Masaryk University

Faculty of Informatics


# Control Strategy Synthesis for Path Planning under Temporal Constraints

PhD Thesis

**Mária Svoreňová**
**Advisor:** Ivana Černá

Brno, June 2015

## Acknowledgement

First, I would like to thank my advisor Ivana Černá who has always been there for me, offering guidance and support in work matters. While this alone makes her an amazing person to work with, I especially want to express my gratitude for her being there for me on the personal level as well. Being as active in the academic community as she is, I always admired the cheerful attitude and the amount of time she devotes to her students.

My big thank you goes to Jiří Barnat, the leader of ParaDiSe laboratory, for his support, many valuable work and life advice and daily supply of goofy jokes. My PhD experience would not have been so enjoyable without all my current and former labmates in ParaDiSe, the "kids". I want to thank them for the great environment they unintentionally created and for all the laughs and knowledge we exchanged. From my experience, it is not common to work in such a close group. I especially want to thank Jana Tůmová (my great teacher and an amazing friend), Milan Češka, Nikola Beneš, Peter Bezděk and Petr Bauch.

Next, I would like express my deep gratitude to Calin Belta from Boston University whom I consider my second advisor. I have spent 9 months in total working in his group which was a crucially valuable experience for me because as a computer scientist, I got to work side by side with engineers getting a great insight in control theory and robotics. Thanks to Calin's friendly and caring attitude, I always felt like being a part of his group. Since recently, I also have the great pleasure working with Krishnendu Chatterjee from IST Austria whose work I deeply admire and I am thankful for the hospitality.

I would not be where I am right now without my family. My mom and dad always encouraged to go after my desires and proudly supported my every choice. They taught me that nothing is as hard as it looks like with the right attitude and no matter the circumstances, there's always a just and kind way of doing things. My sisters have always been there for me, sharing my struggles and joys. The more diverse we grow, the more I learn from them. Thank you for everything.

Finally, I would like to thank all my friends, the long-lasting ones and those rediscovered ones. You all make my life so enjoyable and full of new impulses. I deeply appreciate your support in the best and even more in the not-so-great times. To name at least few – Valika & Miro, Fenďo & Ivka, Ľuba & Standa, Radka, Xomi, Onim & Zuzka, Jaro, Kejty.

Not forgetting, I would like to thank Maťo for all the time we shared and the unlimited support from him and his family.

# Abstract

Modern engineering systems such as autonomous ground and aerial vehicles, embedded medical devices and human-robot collaborative teams evolve quickly due to industrial needs as well as academic and public competitions. As these systems are typically safety-critical, there is an equally growing need for formal approaches to their specification, design and verification.

Motivated by mobile robotics, in this work we focus on the problem of path planning, where the goal is to synthesize a low-resolution high-level path or a control strategy for a complex dynamic system that satisfies a given specification. We focus on specification expressed as a formula of Linear Temporal Logic (LTL) over the state space of the system. We embrace the standard hierarchical approach to employing formal methods in path planning. The approach consists of first modeling the system using a discrete model, synthesizing a control strategy for the model and finally implementing the control rules in the original system. We employ formal methods such as automata-based model checking and game theory to design control synthesis algorithms with strong mathematical guarantees.

In the first part of the thesis, we assume that a discrete model of the system is already given and we focus on the second, synthesis step of the hierarchical approach. We design algorithms for deterministic as well as probabilistic systems to synthesize strategies that guarantee satisfaction of an LTL formula, while at the same time optimize a value function over (possibly) dynamic and partially observed values appearing in states of the system and interpreted either as rewards or penalties. In the second part, we consider the more general problem of synthesizing a control strategy for a stochastic linear dynamic system with respect to an LTL formula. We design an iterative abstraction-refinement algorithm that builds an abstraction of the system using a $2^1/_2$-player game, solves the game obtaining a partial solution and then builds a new, more precise abstraction using a deep analysis of the game. All designed algorithms were implemented and are demonstrated and evaluated on illustrative case studies.



**Keywords:** control strategy synthesis, linear temporal logic, transition system, Markov decision process, $2^1/_2$-player game, linear stochastic system, optimal control, receding horizon control, abstraction-refinement

# Contents

# Chapter 1

# Introduction

In our everyday life, we work with or come into contact with many computer-based systems. We rely on them and we trust them to work as expected. Despite our expectations, computer systems tend to fail every now and then. The consequences of failures range from minor discomfort to catastrophic situations resulting in a huge loss of resources, money or even human lives. At the same time, it is these safety-critical systems that lie in the center of the technological progress, concentrated in areas such as aerospace, automotive, chemical processes, civil infrastructure, energy, health, manufacturing and transportation.

The techniques commonly used in practice to detect faulty behavior of computer systems are testing and simulation. They can reveal a large number of errors in a reasonable time, however, they cannot be used to prove their non-existence. In contrast, *formal verification* methods in computer science can prove the satisfaction or violation of a given property. The area of formal verification has a long history and a number of techniques have been introduced and successfully employed in real-life scenarios. Typically, computer-based systems are digital and hence modeled as discrete-state systems evolving in discrete or continuous time. The properties of interest are usually complex time-dependent properties expressed using a suitable temporal logic such as Computation Tree Logic (CTL) or Linear Temporal Logic (LTL).

A different approach to ensuring properties of computer systems is correct-by-construction design, where instead of verifying that the system meets given requirements once it is designed, one focuses on designing the system in a way that ensures the requirements. The corresponding research area is referred to as *formal synthesis* and in this work, we are particularly interested in branch of synthesis called *strategy synthesis*. Here, the goal is to synthesize a strategy for a system that determines, given the history of the execution of the system, the inputs to be applied in order to satisfy a given objective over a finite or infinite time horizon. Similarly as in formal verification, the objectives are typically temporal properties or alternatively, the behavior of the system can be optimized with respect to a given value function over executions of the system. Some of the well-

known techniques for strategy synthesis rely on *game theory*. Indeed, the control of a system in order to satisfy an objective can be seen as a game between the system controllable through its inputs and the uncontrollable environment that affects the systems' state.

Comparing to computer-based systems, modern engineering systems, recently referred to as cyber-physical systems, integrate hardware and software to create complex behavior that involves continuous, dynamic components. These systems are then modeled with differential or difference equations and examples include autonomous ground and aerial vehicles and embedded medical devices. Analogously to strategy synthesis for computer-based systems, *control theory* is an interdisciplinary branch of engineering and mathematics that deals with the behavior of dynamic engineering systems. Typically, the control rules are synthesized in order to optimize the behavior of a system or to ensure a "simple" temporal objective such as safety, stability or reachability of a given goal configuration.

Over the last several decades, cyber-physical systems have been evolving rapidly due to industrial needs as well as academic and public competitions such as DARPA Grand challenge for autonomous vehicles or the recent DARPA Robotics challenge for semi-autonomous ground robots performing complex tasks in dangerous and degraded environments. The quick evolution emphasizes the need for robust approaches to design and control of these systems. To this end, formal methods from computer science have been employed in the last twenty years. The main advantage of the algorithms based on formal verification and strategy synthesis is that they offer rigorous techniques and result in strategies with strong mathematical guarantees. Also, since computer science deals with "complex" properties of rather "simple" systems and dually, control theory considers "complex" dynamic systems against "simple" objectives, the use of formal methods in control promises analysis of "complex" behavior of "complex" systems.

One of the common approaches to employing formal methods in control of complex systems is the following hierarchical approach. First, the complex dynamical system is abstracted using a finite discrete model. Second, a complying control strategy for the model is synthesized using formal techniques. Finally, the obtained control rules are implemented the original system using low level controllers. If the abstract model captures all behavior of the system relevant to the desired property then it is guaranteed that every control strategy satisfying the property in the abstract model also satisfies the property in the original system.

It is important to note that the existing formal techniques from computer science often cannot be applied on the abstract model directly and without any change. There are two main reasons for that. First, the constructed model is often large and many formal techniques suffer from high computational complexity. Second, problems motivated by real applications usually involve specific aspects that are not typically considered in computer science. For example, the system may involve dynamic elements such as dynamic obstacles and reactive environment. The system can be given an unrealizable objective with the goal to control

the system to behave as close to satisfaction as possible, or we might have a set of systems or robots cooperating on a single common mission. Robotic systems are also heavily dependent on the use of sensors that introduce uncertainty. For example, only some part of the environment in close proximity may be fully observed or the sensor information may provide only a partial information about the current state of the system. Many of these problems are hard to solve or even undecidable in their general form and this area is constantly active, specifying new control problems. The algorithms and techniques designed to solve these problems then contribute not only to the area of control but also to the area of computer science when conceptually new problems are formulated.

Motivated by robotic applications, in this work we focus on the problems arising in path planning, where the goal is to synthesize a low-resolution high-level path or a control strategy for a system that satisfies given specification. Historically, the central problem of path planning was to plan a path for a mobile robot to move from position A to position B while avoiding obstacles. However, with the fast evolving area of cyber-physical systems and especially autonomous cars, more complex properties are becoming of interest. In this work, we consider specification expressed as a formula of LTL over the state space of the system. Recently, LTL has gained a lot of attention in control theory as a language that is expressive enough to describe complex tasks and yet has some resemblance to natural language. We embrace the above mentioned hierarchical approach to employ formal methods such as automata-based model checking and games on graphs to synthesize control strategies with strong mathematical guarantees.

In the first part of the thesis, we assume a finite abstraction of the dynamic system is already given and we focus only on the synthesis step of the hierarchical approach. We consider both deterministic and probabilistic finite systems and we design algorithms to synthesize strategies that guarantee satisfaction of an LTL formula, while at the same time optimize a value function over (possibly) dynamic and partially observed values interpreted either as rewards or penalties. Such a connection between optimal and temporal logic control is an intriguing problem with a potentially high impact in applications such as control of a mobile robot on a complex mission under tight fuel and time constraints.

In the second part, we consider the more general problem of synthesizing control for infinite dynamic systems. Namely, we consider the problem of LTL control synthesis for discrete-time stochastic systems with linear dynamics. These systems are a popular modeling formalism in control theory because they are expressive enough to model many real-life systems, while linear dynamics is simple enough to analyze. Motivated by the hierarchical approach, we design an iterative abstraction-refinement algorithm that builds an abstraction of the system using a finite game.

In Section 1.1, we offer an overview of standard as well as recent results in computer science, control theory and path planning. The most related existing techniques are also discussed at the beginning of the chapters containing our main

results. The remaining structure of the thesis and its contributions are described in detail at the end of this chapter in Section 1.2.

## 1.1 Related Work

### 1.1.1 Formal Methods in Computer Science

#### Formal verification

In computer science, the problem of formal verification requires to prove or disprove that behaviors of a system satisfy certain properties. There exist two main approaches to formal verification that have been pursued in the literature. The first approach is deductive verification, where the system and the specification are used to generate a set of proof obligations. The techniques used to establish the truth of these obligations include theorem provers and satisfiability modulo theories (SMT) solvers, for overview of the techniques see [RV01]. This approach typically requires interaction with a skilled designer.

The second, automatic approach to formal verification that is also embraced in this work is model checking [BK08]. It entails exhaustive systematic analysis of a finite model of the system. Explicit-state model checking suffers from the well-known state-space explosion problem, where even a small implicit representation of the system and the objective can result in a very large intermediate state space that needs to be analyzed. The approaches to combat this problem include the following. Symbolic model checking allows for large numbers of states to be considered in a single step. Historically, the first symbolic approaches used binary decision diagrams (BDDs) [McM93]. Bounded model checking combines model checking with satisfiability solving to investigate only a fixed number of steps of the system at a time [CBRZ01]. Partial order reduction techniques used in systems involving interleaving reduce the number of possible orderings of events that need to be investigated using, e.g., ample sets [Pel93]. Abstraction attempts to first simplify the system at hand using techniques such as quotienting based on bisimulation, simulation or trace equivalence, for overview of these methods see [Ber01]. Finally, counterexample guided abstraction refinement (CEGAR) [CGJ$^+$00] starts verifying the system using a coarse, imprecise abstraction and iteratively refines it. When a counterexample is found, it is analyzed in order to guide the next abstraction.

#### Specification

The specification that is investigated in formal verification is usually expressed as a formula of a suitable temporal logic over states of the system or their labels. Temporal properties of systems can be divided into two categories. Linear-time properties are properties of an individual run, i.e., a sequence of states of a system. However, runs of the system might be branching meaning that in every

particular point of a run there might exist multiple possible extensions of the run. Branching-time properties then specify behaviors of infinite trees of these runs. To express linear-time properties, LTL was introduced in [Pnu77] and analogously, branching-time properties are typically formulated in CTL [CE82, QS82]. CTL* [EH86] combines both linear-time and branching-time properties and modal $\mu$-calculus [Koz83] is an even more powerful specification formalism.

In this work, we consider linear-time properties of systems expressed as formulas of LTL. Our motivation comes from the recent growing interest of engineering and especially robotics community in LTL that we motivate properly later in this section. There are two main approaches to LTL model checking. First, tableau-based techniques were introduced in [LP85, CGH97]. Different, widely used approach to LTL model checking explores the connection between LTL formulas and finite automata over infinite words also known as $\omega$-automata. In [VW94], it was shown that properties expressed as LTL formulas are $\omega$-regular and therefore, every LTL formula can be translated to an $\omega$-automaton. The authors in [VW94] introduce the framework for automata-based LTL model checking, where an LTL formula is first translated to a (non-deterministic) Büchi automaton and the system is analyzed against the automaton using graph algorithms. For systems that contain uncertainty such as non-determinism or probability, a more structured model than a non-deterministic $\omega$-automaton needs to be considered. In such a case, an LTL formula is typically translated to a deterministic Rabin automaton [GTW02].

Note that unlike for deterministic and non-deterministic systems, for systems with probability we can consider various formulations of the verification problem. Besides verifying that all behaviors of the system meet the specification, we can ask to verify whether the specification is satisfied with probability 1 or non-zero probability (referred to as qualitative verification), or with at least a given probability $p \in (0, 1)$ (referred to as quantitative verification). The probability 1 satisfaction is the strongest guarantee one can achieve while accounting for the probability in the system and in this work, we focus only on probability 1 analysis when considering systems with probability. An overview of techniques for qualitative as well as quantitative LTL model checking can be found in [BK08].

### Control synthesis and game theory

The counterpart to formal verification is the problem of control synthesis that aims to find a control strategy for the system that satisfies given requirements. In this case, the principles of automata-based approach from LTL model checking can be utilized in an almost straightforward way, as we discuss in Section 2.5. Once the requirement is represented with an $\omega$-automaton, the system is analyzed against the automaton using techniques from game theory. The control synthesis problem can be interpreted as a game between the controllable system and the uncontrollable environment. Games on graphs [AG11] offer a convenient formalism, where a game arena consists of a finite graph whose states are partitioned

among two players. When the game is in a state of a particular player, this player decides the next state of the game based on the available transitions. Typically, Player 1 represents the system and Player 2 represents the environment. In the most general case, they both behave probabilistically. Such a game arena is called a $2^1/_2$-player game. By considering a weaker Player 1 and/or Player 2, i.e., without probability, or completely deterministic, we obtain less general game arenas corresponding to non-deterministic transition systems, Markov decision processes (MDPs) or deterministic transition systems. To solve a game means to synthesize a strategy for Player 1 that satisfies a given winning objective in a given winning mode. A winning objective defines which plays of the game are winning for Player 1 and thus losing for Player 2, and vice versa. It can either be given as a qualitative objective in the form of a temporal logic formula or an $\omega$-automaton, a quantitative objective in the form of a value function such as mean payoff or energy objective, or a combination of qualitative and quantitative objectives. Similarly as in formal verification, the winning mode determines whether we want Player 1 to win in every play of the game, every play with probability one, non-zero probability or given probability $p \in (0, 1)$, against any Player 2 strategy. Recent surveys such as [CH12, CD10, CDH12, CD12] offer an exhaustive overview of the algorithms, complexity results and remaining open problems for different combinations of game models, winning objectives and winning modes.

