf+str(inext)] for \
188                          inext in range(len(par1[s][a][0]))]) <= \
189                          dualVars['la2'+suf]*dualVars['la2'+suf],'qc'+str(qc))
190                          qcs.append('qc'+str(qc))
191                          qc += 1
192
193         # Set the lower bound to -infinity
194         # for unbounded variables
195         for v in frees:
196             v.setAttr('lb',-GRB.INFINITY)
197
198         model.update()
199
200         return model
```

We now report the full MIQCP formulation that was used to maximize the total expected reward on the Ellipsoidal-MDP in the example of Figure 2.5, subject to satisfying property $\phi = \boldsymbol{P}_{\geq 0.8}[\vartheta\, \boldsymbol{U}\, abs]$. Problem (6.11) written with the data of the model has 24 variables and 21 constraints. In this appendix, we have rounded all real-valued constants to the third decimal digit, but higher accuracy was used when we solved the problem numerically. The value of the "big" number B is:

$$B = |N| \times (r_s^{max} + r_a^{max}) = 4 \times (3 + 9) = 48$$

where r_s^{max} (r_a^{max}) is the maximum value of the state (action) reward across all states (transitions).

Note that variables are constrained by default to be non-negative, so the MIQCP formulation only reports variables that should instead be treated as unconstrained (labeled as "free"). Moreover, two variables are binary. Finally, we report also the additional constraint generated by the *verification engine* after the first iteration of the strategy-synthesis algorithm. This constraint is added only during the second (and final) iteration.

$$\max_{x,\lambda,z,l,n} \; x_0 \tag{D.1}$$

Subject to: $x_3 = 0$

$$x_0 - l_0^a + n_0^a = 10 + \lambda_{1,s_0}^a - \lambda_{2,s_0}^a - 2.958\lambda_{3,s_0s_2}^a - 4.031\lambda_{3,s_0s_3}^a$$
$$x_2 - \lambda_{1,s_0}^a + 8.452\lambda_{3,s_0s_2}^a = 0$$
$$x_3 - \lambda_{1,s_0}^a + 6.202\lambda_{3,s_0s_3}^a = 0$$
$$\|\boldsymbol{\lambda}_{\boldsymbol{3,s_0}}^{\boldsymbol{a}}\|_2 \leq \lambda_{2,s_0}^a$$
$$x_0 - l_0^b + n_0^b = 7 + \lambda_{1,s_0}^b - \lambda_{2,s_0}^b - 8.66\lambda_{3,s_0s_1}^b - 5\lambda_{3,s_0s_2}^b$$
$$x_1 - \lambda_{1,s_0}^b + 11.547\lambda_{3,s_0s_1}^b = 0$$
$$x_2 - \lambda_{1,s_0}^b + 20\lambda_{3,s_0s_2}^b = 0$$
$$\|\boldsymbol{\lambda}_{\boldsymbol{3,s_0}}^{\boldsymbol{b}}\|_2 \leq \lambda_{2,s_0}^b$$
$$l_0^a \leq 48z_0^a$$
$$n_0^a \leq 48z_0^a$$
$$l_0^b \leq 48z_0^b$$
$$n_0^b \leq 48z_0^b$$
$$z_0^a + z_0^b = 1$$
$$0.8x_1 - l_1^a + n_1^a = 3 + 0.8x_2$$
$$x_1 - l_1^b + n_1^b = 3 + x_2$$
$$z_1^a + z_1^b = 1$$
$$l_1^a \leq 48z_1^a$$
$$n_1^a \leq 48z_1^a$$
$$l_1^b \leq 48z_1^b$$
$$n_1^b \leq 48z_1^b$$
$$x_2 = 1 + x_3$$

Free: $\lambda_{1,s_0}^a, \lambda_{1,s_0}^b$

Binary: $z_0^a, z_0^b, z_1^a, z_1^b$

Extra constraint

$$z_0^b + z_1^a \geq 1$$