### 1.1.2 Control Theory

Unlike computer-based systems, engineering systems are generally described by a set of differential or difference equations, for continuous or discrete time systems, respectively. The state space and/or control space is infinite, often continuous given as a polytopic set in a multi-dimensional space. The standard techniques in control theory are the following. To achieve or verify stability and safety of such systems, reachability analysis and approaches based on potential fields and Lyapunov functions can be employed [AM08]. Optimization control problems have also been extensively studied and approached using linear programming [Ber12]. Model predictive control (MPC) offers a great tool to handle complex dynamical systems and systems with uncertainty [RM09]. MPC is an iterative approach to finite-time horizon optimization, where in every iteration, an online and on-the-fly computation is used to explore possible executions starting in the current state of the system and find a cost-minimizing control strategy over a fixed finite time window. Only the first step of the control strategy is implemented and the next iteration is started with the new current state. MPC is also referred to as receding horizon control as the prediction horizon keeps being shifted forward.

In control theory, several methods for building a finite abstraction of the infinite system have been developed with different application scopes. In cell decomposition techniques, a finite graph is build by partitioning the continuous space into a finite set of polytopic sets and a transition is implied between every two adjacent partition elements, or computed based on the dy-

namics of the system. On the one hand, these methods offer strong mathematical guarantees by accounting for the whole state/control space, on the other hand they do not scale well in higher dimensions. For high-dimensional or otherwise complex, e.g., nonlinear, systems, sampling-based methods have proven to be suitable. In this case, the finite graph approximating the system is built over a finite set of samples chosen from the continuous space. For an overview of standard abstraction techniques see [LaV06], more recent results include [BKV10, KF11, KB08, KGFP09, YTv+12, TI12, PGT08, Rei11, ZPMT12].

### 1.1.3 Formal Methods in Path Planning

In our work, we are motivated by the path planning problem for robotic systems. Formal methods have been increasingly employed in this area, especially over the last decade. The breakthrough came when DARPA (Defense Advanced Research Projects Agency), an agency of the United States Department of Defense, funded a prize competition DARPA Grand Challenge for driverless cars. The goal was to create the first fully autonomous ground vehicles capable of completing a substantial mission in an off-road or an urban-like environment within a limited time. Altogether, there were three events in 2004, 2005 and 2007 and they unequivocally manifested a need for robust and provably correct techniques to solve control problems. To formalize the complex behavior of a system, temporal logics are being utilized. In particular, linear time properties are of interest since one is typically interested in the outcome of a single execution of the robot. LTL has been widely used to express such properties as a language that is expressive enough to describe complex tasks and yet has some resemblance to natural language.

The standard three step hierarchical approach consisting of the abstraction, synthesis and implementation step, has been developed for LTL control of simple discrete-time dynamic systems [FGKP09, KGFP09] (implemented in a tool LTLMoP [FJK10]) and discrete and continuous-time linear systems [TP06, GDLB12]. With some added conservatism, more complicated dynamics and stochastic dynamics can also be handled [CB12, Gir12, JGP06]. Motivated by the DARPA competitions, the authors in [WTM12c, Won10] design a framework for LTL control of dynamic systems that alleviates the computational complexity of the hierarchical approach. Using receding horizon planning, the LTL task is broken into successive short-horizon tasks that are achieved by computing a series of discrete trajectories and their continuous implementation. The framework was implemented in a tool TuLiP [WTO+11]. Alternatively, the standard hierarchical approach can be enhanced with an iterative abstraction refinement similar to the CEGAR framework in model checking [GLB12, LAB12, SKC+15].

In the second step of the hierarchical approach, path planning techniques adapt formal methods to synthesize controllers for deterministic finite systems [BK08], Büchi and Rabin games are used to synthesize control for nondeterministic systems [PPS06], and probabilistic games are used to compute a strategy for finite probabilistic systems such as MDPs [BK08]. Many works in this area focus solely

on the second step of the hierarchical approach. In such a case, more complex specification than a single LTL formula is often addressed. In optimal temporal control, the aim is to synthesize a control strategy that satisfies an LTL formula, while optimizing an additional value function concerning the expected time to satisfaction [STBR11], a reward function [DLB12, STBv12, CKSW13] or a cost function [DSBR11, WTM12a, SvB15, SvB13a]. To battle the high computational complexity of LTL synthesis, which is 2EXPTIME-complete in general, a fragment called General Reactivity(1) (GR(1)) was introduced in [PPS06]. This fragment is expressive enough to describe many interesting properties typically considered in robot motion such as reachability, safety and liveness, yet it allows polynomial synthesis algorithms. GR(1) was originally designed to synthesize strategies in the presence of a reactive environment. In such a case the synthesis is exponential in the size of the formula describing both the environment and the desirable behavior.

Besides expressing motion properties such as "go from A to B and avoid obstacles", one can also use LTL to argue about actions such as "grasp a ball". In [TMDK14], the authors synthesize a maximally satisfying control strategy while taking into account that the robot's action executions may fail. Also, it is not always the case that the desired LTL property can be satisfied in the system. For example, an autonomous car is required to avoid collisions and obey traffic rules such as to never enter a pavement or to not enter the left lane. To avoid obstacles, the car might need to temporarily violate some of the rules. Works such as [RKG13] can be used to explore possible causes of unsynthesizability and in [TCK+13], the authors construct least-violating strategies for such scenarios.

In the presence of uncertainty due to unreliable sensors or actuators, partially observable systems such as partially observable MDPs can be used to model the uncertainty. Here, during executions, the current state of the system is not directly observed and only a partial information is provided that implies a belief state, i.e., a set of states in which the system might be based on the past observations. Partially observable games can be employed to solve control problems for these models [CD10, CDH12, CCGK15, SCL+15]. Alternatively, uncertain MDPs [WTM12b] and bounded-parameter MDPs [LAB12] have been considered to model uncertainties.

Other interesting and active areas of motion planning that employ formal methods include multi-robot control, adaptation of machine learning in motion planning, and various applications such as autonomous vehicle and UAV control, swarm robotics, traffic control, robotic surgery, video game artificial intelligence and the study of biological molecules.

## 1.2 Thesis Contribution and Summary

The objective of the thesis is to design correct-by-construction control synthesis algorithms for selected problems motivated by robot path planning. We consider

complex discrete and dynamic systems with complex temporal specification (alone and in a combination with an optimization objective). All designed algorithms build on the standard as well as recent results from both computer science and control theory and thus make a step towards interconnecting the two areas. Below, we summarize our results and the structure of the thesis. The full list of author's contributions can also be found in Appendix A.

**Chapter 2** covers the preliminaries used throughout the thesis. After introducing LTL and its correspondence to automata over infinite words, we define the finite discrete systems as well as dynamic systems that appear in the control problems considered in the later chapters. Finally, we discuss the principles of automata-based approach to model checking and control synthesis.

**Chapter 3** addresses control of finite discrete systems and presents solutions to three closely related control problems that combine temporal logic and optimal control.

In Section 3.1, we consider a deterministic transition system under temporal constraint expressed as an LTL formula containing persistent surveillance subgoal. Assuming that the executions of the system incur time-varying locally sensed rewards in the visited states, we aim to synthesize control strategies that maximize the expected value between states under surveillance, while at the same time satisfy the additional LTL constraint. We prove that an optimal strategy cannot be constructed because the two goals often cannot be reached simultaneously. If the system primarily collects high rewards, the LTL formula might never be satisfied and vice versa, if the system is controlled to satisfy the formula, the collected rewards might be low. We design a framework that allows for a definition of a broad class of optimization functions over finite horizon and a user-guided trade-off between the importance of making progress towards LTL satisfaction and importance of collecting high rewards. The constructed control strategy synthesis algorithm combines receding horizon control with formal methods.

In Section 3.2, we consider a similar problem, where the collected values are interpreted as penalties and the goal is to minimize penalty collection over infinite horizon, namely, the expected average penalty between visits of states under surveillance. We provide two algorithmic approaches to this problem. First, we derive an optimal strategy within the class of strategies that do not exploit values of penalties sensed in real time. Second, we show that by taking advantage of locally sensing the penalties, we can construct a whole class of heuristic receding horizon strategies leading to lower collected penalty. While still ensuring satisfaction of the LTL constraint, we cannot guarantee optimality in the latter case.

Finally, in Section 3.3, we consider a probabilistic system modeled as a Markov decision process, where every application of a control action in a state is associated with a non-negative real value interpreted as a penalty. Similarly as in the first two problems, given an LTL formula with persistent surveillance subgoal, our goal is to synthesize a control strategy that guarantees the satisfaction of the LTL formula

with probability 1, while at the same time minimizes the expected average penalty between consecutive visits of states under surveillance. The existing solutions to this problem are sub-optimal. We provide a way to generate an optimal solution that can be seen as an adaptation of the framework in Section 3.2.

All designed approaches have been implemented and are demonstrated and evaluated on illustrative case studies. Moreover, an experimental control tool ConTool [SMv13] has been created that allows simulation of the control algorithms for the first two problems.

These results are based on the following publications:

[SvB15] M. Svoreňová, I. Černá, and C. Belta. Optimal Temporal Logic Control for Deterministic Transition Systems with Probabilistic Penalties. *IEEE Transactions on Automatic Control*, 60(6):1–14, 2015.

[SvB13a] M. Svoreňová, I. Černá, and C. Belta. Optimal Control of MDPs with Temporal Logic Constraints. In *Proc. of IEEE Conference on Decision and Control CDC*, pages 3938–3943, 2013.

[SvB13b] M. Svoreňová, I. Černá, and C. Belta. Optimal Receding Horizon Control for Finite Deterministic Systems with Temporal Logic Constraints. In *Proc. of American Control Conference ACC*, pages 4399–4404, 2013.

[STBv12] M. Svoreňová, J. Tůmová, J. Barnat, and I. Černá. Attraction-based Receding Horizon Path Planning with Temporal Logic Constraints. In *Proc. of IEEE Conference on Decision and Control CDC*, pages 6749–6754, 2012.

**Chapter 4** addresses a control problem for discrete-time systems with linear dynamics and stochastic uncertainty. We design an iterative abstraction-refinement algorithm that finds the maximal set of initial states of the dynamic system from which a given GR(1) formula can be satisfied with probability 1, together with the corresponding strategies. In every iteration, the system is abstracted using a $2^1/_2$-player game that is then analyzed and a new, more refined abstraction is built. While the synthesis step leverages the existing game theoretical techniques, both abstraction and refinement steps present novel approaches. Every iteration of the algorithm results in a partial solution. The designed algorithm is sound but completeness cannot be guaranteed. To the best of our knowledge, the presented approach is the first attempt to construct abstraction-refinement of stochastic systems with continuous state and control spaces in the form of $2^1/_2$-player games. We argue that in order to obtain the desired control strategies, the system must indeed be abstracted using a $2^1/_2$-player game and not using a weaker model such as a 2 player game. Using a case study, we compare

our approach with two simpler algorithms for reachability analysis. While the simpler algorithms find the set of satisfying initial states considerably faster, they are not able to construct the desired strategies demonstrating that abstraction using a $2^1/_2$-player game is necessary even in the case of reachability.

These results are based on the following publication:

[SKC+15] M. Svoreňová, J. Křetínský, M. Chmelík, K. Chatterjee, I. Černá, and C. Belta. Temporal Logic Control for Stochastic Linear Systems using Abstraction Refinement of Probabilistic Games. In *Proc. of Conference on Hybrid Systems: Computation and Control HSCC*, pages 259–268, 2015.

Finally, **Chapter 5** concludes the main body of the thesis, **Appendix A** lists all publications and contributions of the author of the thesis, and technical **Appendices B** and **C** contain algorithms for game solving and operations with polytopes, respectively, used as routines in the framework designed in Chapter 4.

# Chapter 2

# Preliminaries

In this chapter, we cover the main notions used throughout the thesis. After we set the basic notation, we introduce linear temporal logic as the language to describe rich specification for systems. We follow with the definitions of finite and infinite systems that appear in the following chapters. Given a system and a specification, we formally state the problems of formal verification and control synthesis and overview the well-known automata-based approach.

## 2.1 Notation

For a finite set $X$, we use $|X|$, $X^+$, $X^*$ and $X^\omega$ to denote the cardinality of $X$ and the set of all finite non-empty, finite and infinite sequences of elements of $X$, respectively. A finite sequence $\sigma = x_0 \dots x_n \in X^*$ has length $|\sigma| = n + 1$, also referred to as the number of stages, and $\text{last}(\sigma) = x_n$ is the last state. For $0 \leq i \leq n$, $\sigma(i) = x_i$ is the $i$-th element, $\sigma^{\to i} = x_0 \dots x_i$ is the prefix ending with the $i$-th element and $\sigma^{i\to} = x_i \dots x_n$ is the suffix starting with the $i$-th element. Similarly, for an infinite sequence $\rho = x_0 x_1 \dots \in X^\omega$, $\rho(i) = x_i$, $\rho^{\to i} = x_0 \dots x_i$ and $\rho^{i\to} = x_i x_{i+1} \dots$ for all $i \geq 0$. Additionally, $\inf(\rho)$ denotes the set of all elements appearing infinitely many times in $\rho$. For two sets $X_1 \subseteq X^*, X_2 \subseteq X^* \cup X^\omega$, we use $X_1 \cdot X_2 = \{x_1 \cdot x_2 \mid x_1 \in X_1, x_2 \in X_2\}$ to denote their concatenation.

For two sets $X, Y$, we use $X \times Y = \{(x, y) \mid x \in X, y \in Y\}$ to denote the Cartesian product of the two sets. For every element $(x, y) \in X \times Y$, we denote $\pi_1((x, y)) = x$ and $\pi_2((x, y)) = y$ the projection on the first and second component, respectively. We naturally extend this notation over the sequences from $(X \times Y)^+, (X \times Y)^*, (X \times Y)^\omega$, e.g., $\pi_1\big((x_1, y_1) \dots (x_n, y_n)\big) = x_1 \dots x_n$.

Finally, given a set $X$, $\mathcal{D}(X)$ is the set of all probability distributions over $X$ and $\{x \in X \mid d(x) > 0\}$ is the support set of a distribution $d \in \mathcal{D}(X)$.

## 2.2 Linear Temporal Logic

Linear temporal logic (LTL) is a modal temporal logic with modalities referring to time that was first proposed for formal verification of computer programs by Amir Pnueli in 1977 [Pnu77]. In this section, we formally define the logic, discuss the correspondence between formulas of LTL and automata over infinite words and introduce two special classes of formulas that we use in our work. For a more detailed reading on LTL, we refer the reader to textbooks such as [BK08].

### 2.2.1 Syntax and Semantics

**Definition 1.** *Linear Temporal Logic (LTL) formulas over a finite set AP of atomic propositions are formed according to the following grammar:*

$$\phi ::= \mathsf{true} \mid a \mid \neg\phi \mid \phi \wedge \phi \mid \mathbf{X}\,\phi \mid \phi\,\mathbf{U}\,\phi,$$

*where $a \in AP$ is an atomic proposition, $\neg$ and $\wedge$ are the standard Boolean connectives negation and conjunction, and $\mathbf{X}$ (next), $\mathbf{U}$ (until) are temporal operators.*

Formulas of LTL are interpreted over words in $(2^{AP})^\omega$. Formally, the satisfaction relation $\models \subseteq (2^{AP})^\omega \times \text{LTL}$ is the smallest relation with the following properties:

$$\begin{array}{lcl}
z \models \mathsf{true}, & & \\
z \models a & \Leftrightarrow & a \in z(0), \\
z \models \neg\phi & \Leftrightarrow & z \not\models \phi, \\
z \models \phi_1 \wedge \phi_2 & \Leftrightarrow & z \models \phi_1 \text{ and } z \models \phi_2, \\
z \models \mathbf{X}\phi & \Leftrightarrow & z^1 \models \phi, \\
z \models \phi_1 \mathbf{U} \phi_2 & \Leftrightarrow & \text{there exists } i \geq 0 \text{ such that } z^i \models \phi_2, \\
 & & \text{and for all } 0 \leq j < i, \text{ it holds that } z^j \models \phi_1.
\end{array}$$

Derived temporal operators that are commonly used in LTL are $\mathbf{F}$ (future or eventually) and $\mathbf{G}$ (generally or always) defined as $\mathbf{F}\phi = \mathsf{true}\,\mathbf{U}\phi$ and $\mathbf{G}\phi = \neg\mathbf{F}\neg\phi$. LTL formulas that describe properties often considered in path planning for robotic systems are:

- *reachability*: $\mathbf{F}\phi$ (eventually satisfy $\phi$),
- *safety*: $\mathbf{G}\phi$ ($\phi$ must hold at all times),
- *response* or *liveness*: $\mathbf{G}(\phi_1 \Rightarrow \mathbf{F}\phi_2)$ (every satisfaction of $\phi_1$ is eventually answered with satisfaction of $\phi_2$),
- *exclusive response*: $\mathbf{G}(\phi_1 \Rightarrow \mathbf{X}(\neg\phi_1 \mathbf{U} \phi_2))$ (every satisfaction of $\phi_1$ is eventually answered with satisfaction of $\phi_2$ before $\phi_1$ can be satisfied again),
- *persistent surveillance*: $\mathbf{GF}\phi$ ($\phi$ must hold infinitely many times),
- *stability*: $\mathbf{FG}\phi$ ($\phi$ must hold from some point on).

We use $|\phi|$ to denote the size of a formula $\phi$ defined as the number of operators appearing in $\phi$. Inductively, $|\mathsf{true}| = 0$, $|a| = 0$ for all $a \in AP$, $|\neg\phi| = |\mathbf{X}\phi| = |\phi| + 1$ and $|\phi_1 \wedge \phi_2| = |\phi_1 \mathbf{U} \phi_2| = |\phi_1| + |\phi_2| + 1$.

### 2.2.2 Automata Representation

The set of all words satisfying an LTL formula forms an $\omega$-regular language [VW94], i.e., a language that can be represented by an $\omega$-automaton.

**Definition 2.** *A (non-deterministic) $\omega$-automaton is a tuple*

$$\mathcal{A} = (Q, \Sigma, \delta, q_0, Acc),$$

*where $Q$ is a nonempty finite set of states, $\Sigma$ is a finite alphabet, $\delta \subseteq Q \times \Sigma \times Q$ is a transition relation such that for every $q \in Q$, $a \in \Sigma$, there exists $q' \in Q$ such that $(q, a, q') \in \delta$, $q_0 \in Q$ is the initial state, and $Acc$ is an accepting condition.*

A run $q_0 q_1 \ldots \in Q^\omega$ of $\mathcal{A}$ is an infinite sequence such that for every $i \geq 0$ there exists $a_i \in \Sigma$ with $(q_i, a_i, q_{i+1}) \in \delta$. We say that the word $a_0 a_1 \ldots \in \Sigma^\omega$ induces the run $q_0 q_1 \ldots$. Every word in $\Sigma^\omega$ induces a non-empty set of runs of $\mathcal{A}$. An $\omega$-automaton is called deterministic if every word over its alphabet induces a single run.

A run $q_0 q_1 \ldots$ of $\mathcal{A}$ is accepting if and only if it satisfies the accepting condition $Acc$. A word over $\Sigma$ is accepted by $\mathcal{A}$ if it induces at least one accepting run. The set of all words over $2^{AP}$ accepted by the $\omega$-automaton $\mathcal{A}$ is called the language of $\mathcal{A}$.

Depending on the accepting condition, we recognize Büchi, Rabin, Streett, parity and Müller automata and their non-deterministic and deterministic versions. Below, we define some of these automata that appear in the standard translation techniques for LTL. For further reading on $\omega$-automata and their connection to logics, see [GTW02].

#### Non-deterministic $\omega$-automata

A standard approach is to translate an LTL formula over the set of atomic propositions $AP$ to a *non-deterministic Büchi automaton* (BA) over alphabet $2^{AP}$. A Büchi accepting condition for an $\omega$-automaton $\mathcal{A}$ is given as a set of so called accepting states $F \subseteq Q$. A run $q_0 q_1 \ldots$ is called accepting if and only if it visits at least one accepting state infinitely many times, i.e.,

$$\inf(q_0 q_1 \ldots) \cap F \neq \emptyset.$$

The first algorithm to translate an LTL formula to a BA was presented in [VW94], other appeared in [GO01a, SB00]. The translation is EXPTIME-complete, i.e., the size of the obtained Büchi automaton is in the worst case exponential in the size of the formula [GO01a]. Online implementations such as [GO01b] are available to translate an LTL formula to a BA.

#### Deterministic $\omega$-automata

Every LTL formula can be translated to a non-deterministic Büchi automaton, but there exist LTL formulas for which there is no deterministic Büchi automaton such as the stability property $\mathbf{FG}a$ for $a \in AP$. To translate an LTL formula to a deterministic $\omega$-automaton, the standard approach is to use a *deterministic Rabin automaton* (DRA) over alphabet $2^{AP}$. The Rabin accepting condition is given as a set $Acc \subseteq 2^Q \times 2^Q$. A run $q_0 q_1 \ldots$ of a DRA $\mathcal{A}$ is called accepting if there exists a pair $(E, F) \in Acc$ such that the run visits every state from $E$ only finitely many times and at least one state from $F$ infinitely many times, i.e.,

$$\inf(q_0 q_1 \ldots) \cap E = \emptyset \quad \wedge \quad \inf(q_0 q_1 \ldots) \cap F \neq \emptyset.$$

The problem of translating LTL to DRA is 2EXPTIME-complete [PR89]. There are two general approaches. The first, traditional one translates an LTL formula to a BA and then uses Safra's construction [Saf88] to obtain a deterministic $\omega$-automaton. An implementation with improved Safra's construction is available online at [Kle07]. As this approach may lead to unnecessarily big automata for some formulas, there exist several translations avoiding Safra's construction for chosen fragments of LTL. For comparison of available algorithms and tools, see [BKS13].

### 2.2.3 General Reactivity(1)

To avoid the high computational complexity of full LTL, the General Reactivity(1) (GR(1)) fragment was introduced in [PPS06]. Here, we use the extended version of the standard definition that appears in the same work.

**Definition 3.** *A GR(1) formula $\phi$ is a particular type of an LTL formula over a set of atomic propositions $AP$ of the form*

$$\phi = \Big(\bigwedge_{i=1}^{m} \varphi_i\Big) \implies \Big(\bigwedge_{j=1}^{n} \varphi_j\Big), \tag{2.1}$$

*where each $\varphi_i, \varphi_j$ is an LTL formula that can be represented by a deterministic Büchi $\omega$-automaton.*

Many interesting properties can be expressed using formulas of GR(1) such as the properties mentioned in Section 2.2.1 and their combinations according to the form in Equation 2.1, except for the stability property as it cannot be represented by a deterministic BA.

The advantage of using GR(1) instead of full LTL as the specification language is that realizability for LTL is 2EXPTIME-complete [PR89], whereas for GR(1) it is only cubic in the size of the formula [PPS06].

To represent formulas of GR(1), we use *$\omega$-automata with Büchi implication acceptance condition*, also known as one-pair Streett acceptance condition. The

condition is defined as $Acc = (E, F) \in 2^Q \times 2^Q$ and a run $q_0 q_1 \ldots$ is accepting, if it holds that if the set $E$ is visited infinitely often, then the set $F$ is visited infinitely often, i.e.,

$$\inf(q_0 q_1 \ldots) \cap E \neq \emptyset \quad \Longrightarrow \quad \inf(q_0 q_1 \ldots) \cap F \neq \emptyset.$$

Note that the Büchi acceptance condition is a special case of Büchi implication acceptance condition with $E = Q$. To translate a GR(1) formula to an $\omega$-automata with Büchi implication acceptance condition, one first constructs deterministic Büchi automata for each formula $\varphi_i, \varphi_j$ and the deterministic Büchi automaton that accepts the intersection of the languages of the given automata can be computed using the well-known counting construction [BK08].

### 2.2.4 LTL with Persistent Surveillance

In our work, we often consider LTL formulas of the following form

$$\phi = \varphi \wedge \mathbf{GF}\, a_{\mathsf{sur}}, \tag{2.2}$$

where $\varphi$ is an LTL formula over $AP$ and $a_{\mathsf{sur}} \in AP$ is so-called surveillance proposition. Besides satisfying the formula $\varphi$, formula with persistent surveillance that the surveillance proposition $a_{\mathsf{sur}}$ is satisfied infinitely many times.

Given a word $z \in AP^+ \cup AP^\omega$, we say that every satisfaction of the property $a_{\mathsf{sur}}$ in $z$ completes a *surveillance cycle*. Specifically, the first $i > 0$ such that $a_{\mathsf{sur}} \in z(i)$ completes the first surveillance cycle of a word. For a finite word $z \in AP^+$ such that $a_{\mathsf{sur}} \in \mathrm{last}(z)$, $\sharp(z)$ denotes the number of complete surveillance cycles in $z$, otherwise $\sharp(z)$ is the number of complete surveillance cycles plus one.

Technically, LTL with persistent surveillance is not a fragment of LTL because the form in Equation 2.2 does not restrict the expressiveness of LTL. Indeed, every LTL formula $\varphi$ over $AP$ is equivalent to the formula $\phi = \varphi \wedge \mathbf{GF}\,\mathsf{true}$.

## 2.3 Finite Discrete Systems

In this section, we introduce three increasingly more general types of discrete systems with finite state and control spaces. First, a deterministic transition system is introduced that is a finite system with no uncertainty. The second model is a Markov decision process that includes probabilistic uncertainty. Finally, $2^1/_2$-player games represent the most general type of a finite system that includes both probabilistic and non-deterministic uncertainty.

### 2.3.1 Deterministic Transition Systems

#### System and its runs

**Definition 4.** *A deterministic transition system (DTS) is a tuple*

$$\mathcal{T} = (S, T, AP, L),$$

*where $S$ is a nonempty finite set of states, $T \subseteq S \times S$ is a transition relation, $AP$ is a nonempty finite set of atomic propositions and $L\colon S \to 2^{AP}$ is a labeling function.*

*An initialized DTS is a DTS $\mathcal{T} = (S, T, AP, L)$ with a distinctive initial state $s_{\mathsf{init}} \in S$.*

We assume that for every $s \in S$ there exists $s' \in S$ such that $(s, s') \in T$. A run of a DTS $\mathcal{T}$ is an infinite sequence $\rho = s_0 s_1 \ldots \in S^\omega$ such that for every $i \geq 0$ it holds $(s_i, s_{i+1}) \in T$. We use $\mathrm{Run}^{\mathcal{T}}(s)$ to denote the set of all runs of $\mathcal{T}$ that start in $s \in S$. Let $\mathrm{Run}^{\mathcal{T}} = \bigcup_{s \in S} \mathrm{Run}^{\mathcal{T}}(s)$. A finite run $\sigma = s_0 \ldots s_n$ of $\mathcal{T}$ is a finite prefix of a run of $\mathcal{T}$ and $\mathrm{Run}^{\mathcal{T}}_{\mathrm{fin}}(s)$ denotes the set of all finite runs of $\mathcal{T}$ that start in $s \in S$. Let $\mathrm{Run}^{\mathcal{T}}_{\mathrm{fin}} = \bigcup_{s \in S} \mathrm{Run}^{\mathcal{T}}_{\mathrm{fin}}(s)$. A *cycle* of a DTS $\mathcal{T}$ is a finite run $\mathsf{cyc} = c_0 \ldots c_m$ of $\mathcal{T}$ for which it holds that $(c_m, c_0) \in T$.

By extending the labeling function, every infinite run $\rho = s_0 s_1 \ldots \in \mathrm{Run}^{\mathcal{T}}$ and finite run $\sigma = s_0 \ldots s_n \in \mathrm{Run}^{\mathcal{T}}_{\mathrm{fin}}$ induces a word $L(\rho) = L(s_0)L(s_1)\ldots \in (2^{AP})^\omega$ and $L(\sigma) = L(s_0)\ldots L(s_n) \in (2^{AP})^+$ over the alphabet $2^{AP}$, respectively.

### Control strategy

**Definition 5.** *Let $\mathcal{T} = (S, T, AP, L)$ be a DTS. A control strategy for $\mathcal{T}$ is a function $C\colon \mathrm{Run}^{\mathcal{T}}_{\mathrm{fin}} \to S$ such that for every $\sigma \in \mathrm{Run}^{\mathcal{T}}_{\mathrm{fin}}$, it holds that $(\mathrm{last}(\sigma), C(\sigma)) \in T$.*

A strategy $C$ for which $C(\sigma_1) = C(\sigma_2)$, for all finite runs $\sigma_1, \sigma_2 \in \mathrm{Run}^{\mathcal{T}}_{\mathrm{fin}}$ with $\mathrm{last}(\sigma_1) = \mathrm{last}(\sigma_2)$, is called memoryless. In that case, $C$ is considered to be a function $C\colon S \to S$. A strategy is called finite-memory if it can be defined as a tuple $C = (M, \mathsf{next}, \mathsf{trans}, \mathsf{start})$, where $M$ is a non-empty finite set of modes, $\mathsf{trans}\colon M \times S \to M$ is a transition function, $\mathsf{next}\colon M \times S \to S$ selects a state of $\mathcal{T}$ to be visited next, and $\mathsf{start}\colon S \to M$ selects the starting mode. A strategy that is not finite-memory is called infinite-memory.

A run induced by a strategy $C$ is a run $\rho_C = s_0 s_1 \ldots \in \mathrm{Run}^{\mathcal{T}}$ for which $s_{i+1} = C(\rho_C^{\to i})$ for every $i \geq 0$. For every $s \in S$, there is exactly one run induced by $C$ that starts in $s$. A finite run induced by $C$ is $\sigma_C \in \mathrm{Run}^{\mathcal{T}}_{\mathrm{fin}}$, which is a finite prefix of some $\rho_C$.

**Definition 6.** *Let $\mathcal{T} = (S, T, AP, L)$ be a DTS. A sub-system of $\mathcal{T}$ is a DTS $\mathcal{U} = (S_{\mathcal{U}}, T_{\mathcal{U}}, AP, L|_{\mathcal{U}})$, where $\emptyset \neq S_{\mathcal{U}} \subseteq S$ and $T_{\mathcal{U}} \subseteq T \cap (S_{\mathcal{U}} \times S_{\mathcal{U}})$. We use $L|_{\mathcal{U}}$ to denote the labeling function $L$ restricted to the set $S_{\mathcal{U}}$. If the context is clear, we use $L$ instead of $L|_{\mathcal{U}}$.*

*A sub-system $\mathcal{U}$ of $\mathcal{T}$ is called strongly connected if for every pair of states $s, s' \in S_{\mathcal{U}}$, there exists a finite run $\sigma \in \mathrm{Run}^{\mathcal{U}}_{\mathrm{fin}}(s)$ such that $\mathrm{last}(\sigma) = s'$. A strongly connected component (SCC) of $\mathcal{T}$ is a maximal strongly connected sub-system of $\mathcal{T}$. We use $\mathrm{SCC}(\mathcal{T})$ to denote the set of all strongly connected components of $\mathcal{T}$.*

Every state of a DTS $\mathcal{T}$ belongs to at most one strongly connected component of $\mathcal{T}$. Hence, the cardinality of the set $\mathrm{SCC}(\mathcal{T})$ is bounded by the number of states of $\mathcal{T}$.

Let $C$ be a strategy, finite-memory or not, for a TS $\mathcal{T}$. For every state $s \in S$, the run $\rho_C \in \mathrm{Run}^{\mathcal{T}}(s)$ induced by $C$ satisfies $\inf(\rho_C) \subseteq S_{\mathcal{U}}$ for some $\mathcal{U} \in \mathrm{SCC}(\mathcal{T})$ [BK08]. We say that $C$ leads $\mathcal{T}$ from the state $s$ to the SCC $\mathcal{U}$.

#### Satisfaction of LTL formulas

We say that an infinite run $\rho$ of a DTS $\mathcal{T}$ satisfies an LTL formula $\phi$ over $AP$, denoted as $\rho \models \phi$, iff it holds that $L(\rho) \models \phi$. We say that a strategy $C$ for $\mathcal{T}$ satisfies $\phi$ starting from a state $s \in S$ iff the run induced by $C$ that starts from $s$ satisfies $\phi$.

### 2.3.2 Markov Decision Processes

#### System and its runs

**Definition 7.** *A Markov decision process (MDP) is a tuple*

$$\mathcal{M} = (S, Act, P, AP, L),$$

*where $S$ is a non-empty finite set of states, $Act$ is a non-empty finite set of actions, $P\colon S \times Act \times S \to [0,1]$ is a transition probability function such that for every state $s \in S$ and action $\alpha \in Act$ it holds that $\sum_{s' \in S} P(s,\alpha,s') \in \{0,1\}$, $AP$ is a finite set of atomic propositions, $L\colon S \to 2^{AP}$ is a labeling function.*

*An initialized Markov decision process is an MDP $\mathcal{M} = (S, Act, P, AP, L)$ with a distinctive initial state $s_{\mathsf{init}} \in S$.*

An action $\alpha \in Act$ is enabled in a state $s \in S$ if it holds that $\sum_{s' \in S} P(s,\alpha,s') = 1$. With a slight abuse of notation, $Act(s)$ denotes the set of all actions enabled in a state $s$. We assume $Act(s) \neq \emptyset$ for every $s \in S$.

A run of an MDP $\mathcal{M}$ is an infinite sequence of states $\rho = s_0 s_1 \ldots \in S^\omega$ such that for every $i \geq 0$, there exists $\alpha_i \in Act(s_i)$, $P(s_i, \alpha_i, s_{i+1}) > 0$. We use $\mathrm{Run}^{\mathcal{M}}(s)$ to denote the set of all runs of $\mathcal{M}$ that start in a state $s \in S$ and $\mathrm{Run}^{\mathcal{M}} = \bigcup_{s \in S} \mathrm{Run}^{\mathcal{M}}(s)$. A finite run $\sigma = s_0 \ldots s_n \in S^+$ of $\mathcal{M}$ is a finite prefix of a run in $\mathcal{M}$ and $\mathrm{Run}^{\mathcal{M}}_{\mathrm{fin}}(s)$ denotes the set of all finite runs of $\mathcal{M}$ starting in a state $s \in S$. Let $\mathrm{Run}^{\mathcal{M}}_{\mathrm{fin}} = \bigcup_{s \in S} \mathrm{Run}^{\mathcal{M}}_{\mathrm{fin}}(s)$.

Every run $\rho = s_0 s_1 \ldots \in \mathrm{Run}^{\mathcal{M}}$ induces a word $L(\rho) = L(s_0)L(s_1)\ldots \in (2^{AP})^\omega$. Similarly, a finite run $\sigma = s_0 \ldots s_n \in \mathrm{Run}^{\mathcal{M}}_{\mathrm{fin}}$ induces a finite word $L(\sigma) = L(s_0)\ldots L(s_n) \in (2^{AP})^+$.

**Remark 1.** Note that if the transition probability function of an MDP satisfies $P(s,\alpha,s') \in \{0,1\}$ for all states $s, s' \in S$ and actions $\alpha \in Act$, then the set of actions $Act$ can be omitted and the MDP can be seen as a DTS.

### Control strategy

**Definition 8.** *Let $\mathcal{M} = (S, Act, P, AP, L)$ be an MDP. A control strategy for $\mathcal{M}$ is a function $C\colon \mathrm{Run}_{\mathrm{fin}}^{\mathcal{M}} \to Act$ such that for every $\sigma \in \mathrm{Run}_{\mathrm{fin}}^{\mathcal{M}}$ it holds that $C(\sigma) \in Act(\mathrm{last}(\sigma))$.*

A strategy $C$ for which $C(\sigma) = C(\sigma')$ for all finite runs $\sigma, \sigma' \in \mathrm{Run}_{\mathrm{fin}}^{\mathcal{M}}$ with $\mathrm{last}(\sigma) = \mathrm{last}(\sigma')$ is called memoryless. In that case, we consider $C$ to be a function $C\colon S \to Act$. A strategy is called finite-memory if it is defined as a tuple $C = (M, \mathsf{next}, \mathsf{trans}, \mathsf{start})$, where $M$ is a finite set of modes, $\mathsf{trans}\colon M \times S \to M$ is a transition function, $\mathsf{next}\colon M \times S \to Act$ selects the next action to be applied in the MDP $\mathcal{M}$, and $\mathsf{start}\colon S \to M$ selects the starting mode for every $s \in S$.

A run $\rho_C = s_0 s_1 \ldots \in \mathrm{Run}^{\mathcal{M}}$ of an MDP $\mathcal{M}$ is a run under a strategy $C$ for $\mathcal{M}$ if for every $i \geq 0$, it holds that $P(s_i, C(\rho_C^{\to i}), s_{i+1}) > 0$. A finite run under $C$ is a finite prefix of a run under $C$. The set of all infinite and finite runs of $\mathcal{M}$ under $C$ starting in a state $s \in S$ are denoted by $\mathrm{Run}^{\mathcal{M},C}(s)$ and $\mathrm{Run}_{\mathrm{fin}}^{\mathcal{M},C}(s)$, respectively. Let $\mathrm{Run}^{\mathcal{M},C} = \bigcup_{s\in S} \mathrm{Run}^{\mathcal{M},C}(s)$ and $\mathrm{Run}_{\mathrm{fin}}^{\mathcal{M},C} = \bigcup_{s\in S} \mathrm{Run}_{\mathrm{fin}}^{\mathcal{M},C}(s)$.

**Definition 9.** *A sub-system of an MDP $\mathcal{M} = (S, Act, P, AP, L)$ is an MDP $\mathcal{N} = (S_{\mathcal{N}}, Act_{\mathcal{N}}, P|_{\mathcal{N}}, AP, L|_{\mathcal{N}})$ such that $\emptyset \neq S_{\mathcal{N}} \subseteq S$, $\emptyset \neq Act_{\mathcal{N}} \subseteq Act$. We use $P|_{\mathcal{N}}$ to denote the function $P$ restricted to the sets $S_{\mathcal{N}}$ and $Act_{\mathcal{N}}$. Similarly, we use $L|_{\mathcal{N}}$ with the obvious meaning. If the context is clear, we only use $P, L$ instead of $P|_{\mathcal{N}}, L|_{\mathcal{N}}$.*

*An end component (EC) of $\mathcal{M}$ is a sub-system $\mathcal{N}$ such that for every pair of states $s, s' \in S_{\mathcal{N}}$, there exists a finite run $\sigma \in \mathrm{Run}_{\mathrm{fin}}^{\mathcal{N}}(s)$ with $last(\sigma) = s'$. The set of all end components and maximal end components of $\mathcal{M}$ is denoted by $\mathrm{EC}(\mathcal{M})$ and $\mathrm{MEC}(\mathcal{M})$, respectively.*

The number of ECs of an MDP $\mathcal{M}$ can be up to exponential in the number of states of $\mathcal{M}$ and they can intersect. On the other hand, MECs are pairwise disjoint and every EC is contained in a single MEC. Hence, the number of MECs of $\mathcal{M}$ is bounded by the number of states of $\mathcal{M}$.

The following properties hold for any MDP $\mathcal{M}$ [BK08]. For every EC $\mathcal{N}$ of $\mathcal{M}$, there exists a finite-memory strategy $C$ for $\mathcal{M}$ such that $\mathcal{M}$ under $C$ starting from any state of $\mathcal{N}$ never visits a state outside $\mathcal{N}$ and all states of $\mathcal{N}$ are visited infinitely many times with probability 1. On the other hand, having any, finite-memory or not, strategy $C$, a state $s$ of $\mathcal{M}$ and a run $\rho$ of $\mathcal{M}$ under $C$ that starts in $s$, the set of states visited infinitely many times by $\rho$ forms an end component. Let $\mathsf{ec} \subseteq \mathrm{EC}(\mathcal{M})$ be the set of all ECs of $\mathcal{M}$ that correspond, in the above sense, to at least one run under the strategy $C$ that starts in the state $s$. We say that the strategy $C$ leads $\mathcal{M}$ from the state $s$ to the set $\mathsf{ec}$.

### Probability measure

Let $\mathcal{M}$ be an MDP, $s \in S$ a state of $\mathcal{M}$, and $C$ a strategy for $\mathcal{M}$. The following probability measure is used to argue about the possible outcomes of applying $C$

in $\mathcal{M}$ starting from the state $s$.

Let $\sigma \in \mathrm{Run}_{\mathrm{fin}}^{\mathcal{M},C}(s)$ be a finite run. A cylinder set $\mathrm{Cyl}(\sigma)$ of $\sigma$ is the set of all runs of $\mathcal{M}$ under $C$ that have $\sigma$ as a finite prefix. There exists a unique probability measure $\Pr_s^{\mathcal{M},C}$ on the $\sigma$-algebra generated by the set of cylinder sets of all runs in $\mathrm{Run}_{\mathrm{fin}}^{\mathcal{M},C}(s)$. For $\sigma = s_0 \dots s_n \in \mathrm{Run}_{\mathrm{fin}}^{\mathcal{M},C}(s)$, it holds

$$\Pr_s^{\mathcal{M},C}(\mathrm{Cyl}(\sigma)) = \prod_{i=0}^{n-1} P(s_i, C(\sigma^{\to i}), s_{i+1})$$

and $\Pr_s^{\mathcal{M},C}(\mathrm{Cyl}(s)) = 1$. Intuitively, given a subset $X \subseteq \mathrm{Run}^{\mathcal{M},C}(s)$, $\Pr_s^{\mathcal{M},C}(X)$ is the probability that a run of $\mathcal{M}$ under $C$ that starts in $s$ belongs to the set $X$.

#### Satisfaction of LTL formulas

Given an MDP $\mathcal{M}$ and an LTL formula $\phi$ over $AP$, we say that an infinite run $\rho$ of $\mathcal{M}$ satisfies $\phi$ over $AP$, denoted as $\rho \models \phi$, iff it holds that $L(\rho) \models \phi$. The set of all runs under given strategy $C$ that satisfy an LTL formula form a set measurable in the probability measure $\Pr_s^{\mathcal{M},C}$ [BK08]. The strategy $C$ for $\mathcal{M}$ is said to satisfy $\phi$ almost-surely, or with probability 1, starting from a state $s \in S$ iff it holds that

$$\Pr_s^{\mathcal{M},C}(\{\rho \mid \rho \models \phi\}) = \Pr_s^{\mathcal{M},C}(\phi) = 1.$$

Note that in such case, there might exist runs $\rho \in \mathrm{Run}^{\mathcal{M}}(s)$ that violate the formula but the probability of $\mathcal{M}$ executing such a run under the strategy $C$ is 0.

### 2.3.3 $2^1/_2$-player Games

#### System and its runs

**Definition 10.** *A two-player turn-based probabilistic game, or $2^1/_2$-player game, is a tuple*

$$\mathcal{G} = (S_1 \cup S_2, Act, P, AP, L),$$

*where $S = S_1 \cup S_2$ is a non-empty finite set of states partitioned into a set $S_1$ of Player 1 states and a set $S_2$ of Player 2 states, Act is a non-empty finite set of actions for the players, $P\colon (S_1 \cup S_2) \times Act \times (S_1 \cup S_2) \to [0,1]$ is a probabilistic transition function such that for every state $s \in S$ and action $\alpha \in Act$ it holds that $\sum_{s' \in S} P(s, \alpha, s') \in \{0, 1\}$, $AP$ is a finite set of atomic propositions and $L\colon S \to 2^{AP}$ is a labeling function.*

An action $\alpha \in Act$ is enabled in a state $s \in S$ if it holds that $\sum_{s' \in S} P(s, \alpha, s') = 1$. We use $Act(s)$ to denote the set of all actions enabled in a state $s$. We assume $Act(s) \neq \emptyset$ for every $s \in S$.

A run or a play of a $2^1/_2$-player game $\mathcal{G}$ is a sequence $\rho \in S^\omega$ such that for all $n \geq 1$ there exists $\alpha \in Act$ such that $P(\rho(n), \alpha, \rho(n+1)) > 0$. We use $\mathrm{Run}^{\mathcal{G}}(s)$ to denote the set of all plays of $\mathcal{G}$ that start in a state $s \in S$ and

$\mathrm{Run}^{\mathcal{G}} = \bigcup_{s \in S} \mathrm{Run}^{\mathcal{G}}(s)$. A finite play $\sigma$ is a finite prefix of a play of $\mathcal{G}$. Let $\mathrm{Run}^{\mathcal{G}}_{\mathrm{fin}}(s)$ denote the set of all finite plays of $\mathcal{G}$ starting in a state $s \in S$ and $\mathrm{Run}^{\mathcal{G}}_{\mathrm{fin}} = \bigcup_{s \in S} \mathrm{Run}^{\mathcal{G}}_{\mathrm{fin}}(s)$.

The word induced by a play $\rho = s_0 s_1 \ldots \in \mathrm{Run}^{\mathcal{G}}$ is an infinite sequence $L(\rho) = L(s_0)L(s_1)\ldots \in (2^{AP})^{\omega}$. Similarly, a finite play of $\mathcal{G}$ induces a finite word $L(\sigma)$ in $(2^{AP})^{+}$.

**Remark 2.** Note that if $S = S_1$, or equivalently $S_2 = \emptyset$, the $2^1/_2$-player game $\mathcal{G}$ is an MDP, also called a $1^1/_2$-player game in game theory. Given a $2^1/_2$-player game $\mathcal{G}$, the $1^1/_2$-player interpretation of $\mathcal{G}$, where the players cooperate, is denoted as $\mathcal{G}^{\mathsf{coop}}$.

### Control strategy

**Definition 11.** *Let $\mathcal{G} = (S_1 \cup S_2, Act, P, AP, L)$ be a $2^1/_2$-player game. A Player 1 strategy for $\mathcal{G}$ is a function $C^1 \colon \mathrm{Run}^{\mathcal{G}}_{\mathrm{fin}} \cdot S_1 \to Act$ such that for every play $\sigma \in \mathrm{Run}^{\mathcal{G}}_{\mathrm{fin}} \cdot S_1$ of $\mathcal{G}$ it holds that $C^1(\sigma) \in Act(\mathrm{last}(\sigma))$. Strategies $C^2$ for Player 2 are defined analogously.*

A strategy $C^1$ for Player 1 for which $C^1(\sigma) = C^1(\sigma')$ for all finite plays $\sigma, \sigma' \in \mathrm{Run}^{\mathcal{G}}_{\mathrm{fin}}$ with $\mathrm{last}(\sigma) = \mathrm{last}(\sigma') \in S_1$ is called memoryless and we consider the strategy to be a function $C^1 \colon S_1 \to Act$. A strategy is called finite-memory if it is defined as a tuple $C^1 = (M, \mathsf{next}, \mathsf{trans}, \mathsf{start})$, where $M$ is a finite set of modes, $\mathsf{trans} \colon M \times S \to M$ is a transition function, $\mathsf{next} \colon M \times S_1 \to Act$ selects the next action to be applied in Player 1 states, and $\mathsf{start} \colon S \to M$ selects the starting mode for every $s \in S$. Memoryless and finite-memory strategies for Player 2 are defined analogously.

A play $\rho_{C^1,C^2} = s_0 s_1 \ldots \in \mathrm{Run}^{\mathcal{G}}$ of a game $\mathcal{G}$ is called a play under strategies $C^1, C^2$ for Player 1 and Player 2, respectively, if for every $i \geq 0$, it holds that $P(s_i, C(\rho^{\to i}_{C^1,C^2}), s_{i+1}) > 0$, where $C$ is the Player 1 strategy $C^1$ iff $s_i \in S_1$ and it is the Player 2 strategy $C^2$ otherwise. A finite play under $C^1, C^2$ is a finite prefix of a run under these strategies. The set of all infinite and finite plays of $\mathcal{G}$ under the two strategies starting in a state $s \in S$ are denoted by $\mathrm{Run}^{\mathcal{G},C^1,C^2}(s)$ and $\mathrm{Run}^{\mathcal{G},C^1,C^2}_{\mathrm{fin}}(s)$, respectively. Let $\mathrm{Run}^{\mathcal{G},C^1,C^2} = \bigcup_{s \in S} \mathrm{Run}^{\mathcal{G},C^1,C^2}(s)$ and finally $\mathrm{Run}^{\mathcal{G},C^1,C^2}_{\mathrm{fin}} = \bigcup_{s \in S} \mathrm{Run}^{\mathcal{G},C^1,C^2}_{\mathrm{fin}}(s)$.

### Probability measure

Analogously as for MDPs, we define a probability measure over sets of plays of a game. Let $\mathcal{G}$ be a $2^1/_2$-player game, $s$ a state of $\mathcal{G}$, and $C^1$ and $C^2$ strategy for Player 1 and Player 2, respectively.

Let $\sigma \in \mathrm{Run}^{\mathcal{G},C^1,C^2}_{\mathrm{fin}}(s)$ be a finite play. A cylinder set $\mathrm{Cyl}(\sigma)$ of $\sigma$ is the set of all plays of $\mathcal{G}$ under $C^1, C^2$ that have $\sigma$ as a finite prefix. There exists a unique

probability measure $\Pr_s^{\mathcal{G},C^1,C^2}$ on the $\sigma$-algebra generated by the set of cylinder sets of all runs in $\mathrm{Run}_{\mathrm{fin}}^{\mathcal{G},C^1,C^2}(s)$. For $\sigma =\in \mathrm{Run}_{\mathrm{fin}}^{\mathcal{G},C^1,C^2}(s)$, it holds

$$\Pr_s^{\mathcal{G},C^1,C^2}(\mathrm{Cyl}(\sigma)) = \prod_{i=0}^{n-1} P(\sigma(i), C(\sigma^{\to i}), \sigma(i+1)),$$

where $C$ is the Player 1 strategy $C^1$ iff $\sigma(i) \in S_1$ and it is the Player 2 strategy $C^2$ otherwise, and $\Pr_s^{\mathcal{G},C^1,C^2}(\mathrm{Cyl}(s)) = 1$.

Note that for the cooperative game $\mathcal{G}^{\mathsf{coop}}$ the probability measure matches the definition of the probability measure for MDPs.

#### Satisfaction of specification

Given a $2^1/_2$-player game $\mathcal{G}$ and an LTL formula $\phi$ over $AP$, we say that a play $\rho$ of $\mathcal{G}$ satisfies $\phi$, denoted as $\rho \models \phi$, iff it holds that $L(\rho) \models \phi$. The set of all runs under given strategies $C^1, C^2$ that satisfy an LTL formula form a set measurable in the probability measure $\Pr_s^{\mathcal{G},C^1,C^2}$ [BK08]. Strategies $C^1, C^2$ for Player 1 and Player 2 are said to satisfy $\phi$ almost-surely, or with probability 1, starting from a state $s \in S$ iff it holds that

$$\Pr_s^{\mathcal{G},C^1,C^2}(\{\rho \mid \rho \models \phi\}) = \Pr_s^{\mathcal{G},C^1,C^2}(\phi) = 1.$$

We use $\mathsf{Almost}^{\mathcal{G}}(\phi)$ to denote the almost-sure winning set in $\mathcal{G}$ for a formula $\phi$, i.e., the set of all states of the game such that Player 1 has a strategy to ensure satisfaction of the LTL formula $\phi$ with probability 1 irrespective of the strategy of Player 2. Formally,

$$\mathsf{Almost}^{\mathcal{G}}(\phi) = \{s \in S \mid \exists C^1 \, \forall C^2 : \Pr_s^{\mathcal{G},C^1,C^2}(\phi) = 1\}.$$

The almost-sure winning set in the cooperative game $\mathcal{G}^{\mathsf{coop}}$ for a formula $\phi$ is then defined as

$$\mathsf{Almost}^{\mathcal{G}^{\mathsf{coop}}}(\phi) = \{s \in S \mid \exists C^1 \, \exists C^2 : \Pr_s^{\mathcal{G},C^1,C^2}(\phi) = 1\}.$$

## 2.4 Dynamic Systems

In this section, we introduce systems that have infinite, continuous state and control spaces. Namely, we consider systems that have linear dynamics and include stochastic uncertainty. We start with several necessary definitions from geometry.

### 2.4.1 Polytopes

A *(convex) polytope* $\mathcal{X} \subset \mathbb{R}^N$ is defined as the convex hull of a finite set $X = \{x_i\}_{i \in I} \subset \mathbb{R}^N$:

$$\mathcal{X} = \mathrm{hull}(X) = \{\sum_{i \in I} \lambda_i x_i \mid \forall i : \lambda_i \in [0,1], \sum_{i \in I} \lambda_i = 1\}. \tag{2.3}$$

We use $V(\mathcal{X})$ to denote the vertices of $\mathcal{X}$ that is the minimum set of vectors in $\mathbb{R}^N$ for which $\mathcal{X} = \text{hull}(V(\mathcal{X}))$. Alternatively, a polytope can be defined as an intersection of a finite number of half-spaces in $\mathbb{R}^N$:

$$\mathcal{X} = \{x \in \mathbb{R}^N \mid H_{\mathcal{X}} x \leq K_{\mathcal{X}}\}, \tag{2.4}$$

where $H_{\mathcal{X}}, K_{\mathcal{X}}$ are matrices of appropriate sizes. Forms in Equations 2.3 and 2.4 are referred to as the V-representation and H-representation of polytope $\mathcal{X}$, respectively.

A polytope $\mathcal{X} \subset \mathbb{R}^N$ is called full-dimensional if it has at least $N+1$ vertices. In this work, we consider all polytopes to be full-dimensional, i.e., a polytope that is not full-dimensional is considered to be empty.

### 2.4.2 Linear Stochastic Systems

#### System and its runs

**Definition 12.** *A linear stochastic system $\mathcal{L}$ is defined as*

$$\mathcal{L} : x_{t+1} = Ax_t + Bu_t + w_t,$$

*where $x_t \in \mathcal{X} \subset \mathbb{R}^N, u_t \in \mathcal{U} \subset \mathbb{R}^M$ are column vectors, $\mathcal{X}$,$\mathcal{U}$ are polytopes in the corresponding Euclidean spaces called the state space and control space, respectively, column vector $w_t \in \mathcal{W} \subset \mathbb{R}^N$ is the value at time $t$ of a random vector with values in polytope $\mathcal{W}$. The random vector has positive density $f_{\mathcal{W}} \colon \mathcal{W} \to [0,1]$ on all values in $\mathcal{W}$. Finally, $A$ and $B$ are matrices of appropriate sizes.*

*The set of atomic propositions $AP$ is given as a finite set of linear predicates over the state space $\mathcal{X}$ of $\mathcal{L}$:*

$$AP = \{a : cx \leq d \mid c \in \mathbb{R}^N, d \in \mathbb{R}\},$$

*where $c$ is a row vector. A labeling function $L \colon \mathcal{X} \to 2^{AP}$ is defined as the set of all linear predicates that are true in a state $x$, i.e., $L(x) = \{a \in AP \mid cx \leq d\}$.*

We use $\mathcal{X}_{\mathsf{out}}$ to denote the set of all states in $\mathbb{R}^N$ outside of the state space $\mathcal{X}$ that can be reached within one step in system $\mathcal{L}$:

$$\mathcal{X}_{\mathsf{out}} = \{x \in \mathbb{R}^N \setminus \mathcal{X} \mid \exists x' \in \mathcal{X}, \exists u \in \mathcal{U}, \exists w \in \mathcal{W} : x = Ax' + Bu + w\}. \tag{2.5}$$

A run of a linear stochastic system $\mathcal{L}$ is an infinite sequence $\rho = x_0 x_1 \ldots \in \mathcal{X}^\omega$ such that for every $n \geq 0$, it holds that $x_{n+1} = Ax_n + Bu + w$ for some $u \in \mathcal{U}, w \in \mathcal{W}$. Moreover, every infinite sequence $x_0 \ldots x_{n-1}(x_n)^\omega$ such that $x_0 \ldots x_n \in \mathcal{X}^* \cdot \mathcal{X}_{\mathsf{out}}$ satisfies the above condition for every $0 \leq i \leq n-2$ is also considered to be a run of $\mathcal{L}$. We use $\text{Run}^{\mathcal{L}}(x)$ to denote the set of all runs of $\mathcal{L}$ that start in a state $x \in \mathcal{X}$ and $\text{Run}^{\mathcal{L}} = \bigcup_{x \in \mathcal{X}} \text{Run}^{\mathcal{L}}(x)$. A finite run $\sigma = x_0 \ldots x_n \in \mathcal{X}^+$ of $\mathcal{L}$ is a finite prefix of a run of $\mathcal{L}$. For $x \in \mathcal{X}$, $\text{Run}^{\mathcal{L}}_{\text{fin}}(x)$ denotes the set of all finite runs of $\mathcal{L}$ starting in $x$ and $\text{Run}^{\mathcal{L}}_{\text{fin}} = \bigcup_{x \in \mathcal{X}} \text{Run}^{\mathcal{L}}_{\text{fin}}(x)$.

Every run $\rho = x_0 x_1 \ldots$ of a linear stochastic system generates an infinite word $L(\rho) = L(x_0)L(x_1)\ldots \in (2^{AP})^\omega$ and similarly, every finite run $\sigma = x_0 \ldots x_n$ generates a finite word $L(\sigma) = L(x_0) \ldots L(x_n) \in (2^{AP})^+$.

### Control strategy

**Definition 13.** *Let $\mathcal{L}$ be a linear stochastic system. A control strategy for $\mathcal{L}$ is a function $C\colon \mathrm{Run}_{\mathrm{fin}}^{\mathcal{L}} \to \mathcal{U}$.*

A strategy $C$ for which $C(\sigma_1) = C(\sigma_2)$, for all finite runs $\sigma_1, \sigma_2 \in \mathrm{Run}_{\mathrm{fin}}^{\mathcal{L}}$ with $\mathrm{last}(\sigma_1) = \mathrm{last}(\sigma_2)$, is called memoryless. In that case, $C$ is a function $C\colon \mathcal{X} \to \mathcal{U}$. A strategy is called finite-memory if it can be defined as a tuple $C = (M, \mathsf{next}, \mathsf{trans}, \mathsf{start})$, where $M$ is a finite set of modes, $\mathsf{trans}\colon M \times \mathcal{X} \to M$ is a transition function, $\mathsf{next}\colon M \times \mathcal{X} \to \mathcal{U}$ selects the control input to be applied, and $\mathsf{start}\colon \mathcal{X} \to M$ selects the starting mode. A strategy that is not finite-memory is called infinite-memory.

A run induced by $C$ is a run $\rho_C = x_0 x_1 \ldots \in \mathrm{Run}^{\mathcal{L}}$ such that for every $i \geq 0$, there exists $w \in \mathcal{W}$ such that $x_{i+1} = Ax_i + BC(\rho_C^{\to i}) + w$. A finite run induced by $C$ is $\sigma_C \in \mathrm{Run}_{\mathrm{fin}}^{\mathcal{L}}$ that is a finite prefix of some $\rho_C$. The set of all infinite and finite runs of $\mathcal{L}$ under $C$ starting in a state $x \in \mathcal{X}$ are denoted by $\mathrm{Run}^{\mathcal{L},C}(x)$ and $\mathrm{Run}_{\mathrm{fin}}^{\mathcal{L},C}(x)$, respectively, and $\mathrm{Run}^{\mathcal{L},C} = \bigcup_{x\in\mathcal{X}} \mathrm{Run}^{\mathcal{L},C}(x)$ and $\mathrm{Run}_{\mathrm{fin}}^{\mathcal{L},C} = \bigcup_{x\in\mathcal{X}} \mathrm{Run}_{\mathrm{fin}}^{\mathcal{L},C}(x)$.

### Probability measure

Let $\mathcal{L}$ be a linear stochastic system of the form in Equation 12, $x \in \mathcal{X}$ a state of $\mathcal{L}$ and $C$ a strategy for $\mathcal{L}$.

Given a finite sequence $\mathcal{X}_1 \ldots \mathcal{X}_n$ of polytopes within the state space $\mathcal{X}$, a cylinder set $\mathrm{Cyl}(\mathcal{X}_1 \ldots \mathcal{X}_n)$ is the set of all runs $\rho = x x_1 \ldots x_n \ldots \in \mathrm{Run}^{\mathcal{L},C}(x)$ such that $x_i \in \mathcal{X}_i$ for all $1 \leq i \leq n$.

There exists a unique probability measure $\Pr_x^{\mathcal{L},C}$ on the $\sigma$-algebra generated by the set of all cylinder sets. For a sequence $\mathcal{X}_1 \ldots \mathcal{X}_n$, it holds

$$\Pr_x^{\mathcal{L},C}(\mathrm{Cyl}(\mathcal{X}_1 \ldots \mathcal{X}_n)) = \int_{x_1\in\mathcal{X}_1} \cdots \int_{x_{n-1}\in\mathcal{X}_{n-1}} P(x, dx_1)P(x_1, dx_2)\cdot\ldots\cdot P(x_{n-1}, \mathcal{X}_n),$$

where $P$ is the stochastic kernel of the linear stochastic system $\mathcal{L}$.

### Satisfaction of LTL formulas

A run $\rho$ of a linear stochastic system $\mathcal{L}$ satisfies an LTL formula $\phi$ over the set $AP$, denoted as $\rho \models \phi$, iff it holds that $L\rho \models \phi$. A strategy $C$ for $\mathcal{L}$ is said to satisfy $\phi$ almost-surely, or with probability 1, starting from a state $x \in \mathcal{X}$ if it holds that

$$\Pr_x^{\mathcal{L},C}(\{\rho \mid \rho \models \phi\}) = \Pr_x^{\mathcal{L},C}(\phi) = 1.$$

## 2.5 Automata-based LTL Model Checking and Control Synthesis

The problem of LTL model checking aims to answer the following question:

> *Given a system and an LTL formula, does it hold that all runs under all strategies satisfy the formula?*

For probabilistic systems, we consider the qualitative version of the problem:

> *Given a probabilistic system and an LTL formula, does it hold that the formula is satisfied with probability 1 under all strategies?*

Dually, the problem of LTL control synthesis is defined as follows:

> *Given a system and an LTL formula, does there exist a control strategy such that all runs under the strategy satisfy the formula?*

Finally, the qualitative LTL control synthesis poses the following question:

> *Given a probabilistic system and an LTL formula, does there exist a control strategy such that the formula is satisfied with probability 1 under the strategy?*

The automata-based approach to model checking relies on the correspondence between LTL formulas and $\omega$-automata. The main idea is to search for a run that violates the formula. For probabilistic systems, the aim is to find a control strategy that violates the formula with non-zero probability. If no such run or strategy exists, the model satisfies the formula. The general idea of the algorithm is as follows. First, the system is abstracted using a finite discrete model such as the ones introduced in Section 2.3. The formula is negated and translated to an $\omega$-automaton using techniques listed in Section 2.2. Next, a synchronous product of the model and the $\omega$-automaton is constructed, where the runs of the model that satisfy the negated formula can be easily identified through the accepting condition of the $\omega$-automaton. Finally, the product is systematically analyzed to find an accepting run of the product or a strategy that is accepting with non-zero probability. If it exists, the answer to the model checking problem is no and the found run or strategy serves as a counterexample for the formula satisfaction that can be further analyzed.

To use automata-based approach in control synthesis, the above algorithm can be applied with the following changes. First, the $\omega$-automaton is built for the LTL formula, not its negation. In the synchronous product, one aims to find a control strategy that induces only accepting runs. Analogously, for probabilistic systems, the strategy must induce an accepting run with probability 1. The constructed strategy is then the desired control strategy that guarantees satisfaction of the formula.

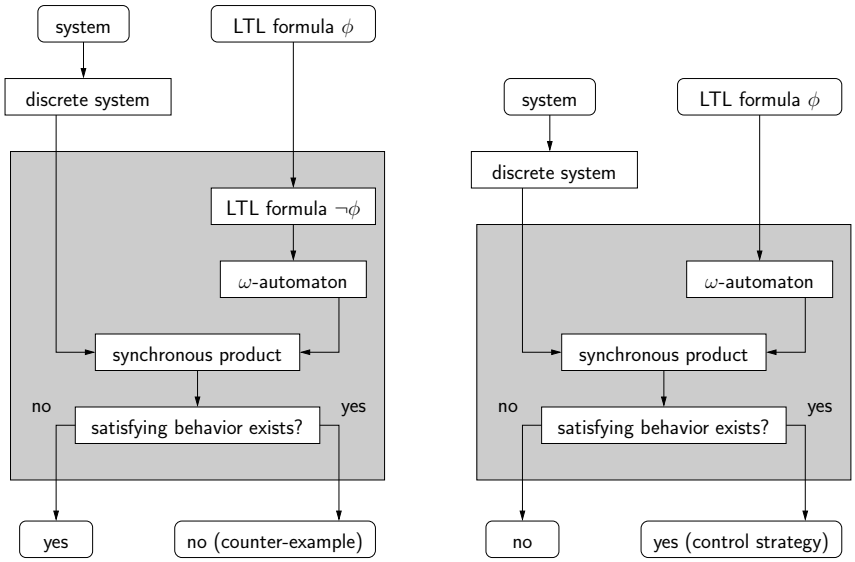



**Figure 2.1:** The outline of the automata-based approach to (a) LTL model checking and (b) LTL control synthesis.

We summarize the framework for the automata-based approach to LTL model checking and control synthesis in Figure 2.1. Based on the finite system at hand, an appropriate $\omega$-automaton is used to represent the formula and the algorithm to analyze the synchronous product explores the specific properties of both the system and the $\omega$-automaton. The corresponding algorithms for DTSs, MDPs and $2^1/_2$-player games appear in the following chapters and serve as the cornerstones for the designed techniques.

# Chapter 3

# Control for Finite Discrete Systems

In this chapter, we address a control problem for finite discrete models that is intuitively formulated as follows. Assume that every state of the system is associated with a (possibly changing) value. The values can be used to encode quantitative aspects of the system such as energy or time demands in given locations, or rewards associated with visits of particular states. Motivated by persistent surveillance missions for mobile robots, our goal is to optimize the expected cumulative value incurred between consecutive satisfactions of a property associated with some states of the system, while at the same time satisfying an additional temporal constraint in the form of an LTL formula. While the problems of optimal control and temporal logic control for finite discrete models are fairly well-studied, the connection between the two is an intriguing problem with a potentially high impact in applications.

In Section 3.1, we analyze the case, where the system is described using a DTS and the values are interpreted as rewards, i.e., the goal is to maximize the collected value. In Section 3.2, we follow with the dual problem, where the values are interpreted as penalties and construct strategies that while satisfying the LTL formula, minimize the expected collected value. Finally, in Section 3.3, a version of this problem is considered for MDPs. Every section starts with a motivation for the particular setup and a discussion on existing related work. We follow with the formal problem formulation and the description of the designed solution including illustrative examples for better understanding. All presented algorithms were implemented and their usability is demonstrated on case studies.

## 3.1 Deterministic Systems with Rewards

### 3.1.1 Motivation

In this section, we assume that the system is modeled using a deterministic transition system, as introduced in Definition 4. The task is to collect rewards that dynamically appear, disappear and change their values in the states of the DTS and that can only be sensed locally in close proximity from the current state. An example of such a system is a robot moving through a partitioned environment. The regions of the environment partition are states of the DTS and the robot can move between the regions using motion primitives such as "go straight", "go left" and so on. The rewards can represent benefits associated with visiting a particular region at a particular time. A traditional approach to this kind of problem, i.e., an optimization problem defined on a dynamically changing plant, is model predictive control (MPC) [RM09]. The method is based on iterative re-planning and optimization of a cost function over a finite time horizon and hence, it is also called receding horizon control.

In this work, we focus on interconnecting the receding horizon control with the synthesis of a strategy, or equivalently a run that is provably correct with respect to a given temporal logic formula referred to as a mission. This idea appeared in [WTM12c], where the receding horizon approach was employed to fight the high computational complexity of reactive motion planning with a specification in GR(1) fragment of LTL. However, the authors did not consider any reward collection to be optimized. In contrast, the authors in [DBC10] addressed a similar problem that we do. They assumed a deterministic transition system with locally sensed rewards changing according to an unknown dynamics. The problem addressed is to design a control strategy that (1) guarantees the satisfaction of the mission and (2) locally maximizes the collected rewards. These two goals often cannot be reached simultaneously. If the system primarily collects high rewards, the mission might never be satisfied and vice versa, if the system is controlled to accomplish the mission, the collected rewards might become low. The authors utilized ideas from the automata-based approach to model checking in order to iteratively find a local path maximizing the collected rewards among the local paths that ensure that a step towards the mission satisfaction is made. This way, they managed to compromise between the two goals.

Our work can be seen as a different, generalized approach to the above problem. In particular, we assume an LTL mission that includes persistent surveillance of a set of states and a user-defined *preference function* expressing the desired trade-off between the surveillance and the reward collection given the history of the system execution. In other words, the preference function determines in each moment whether moving towards a surveyed place or optimization of the collected rewards is of a higher priority. Whereas the local path planned in [DBC10] always guarantees progress towards the satisfaction of the mission, in our case this progress may be deliberately postponed (for a finite amount of time) if the

collection of the rewards is prioritized. For example, consider a garbage truck that is required to periodically visit two garbage disposal plants $A$ and $B$ and to arrive to a plant as fully loaded as possible. In [DBC10], each local plan for the truck would send the truck closer to $A$ (or $B$, respectively) and the truck might arrive half-empty. In contrast, through the preference function, we can define that collecting the garbage is the primary target until the truck is full enough to drive to a plant and that once it is, driving towards $A$ (or $B$, respectively) becomes the priority. Besides that, we generalize the problem from [DBC10] in the following sense. The authors there assumed that the reward dynamics is completely unknown. Therefore, when planning, they estimate that the rewards collected along a planned local path would be equal to the sum of the rewards that are currently seen at the states of this path and they aim to maximize it. We consider arbitrary reward dynamics that might be unknown, known partially or even fully. We capture the concrete reward dynamics assumptions through a so-called *state potential function*. Through these user-defined functions, we also allow for a broad class of optimization objectives that aim at optimizing reward collection over a finite time horizon such as the one discussed for the garbage truck. The problem we address is to design a control strategy that (1) guarantees the satisfaction of the formula, (2) locally optimizes the collection of rewards, and (3) takes into consideration the preference function and the reward dynamics assumptions.

In our solution, we leverage ideas from the automata-based approach to model-checking to provably guarantee the satisfaction of the mission and we introduce several extensions that allow us to support both the preference function and the arbitrary assumptions on the reward dynamics. We build a synchronous product of the DTS and an $\omega$-automaton for the LTL mission that captures all the runs of the system that satisfy the mission. We employ the preference function to compute the *attraction* of states in the product and at each time, we choose the most attractive state to be visited next. While the value of the preference function is low, the system is primarily driven by the sensed rewards. However, as the preference function grows, the surveillance is prioritized and the attraction forces the system to move not only towards the surveyed regions, but also towards accepting states of the product, i.e., towards the satisfaction of the global specification.

The results presented in this section are based on our results in [STBv12]. The rest of the section is organized as follows. In Section 3.1.2, the problem is described in detail and stated formally. The designed solution is presented in Section 3.1.3. Finally, an illustrative case study is presented in Section 3.1.4.

### 3.1.2 Problem Formulation

Let $\mathcal{T} = (S, T, AP, L)$ be an initialized DTS with initial state $s_{\mathsf{init}} \in S$, see Definition 4. Let

$$w \colon T \to \mathbb{R}^+ \tag{3.1}$$

be a weight function. With a slight abuse of notation, we use $w(\sigma)$ to denote the weight of a finite run $\sigma = s_0 \dots s_n$ of the DTS, i.e., $w(\sigma) = \sum_{i=0}^{n-1} w((s_i, s_{i+1}))$. Moreover, $w^*(s, s')$ denotes the minimum weight of a finite run from $s$ to $s'$. Specifically, $w^*(s, s) = 0$ for every $s \in S$ and if there does not exist a run from $s$ to $s'$, then $w^*(s, s') = \infty$. For a set $S' \subseteq S$ we let $w^*(s, S') = \min_{s' \in S'} w^*(s, s')$. We say that a state $s'$ and a set $S'$ is reachable from $s$ if and only if $w^*(s, s') \neq \infty$ and $w^*(s, S') \neq \infty$, respectively.

We interpret the weight $w((s, s'))$ of a transition $(s, s') \in T$ as the amount of time that this transition takes. That means, if the system is in a state $s$ at time $t$ and follows the transition $(s, s')$ then it is in the state $s'$ at time $t + w((s, s'))$. The time spent in states is 0. Associated with a run $\rho = s_0 s_1 \dots \in \mathrm{Run}^{\mathcal{T}}$ and a finite run $\sigma = s_0 \dots s_n \in \mathrm{Run}^{\mathcal{T}}_{\mathrm{fin}}$ there is a sequence of time instances $t_0 t_1 \dots$ and $t_0 \dots t_n$, respectively, where $t_0 = 0$ and $t_i$ denotes the time at which the state $s_i$ is reached, i.e., $t_{i+1} = t_i + w((s_i, s_{i+1}))$.

We assume a *visibility range* $v \in \mathbb{R}_{>0}, v \geq \max_{(s,s') \in T} w(s, s')$ is given and we use $\mathrm{Vis}(s) = \{s' \mid w^*(s, s') \leq v\}$ to denote the set of states that are within the visibility range $v$ from a state $s \in S$. Let

$$\mathrm{rew} : S \times \mathrm{Run}^{\mathcal{T}}_{\mathrm{fin}} \to \mathbb{R}_{\geq 0}$$

be the *reward function*, where $\mathrm{rew}(s, s_0 \dots s_k)$ is the reward sensed in a state $s \in S$ at time $t_k$ after executing a finite run $s_0 \dots s_k \in \mathrm{Run}^{\mathcal{T}}_{\mathrm{fin}}$. Note that $\mathrm{rew}(s, s_0 \dots s_k)$ is defined if and only if $s \in \mathrm{Vis}(s_k)$ and it is known only at time $t_k$ (and later), not earlier.

A user-defined *planning horizon* and a *state potential function* are employed to capture user's assumptions about the reward dynamics and her interests. For instance, the values of the rewards may increase or decrease at most by 1 during 1 time unit, they may appear according to a probabilistic distribution, or their changes might be random. The user might have full, partial or no knowledge of the reward dynamics. The reward might disappear once it are collected at the state, or it might not. The user might be interested in, e.g., the maximal, expected, or minimal sum of rewards that can be collected from a given state during a finite run whose weight is no more than the planning horizon. The concrete definitions of the planning horizon and the state potential function are meant to be specifically tailored for different cases. Formally, the horizon is $h \in \mathbb{R}_{>0}$, $h \geq \max_{(s,s') \in T} w(s, s')$ and the state potential function is

$$\mathrm{pot} : S \times \mathrm{Run}^{\mathcal{T}}_{\mathrm{fin}} \times \mathbb{R}_{>0} \to \mathbb{R}_{\geq 0}, \tag{3.2}$$

where $\mathrm{pot}(s, s_0 \dots s_k, h)$, is the potential of the state $s$ at time $t_k$ assuming that $s_0 \dots s_k$ is the so far executed run of the system. It is defined for all $s$ such that $(s_k, s) \in T$ and it specifies the optimal rewards that can be collected during a

finite run $\sigma \in \mathrm{Run}^{\mathcal{T}}_{\mathrm{fin}}(s, s_k, h)$, where

$$\begin{aligned}\mathrm{Run}^{\mathcal{T}}_{\mathrm{fin}}(s, s_k, h) = \{\sigma \mid &\text{(i) } \sigma \in \mathrm{Run}^{\mathcal{T}}_{\mathrm{fin}}(s),\\ &\text{(ii) } w((s_k, s)) + w(\sigma) \leq h,\\ &\text{(iii) the states that appear in } \sigma \text{ belong to } \mathrm{Vis}(s_k)\}.\end{aligned}$$

Note, that the visibility range $v$ and the planning horizon $h$ are independent. While $v$ determines the set of states whose rewards are visible from the current state $s_k$, $h$ gives the maximal total weight of a planned finite run $\sigma$ within $\mathrm{Vis}(s_k)$, which can be even greater than $v$.

**Example 1.** The function stating that the potential of $s$ is the maximal sum of rewards that can be collected from $s$ assuming that the rewards do not change while the system can sense them and that they disappear once collected is

$$\mathrm{pot}(s, s_0 \dots s_k, h) = \max_{\sigma \in \mathrm{Run}^{\mathcal{T}}_{\mathrm{fin}}(s, s_k, h)} \sum_{s' \in \sigma} \mathrm{rew}(s', s_0 \dots s_k).$$

In fact, this is how authors in [DBC10] estimate the amount of rewards collected on a local path.

To define our problem, we assume that the specification is given as an LTL formula with persistent surveillance as defined in Section 2.2.4. Recall that a formula with persistent surveillance is

$$\phi = \varphi \wedge \mathbf{GF}\, a_{\mathsf{sur}},$$

where $\varphi$ is an arbitrary LTL formula over $AP$ and $a_{\mathsf{sur}} \in AP$ is a surveillance proposition. The formula states that the system must satisfy a temporal constraint $\varphi$ and at the same time, it must infinitely many times survey states $S_{\mathsf{sur}} = \{s_{\mathsf{sur}} \mid a_{\mathsf{sur}} \in L(s_{\mathsf{sur}})\}$ labeled with proposition $a_{\mathsf{sur}}$.

The user can partially guide whether the system should collect high rewards or whether it should rather make a step towards the satisfaction of the surveillance proposition $a_{\mathsf{sur}}$ through a *preference function*. For example, the preference function can grow linearly with time since the latest visit to $a_{\mathsf{sur}}$, meaning that going towards $a_{\mathsf{sur}}$ gradually gains more importance. In contrast, the value of the preference function can stay low until the latest visit to $a_{\mathsf{sur}}$ happened no later than 100 time units ago and after that increase rapidly, expressing that the system is preferred to collect rewards for 100 time units and then to move towards $a_{\mathsf{sur}}$ quickly.

Formally, the preference function

$$\mathrm{pref} : \mathrm{Run}^{\mathcal{T}}_{\mathrm{fin}} \to \mathbb{R}_{\geq 0} \tag{3.3}$$

assigns a non-negative real value to every executed finite run $s_0 \dots s_k$ of $\mathcal{T}$ (possibly) taking into account the current values of the state potential function.

**Example 2.** An example of a preference function is

$$\mathrm{pref}(s_0 \dots s_k) = 0.01 \cdot w_i \cdot \max_{(s_k,s)\in T} \mathrm{pot}(s, s_0 \dots s_k, h),$$

where $w_i = w(s_i \dots s_k)$ such that $a_{\mathsf{sur}} \in L(s_i)$ and $a_{\mathsf{sur}} \notin L(s_j)$, for all $i < j \leq k$. If the surveyed state is being avoided, the total weight (or time) $w_i$ since the last visit to a surveyed state gradually grows and eventually, the value of $\mathrm{pref}(s_0 \dots s_k)$ overgrows the value of $\mathrm{pot}(s, s_0 \dots s_k, h)$ for all $s$.

A *shortening indicator function* $I_{S_{\mathsf{sur}}} \colon T \to \{0, 1\}$ indicates whether a transition leads the system closer to a state subject to surveillance, i.e.,

$$I_{S_{\mathsf{sur}}}((s, s')) = \begin{cases} 1 & \text{if } \min\limits_{s_{\mathsf{sur}}\in S_{\mathsf{sur}}} w^*(s', s_{\mathsf{sur}}) < \min\limits_{s_{\mathsf{sur}}\in S_{\mathsf{sur}}} w^*(s, s_{\mathsf{sur}}), \\ 0 & \text{otherwise,} \end{cases} \tag{3.4}$$

where $(s, s') \in T$.

**Problem 1.** ***Given***

- *an initialized DTS* $\mathcal{T} = (S, T, AP, L)$ *with initial state* $s_{\mathsf{init}} \in S$,
- *a weight function* $w$,
- *an LTL formula* $\phi$ *over* $AP$ *with persistent surveillance,*
- *rewards* $\mathrm{rew}(s, s_0 \dots s_k)$ *at time* $t_k$*, for all* $s \in \mathrm{Vis}(s_k)$,
- *a visibility range* $v \in \mathbb{R}_{>0}, v \geq \max_{(s,s')\in T} w(s, s')$,
- *a planning horizon* $h \in \mathbb{R}_{>0}, h \geq \max_{(s,s')\in T} w(s, s')$,
- *a state potential function* pot,
- *a preference function* pref,

***find*** *a control strategy* $C \colon \mathrm{Run}^{\mathcal{T}}_{\mathrm{fin}} \to S$ *such that*

*(i)* $C$ *satisfies* $\phi$ *starting from* $s_{\mathsf{init}}$ *and*

*(ii) assuming that* $s = C(s_0 \dots s_k)$*, the cost function*

$$\mathrm{pot}(s, s_0 \dots s_k, h) + I_{S_{\mathsf{sur}}}((s_k, s)) \cdot \mathrm{pref}(s_0 \dots s_k) \tag{3.5}$$

*is maximized at each time* $t_k$.

Intuitively, condition (ii) is interpreted as follows. At each time, the aim is to go to the state with the best trade-off between the amount of potentially collected rewards and the importance of fast surveillance. The higher the value of the preference function, the more likely a state closer to $a_{\mathsf{sur}}$ is chosen. Note that, in general, the satisfaction of the condition (ii) may cause violation of the objective (i). Our goal is thus to provably guarantee satisfaction of the formula and to maximize the value in Equation 3.5 if possible.

Our solution to Problem 1 consists of two consecutive steps. The first step involves computation performed before the system starts its execution and the second step is an online receding horizon control algorithm that is employed during the system's execution. In the first step, we construct a Büchi automaton for the

LTL formula $\phi$ and its synchronous product with the DTS. We assign two Boolean indicators to each transition of the product which indicate whether the transition induces a progress towards a surveyed state of the DTS and both a surveyed state and an accepting state of the BA, respectively. The receding horizon algorithm computes the control strategy in real time, i.e., it determines the next state to be visited by the system in every step of the system's execution. In every iteration, attractions of the states of the product are computed. The repeated choices of the maximal attraction states lead to an eventual visit not only to a surveyed state, but also to an accepting state of the BA, assuming that the following holds:

**Assumption 1.** For every run $s_0 s_1 \ldots \mathrm{Run}^{\mathcal{T}}$ with the property that there exists $n_1$ such that for every $m > n_1$ it holds $a_{\mathsf{sur}} \notin L(s_m)$, it holds that there exists $n_2$ so that for every $m > n_2$ it holds $\mathrm{pref}(s_0 \ldots s_m) > \mathrm{pot}(s, s_0 \ldots s_m, h)$ for all $s$, where $(s_m, s) \in T$.

Intuitively, the assumption states that if a visit to a surveyed state is postponed for a long time, the value of the preference function overweights the value of the state potentials. Note that this is, in fact, quite natural. It only captures the fact, that the user who defines the potential and the preference function wishes to satisfy the LTL formula in long term and therefore the interest in making a progress towards the satisfaction of the formula at some point naturally prevails the interest in collecting the rewards.

### 3.1.3 Problem Solution

In this section, we give the details of our solution to Problem 1. We start with the description of the pre-computation step. We define the product automaton and describe how the two Boolean indicators for transitions of the product are computed. We follow with the description of the online receding horizon control algorithm. Finally, we discuss the properties and optimality of the designed approach.

#### Product construction

To employ the automata-based approach to control synthesis as described in Section 2.5, we first translate the LTL formula $\phi$ to a non-deterministic Büchi automaton $\mathcal{A}$ using techniques discussed in Section 2.2. Next, we construct the synchronous product of the DTS $\mathcal{T}$ and the BA $\mathcal{A}$.

**Definition 14.** *A* product *of a DTS $\mathcal{T} = (S, T, AP, L)$ with initial state $s_{\mathsf{init}} \in S$ and a non-deterministic BA $\mathcal{A} = (Q, 2^{AP}, \delta, q_0, F)$ is a tuple*

$$\mathcal{P} = \mathcal{T} \times \mathcal{A} = (S_{\mathcal{P}}, T_{\mathcal{P}}, s_{\mathcal{P}\mathsf{init}}, AP, L_{\mathcal{P}}, F_{\mathcal{P}}),$$

*where $S_{\mathcal{P}} = S \times Q$ is a set of states, $T_{\mathcal{P}} \subseteq S_{\mathcal{P}} \times S_{\mathcal{P}}$ is a transition relation, where $\big((s,q),(s',q')\big) \in T_{\mathcal{P}}$ if and only if $(s,s') \in T$ and $(q, L(s), q') \in \delta$, $s_{\mathcal{P}\mathsf{init}} = (s_{\mathsf{init}}, q_0)$*

*is the initial state,* $L_{\mathcal{P}}\colon S_{\mathcal{P}} \to 2^{AP}$ *is a labeling function such that* $L_{\mathcal{P}}((s,q)) = L(s)$ *for every* $(s,q) \in S_{\mathcal{P}}$*, and* $F_{\mathcal{P}} = S \times F$ *is the set of accepting states.*

We naturally extend the weight function $w$ of the DTS $\mathcal{T}$ to a weight function $w_{\mathcal{P}}\colon T_{\mathcal{P}} \to \mathbb{R}^+$ for the product $\mathcal{P}$ such that $w_{\mathcal{P}}\big(\big((s,q),(s',q')\big)\big) = w((s,s'))$ for every $((s,q),(s',q)) \in T_{\mathcal{P}}$.

The product $\mathcal{P}$ can be viewed as an initialized TS with a set of accepting states. Therefore, we adopt the definitions of a run $\rho$, a finite run $\sigma$, its weight $w_{\mathcal{P}}(\sigma)$, and sets $\mathrm{Run}^{\mathcal{P}}((s,q))$, $\mathrm{Run}^{\mathcal{P}}$, $\mathrm{Run}^{\mathcal{P}}_{\mathrm{fin}}((s,q))$ and $\mathrm{Run}^{\mathcal{P}}_{\mathrm{fin}}$ from Section 2.3.1. Similarly, a cycle $\mathsf{cyc}$ of $\mathcal{P}$, a strategy $C_{\mathcal{P}}$ for $\mathcal{P}$ and runs $\rho_{C_{\mathcal{P}}}, \sigma_{C_{\mathcal{P}}}$ induced by $C_{\mathcal{P}}$ are defined in the same way as for a DTS. We also adopt the definitions of a sub-system and a strongly connected component. On the other hand, $\mathcal{P}$ can be viewed as a weighted BA over the trivial alphabet with a labeling function, which gives us the definition of an accepting run of $\mathcal{P}$.

Every run $\rho$ and finite run $\sigma$ of $\mathcal{P}$ projects to a run $\pi_1(\rho)$ and a finite run $\pi_1(\sigma)$ of $\mathcal{T}$, respectively. Vice versa, for every run $s_0 s_1 \dots$ and finite run $s_0 \dots s_n$ of $\mathcal{T}$, there exists a run $(s_0,q_0)(s_1,q_1)\dots$ and finite run $(s_0,q_0)\dots(s_n,q_n)$. Similarly, every strategy for $\mathcal{P}$ projects to a strategy for $\mathcal{T}$ and for every strategy for $\mathcal{T}$ there exists a strategy for $\mathcal{P}$ that projects to it. The projection of a finite-memory strategy for $\mathcal{P}$ is also finite-memory.

**Definition 15.** *Let* $\mathcal{P} = (S_{\mathcal{P}}, T_{\mathcal{P}}, s_{\mathcal{P}init}, AP, L_{\mathcal{P}}, F_{\mathcal{P}})$ *be the product of an initialized DTS* $\mathcal{T}$ *and a BA* $\mathcal{A}$*. An accepting strongly connected component (ASCC) of* $\mathcal{P}$ *is an SCC* $\mathcal{U} = (S_{\mathcal{U}}, T_{\mathcal{U}}, AP, L_{\mathcal{P}})$ *such that the set* $S_{\mathcal{U}} \cap F_{\mathcal{P}}$ *is nonempty and we refer to it as the set* $F_{\mathcal{U}}$ *of accepting states of* $\mathcal{U}$*. We use* $\mathrm{ASCC}(\mathcal{P})$ *to denote the set of all ASCCs of* $\mathcal{P}$ *that are reachable from the initial state* $s_{\mathcal{P}init}$*.*

### Boolean indicator assignment

In this section, we define two Boolean indicators for every transition of the product $\mathcal{P}$ of the DTS and the BA for the formula $\phi$ that will be used to compute the control strategy during the online control algorithm.

Let $S_{\mathcal{P}\mathsf{sur}} = \{(s,q) \in S_{\mathcal{P}} \mid a_{\mathsf{sur}} \in L(s)\}$ denote the subset of states of $\mathcal{P}$ that project onto the surveyed states in $\mathcal{T}$. Furthermore, let $F^{\infty}_{\mathcal{P}} \subseteq F_{\mathcal{P}}$ and $S^{\infty}_{\mathcal{P}\mathsf{sur}} \subseteq S_{\mathcal{P}\mathsf{sur}}$ be the sets of states from which $F_{\mathcal{P}}$ and $S_{\mathcal{P}\mathsf{sur}}$ can be visited infinitely many times, respectively. Sets $F^{\infty}_{\mathcal{P}}$ and $S^{\infty}_{\mathcal{P}\mathsf{sur}}$ can be computed iteratively as the maximal sets of states from which a state in $S^{\infty}_{\mathcal{P}\mathsf{sur}}$ and $F^{\infty}_{\mathcal{P}}$ is reachable via a finite run of nonzero length, respectively, see Algorithm 1, lines 4-11.

**Lemma 1.** A run $\rho$ of the product $\mathcal{P}$ is accepting if and only if it holds that $F^{\infty}_{\mathcal{P}} \cap \inf(\rho) \neq \emptyset$ and $S^{\infty}_{\mathcal{P}\mathsf{sur}} \cap \inf(\rho) \neq \emptyset$.

*Proof.* Let $\rho$ be an accepting run of $\mathcal{P}$, i.e., $F_{\mathcal{P}} \cap \inf(\rho) \neq \emptyset$. Note that there is a state in $S_{\mathcal{P}\mathsf{sur}}$ that appears in $\rho$ infinitely many times, because $\rho$ satisfies $\phi$ and hence also $\mathbf{GF}\, a_{\mathsf{sur}}$. Then there exist infinite index sets $I, J \subseteq \mathbb{N}$ such

that $\rho(i) \in F_{\mathcal{P}}$ and $\rho(j) \in S_{\mathcal{P}_{\mathsf{sur}}}$ for every $i \in I$ and $j \in J$ . For each state $\rho(i) \in F_{\mathcal{P}}, i \in I$ there exist infinitely many states $\rho(j) \in S_{\mathcal{P}_{\mathsf{sur}}}$ such that $i < j \in J$, and analogous holds for each state $\rho(j) \in S_{\mathcal{P}_{\mathsf{sur}}}, j \in J$. Hence, all states $\rho(i), \rho(j)$ for $i \in I, j \in J$ belong to $F^{\infty}_{\mathcal{P}}, S^{\infty}_{\mathcal{P}_{\mathsf{sur}}}$, respectively. On the other hand, if a state from $F^{\infty}_{\mathcal{P}}$ occurs on $\rho$ infinitely many times, then $\rho$ is clearly accepting. □

For each state $u = (s, q) \in S_{\mathcal{P}}$ we define the minimum weight of a finite run from $u$ to a state from $S^{\infty}_{\mathcal{P}_{\mathsf{sur}}}$

$$w^*_{\mathcal{P}_{\mathsf{sur}}}(u) = \min_{u' \in S^{\infty}_{\mathcal{P}_{\mathsf{sur}}}} w^*_{\mathcal{P}}(u, u') \tag{3.6}$$

and the minimum weight of a finite run from $u$ to $S^{\infty}_{\mathcal{P}_{\mathsf{sur}}}$ containing a state $u' \in F^{\infty}_{\mathcal{P}}$

$$w^*_{\mathcal{P}F_{\mathsf{sur}}}(u, u') = \min_{u'' \in S^{\infty}_{\mathcal{P}_{\mathsf{sur}}}} \big(w^*_{\mathcal{P}}(u, u') + w^*_{\mathcal{P}}(u', u'')\big). \tag{3.7}$$

Moreover, we define

$$w^*_{\mathcal{P}_\phi}(u) = \big(w^*_{\mathcal{P}}(u, u'), w^*_{\mathcal{P}F_{\mathsf{sur}}}(u, u')\big) \tag{3.8}$$

where $u' \in F^{\infty}_{\mathcal{P}}$ minimizes $w^*_{\mathcal{P}}(u, u')$ among the set of states that minimize Equation 3.7. Given $w^*_{\mathcal{P}_\phi}(u_1) = (x_1, y_1)$ and $w^*_{\mathcal{P}_\phi}(u_2) = (x_2, y_2)$, $w^*_{\mathcal{P}_\phi}(u_1) < w^*_{\mathcal{P}_\phi}(u_2)$ if and only if $x_1 < x_2$ and $y_1 < y_2$.

Note that each state $u \in S_{\mathcal{P}}$ with $w^*_{\mathcal{P}_{\mathsf{sur}}}(u) = \infty$ or $w^*_{\mathcal{P}_\phi}(u) = (\infty, \infty)$ cannot occur on any accepting run of $\mathcal{P}$. Therefore, we assume from now on that $\mathcal{P}$ contains only states $u \in S_{\mathcal{P}}$ with $w^*_{\mathcal{P}_{\mathsf{sur}}}(u) \neq \infty$ and $w^*_{\mathcal{P}_\phi}(u) \neq (\infty, \infty)$.

**Lemma 2.** $\forall u \in S_{\mathcal{P}} \setminus S^{\infty}_{\mathcal{P}_{\mathsf{sur}}}, \exists u' \in S_{\mathcal{P}} : (u, u'), \in T_{\mathcal{P}}$, $w^*_{\mathcal{P}_{\mathsf{sur}}}(u) > w^*_{\mathcal{P}_{\mathsf{sur}}}(u')$, and $\forall u \in S_{\mathcal{P}} \setminus F^{\infty}_{\mathcal{P}}, \exists u' \in S_{\mathcal{P}} : (u, u'), \in T_{\mathcal{P}}, w^*_{\mathcal{P}_\phi}(u) > w^*_{\mathcal{P}_\phi}(u')$.

*Proof.* Follows directly from Equations 3.6, 3.7 and 3.8. □

We are now ready to define the *shortening indicator functions* $I_{\mathcal{P}_{\mathsf{sur}}}, I_{\mathcal{P}_\phi} \colon T_{\mathcal{P}} \to \{1, 0\}$, which indicate whether a transition induces progress towards the set $S^{\infty}_{\mathcal{P}_{\mathsf{sur}}}$ and towards both the set $F^{\infty}_{\mathcal{P}}$ and the set $S^{\infty}_{\mathcal{P}_{\mathsf{sur}}}$ via a state in $F^{\infty}_{\mathcal{P}}$, respectively. Formally, we let

$$I_{\mathcal{P}x}((u, u')) = \begin{cases} 1 & \text{if } w^*_{\mathcal{P}_x}(u) > w^*_{\mathcal{P}_x}(u'), \\ 0 & \text{otherwise,} \end{cases} \tag{3.9}$$

where $x \in \{\mathsf{sur}, \phi\}$.

**Corollary 1.** $\forall u \in S_{\mathcal{P}} \setminus S^{\infty}_{\mathcal{P}_{\mathsf{sur}}}, \exists (u, u') \in T_{\mathcal{P}}$ such that $I_{\mathcal{P}_{\mathsf{sur}}}((u, u')) = 1$ and $\forall u \in S_{\mathcal{P}} \setminus F^{\infty}_{\mathcal{P}}, \exists (u, u') \in T_{\mathcal{P}}$ such that $I_{\mathcal{P}_\phi}((u, u')) = 1$.

The outline of the indicator assignment procedure for the product $\mathcal{P}$ is summarized in Algorithm 1.

**Algorithm 1** Boolean indicator assignment

Input: $\mathcal{P} = \mathcal{T} \times \mathcal{A} = (S_\mathcal{P}, T_\mathcal{P}, s_{\mathcal{P}\mathsf{init}}, AP, L_\mathcal{P}, F_\mathcal{P}), w_\mathcal{P}$
Output: $I_{\mathcal{P}\mathsf{sur}}, I_{\mathcal{P}\phi}$
$F^\infty_\mathcal{P} := F_\mathcal{P}, S^\infty_{\mathcal{P}\mathsf{sur}} := S_{\mathcal{P}\mathsf{sur}}$
**while** fix-point of $F^\infty_\mathcal{P}, S^\infty_{\mathcal{P}\mathsf{sur}}$ not found **do**
    **for all** $u \in F^\infty_\mathcal{P}$, s.t. $\min_{(u,u')\in T_\mathcal{P}, u''\in S^\infty_{\mathcal{P}\mathsf{sur}}} w^*_\mathcal{P}(u', u'') = \infty$ **do**
        remove $p$ from $F^\infty_\mathcal{P}$
    **end for**
    **for all** $u \in S^\infty_{\mathcal{P}\mathsf{sur}}$, s.t. $\min_{(u,u')\in T_\mathcal{P}, u''\in F^\infty_\mathcal{P}} w^*_\mathcal{P}(u', u'') = \infty$ **do**
        remove $u$ from $S^\infty_{\mathcal{P}\mathsf{sur}}$
    **end for**
**end while**
**for all** $u \in S_\mathcal{P}$, s.t. $w^*_{\mathcal{P}\mathsf{sur}}(u) = \infty \vee w^*_{\mathcal{P}\phi}(u) = (\infty, \infty)$ **do**
    remove $u$ together with incident transitions
**end for**
**for all** $(u, u') \in T_\mathcal{P}$ **do**
    compute $I_{\mathcal{P}\mathsf{sur}}((u, u')), I_{\mathcal{P}\phi}((u, u'))$ according to Equation 3.9
**end for**

### Online control

The online control algorithm computes the control strategy for Problem 1 in real time. At time $t_k$ such that $s_0 \dots s_k$ is the so far executed finite run of the DTS $\mathcal{T}$ the algorithm determines the next state $C(s_0 \dots s_k)$ of $\mathcal{T}$ to be visited. Simply put, we compute the next state in the product $\mathcal{P}$ and then we project it onto $\mathcal{T}$. Formally, $\mathcal{T}$ starts in its initial state $s_{\mathsf{init}}$ and $\mathcal{P}$ in its initial state $s_{\mathcal{P}\mathsf{init}} = (s_{\mathsf{init}}, q_0)$. For each finite run $u_0 \dots u_k$ of $\mathcal{P}$, the algorithm computes the next state of $\mathcal{P}$ denoted by $C_\mathcal{P}(u_0 \dots u_k)) = u_{k+1}$. The next state of $\mathcal{T}$ is then $C(s_0 \dots s_k) = \pi_1(u_{k+1}) = s_{k+1}$.

To guarantee that the control strategy $C$ generates a run of $\mathcal{T}$ satisfying $\phi$, it is sufficient to ensure that the control strategy $C_\mathcal{P}$ generates a run of $\mathcal{P}$ that visits $F_\mathcal{P}$ infinitely many times. In $\mathcal{T}$, the high value of the preference function pref was used to guide the system towards $\mathsf{sur}$. Projected into the product, the high value of pref can "send" the system towards a state in $S^\infty_{\mathcal{P}\mathsf{sur}}$. We expand this idea and use the preference function to guide the robot not only towards $S^\infty_{\mathcal{P}\mathsf{sur}}$, but also towards $F^\infty_\mathcal{P}$. This way, we ensure that $F^\infty_\mathcal{P}$ is indeed visited infinitely many times.

In particular, we introduce two subgoals in $\mathcal{P}$. The first one is the *mission subgoal*, when a visit to $F^\infty_\mathcal{P}$ is targeted. The second one is the *surveillance subgoal*, when we aim to visit $S^\infty_{\mathcal{P}\mathsf{sur}}$. At each time, one of the subgoals is to be achieved and once it is, the subgoals are switched and the other one is to be achieved. Progress towards both subgoals is governed by maximization of the *attraction function* attr that is defined for the product automaton in an analogous way as the cost function in Equation 3.5 for Problem 1.

Assume, that $\phi$ is satisfiable, i.e., that $F^\infty_\mathcal{P}$ and $S^\infty_{\mathcal{P}\mathsf{sur}}$ computed in Algorithm 1 are both non-empty and $s_{\mathcal{P}\mathsf{init}} \in F^\infty_\mathcal{P}$. The product $\mathcal{P}$ naturally inherits the rewards from $\mathcal{T}$, i.e., $\mathrm{rew}_\mathcal{P}((s,q),(s_0,q_0)\dots(s_k,q_k)) = \mathrm{rew}(s,s_0\dots s_k)$. Thus, the value of pot function can be computed on the product using $\mathrm{rew}_\mathcal{P}$. We use $\mathrm{pot}_\mathcal{P}(u,u_0\dots u_k,h)$ to denote the value of the state potential function for a state $u$ computed on $\mathcal{P}$.

The value of the attraction

$$\mathrm{attr} : S_\mathcal{P} \times \mathrm{Run}^\mathcal{P}_{\mathrm{fin}} \times \mathbb{R}_{>0} \to \mathbb{R}_{\geq 0}$$

is computed differently for the two subgoals. Initially, the subgoal to be achieved is the surveillance one and the attraction is

$$\mathrm{attr}(u,u_0\dots u_k,h) = \mathrm{pot}_\mathcal{P}(u,u_0\dots u_k,h) + I_{\mathcal{P}\mathsf{sur}}((u_k,u)) \cdot \mathrm{pref}(\pi_1(u_0\dots u_k)), \tag{3.10}$$

where $(u_k,u) \in T_\mathcal{P}$. For a finite run $u_0\dots u_k$, let $C_\mathcal{P}(u_0\dots u_k)$ be the state with the highest attraction (if there are more of them, we choose one randomly). Hence, if the attraction of a state that is not closer to the subgoal is higher than the attraction of ones that are, the collection of rewards is preferred and vice versa. However, note that repeated choices of the states that maximize attr together with Assumption 1 guarantee that the surveillance subgoal, i.e., a visit to $S^\infty_{\mathcal{P}\mathsf{sur}}$, will be eventually achieved. Once it is, the mission subgoal becomes the one to be reached.

For the mission subgoal, the attraction needs to be defined in a different way. The reason is that with an analogous definition as for the surveillance subgoal, we would not be able to ensure eventual visit to $F^\infty_\mathcal{P}$. Intuitively, if $a_\mathsf{sur}$ was repeatedly unintentionally visited, the value of $\mathrm{pref}(\pi_1(u_0\dots u_k))$ might not overgrow the value of $\mathrm{pot}_\mathcal{P}(u,u_0\dots u_k,h)$, the "non-shortening" transitions might always be chosen to follow and a visit to $F^\infty_\mathcal{P}$ might be infinitely postponed.

Thus, we define a projection function $\bar{\pi}_1$ that projects a finite run $u_0\dots u_k$ of $\mathcal{P}$ onto the corresponding finite run of $\mathcal{T}$ while removing $a_\mathsf{sur}$ from some of the states. In particular, on $\bar{\pi}_1(u_0\dots u_k)$, the proposition $a_\mathsf{sur}$ appears at most once between every two successive visits to an accepting state in $F^\infty_\mathcal{P}$.

**Definition 16.** *Let $\overline{\mathcal{T}} = (\overline{S}, s_\mathsf{init}, \overline{T}, AP, \overline{L})$ be a DTS, where $\overline{S} = S \cup \{\bar{s} \mid s \in S\}$, if $(s,s') \in T$ then $(s,s'),(\bar{s},s'),(s,\bar{s'}),(\bar{s},\bar{s'}) \in \overline{T}$, and $\overline{L}(s) = L(s)$ and $\overline{L}(\bar{s}) = L(s) \setminus \{a_\mathsf{sur}\}$ for all $s \in S$. Let $u_0\dots u_k$ be a finite run of $\mathcal{P}$. We define:*

$$\overline{\pi_1}(u_0\dots u_k)(0) = \pi_1(u_0)$$

$$\overline{\pi_1}(u_0\dots u_k)(i) = \begin{cases} \pi_1(u_i) & \text{if } a_\mathsf{sur} \notin L_\mathcal{P}(u_i) \text{ or} \\ & a_\mathsf{sur} \in L_\mathcal{P}(u_i) \text{ and } \exists j < i : u_j \in F^\infty_\mathcal{P} \text{ such that} \\ & \forall j \leq l < i : a_\mathsf{sur} \notin L_\mathcal{P}(u_l) \\ \overline{\pi_1(u_i)} & \text{otherwise.} \end{cases}$$

The definition of the attraction for the mission mode is

$$\mathrm{attr}(u, u_0 \dots u_k, h) = \mathrm{pot}_{\mathcal{P}}(u, u_0 \dots u_k, h) + I_{\mathcal{P}_\phi}((u_k, u)) \cdot \mathrm{pref}(\overline{\pi_1}(u_0 \dots u_k), \tag{3.11}$$

where $(u_k, u) \in T_{\mathcal{P}}$. Similarly as for the surveillance subgoal, the state $C_{\mathcal{P}}(u_0 \dots u_k)$ is the state maximizing the attraction (if there are more of them, we choose one randomly). The construction of the attraction together with Assumption 1 ensure that the mission subgoal is always eventually reached. Once it is, we aim for the surveillance subgoal again. If both of the subgoals are reached simultaneously, the surveillance subgoal is set to be reached.

The outline of the solution to Problem 1 is given in Algorithm 2.

**Algorithm 2** Solution to Problem 1

```
Input: 𝒯, w, φ, rew, v, h, pot, pref
Output: control strategy C
compute a BA 𝒜 for φ and the product 𝒫 = 𝒯 × 𝒜
run Algorithm 1
if F^∞_𝒫 = ∅ or s_{𝒫init} ∉ F^∞_𝒫 then return "Formula cannot be satisfied"
end if
σ := s_{𝒫init}, subgoal := a_sur, k := 0
while true do
    for all u, s.t. (u_k, u) ∈ T_𝒫 do
        compute attr(u, σ, h) using Equation 3.10 if subgoal = a_sur
                                  and Equation 3.11 if subgoal = φ
    end for
    C_𝒫(σ) := arg max_{u∈S_𝒫} attr(u, σ, h)
    C(π1(σ)) := π1(C_𝒫(σ))
    if subgoal = a_sur and C_𝒫(σ) ∈ S^∞_{𝒫sur} then
        subgoal := φ
    end if
    if subgoal = φ and C_𝒫(σ) ∈ F^∞_𝒫 then
        subgoal := a_sur
    end if
    concatenate C_𝒫(σ) to σ; k := k + 1
end while
```

### Properties of the solution

In this section, we prove that under Assumption 1, our algorithm is correct and complete with respect to the satisfaction of the LTL formula (condition (i) of Problem 1). We discuss the sub-optimality of the solution and we introduce an assumption under which the locally planned run is optimal with respect to condition (ii) of Problem 1 among the solutions that do not cause an immediate, unrepairable violation of $\phi$.

**Theorem 1.** Algorithm 2 results in a strategy $C$ that satisfies $\phi$ starting from the initial state $s_{\mathsf{init}}$, if such a strategy exists.

*Proof.* If Algorithm 2 returns "Formula cannot be satisfied" then $F_{\mathcal{P}}^{\infty}$ is empty and according to Lemma 1 and $\phi$ cannot be satisfied in $\mathcal{T}$.

Now assume that Algorithm 2 computes a strategy $C_{\mathcal{P}}$ for the product $\mathcal{P}$. We show by contradiction that $C_{\mathcal{P}}$ generates a run $\rho$ of $\mathcal{P}$ that visits the set $F_{\mathcal{P}}^{\infty}$ infinitely many times. Assume that there is a finite prefix $u_0 \dots u_k$ of $\rho$ such that $u_n \notin F_{\mathcal{P}}^{\infty}$ for all $n \geq k$ and assume that the current subgoal is the surveillance one. Then according to Assumption 1 and the definition of the attraction function in Equation 3.10, the value of $\mathrm{pref}(\pi_1(u_0 \dots u_k \dots u_l)) > \mathrm{pot}_{\mathcal{P}}(u, u_0 \dots u_k \dots u_l, h)$ for all prefixes $u_0 \dots u_k \dots u_l$ of $\rho$ such that $l \geq m$ for some $m \geq k$. This means that the "shortening" transitions will be preferred over the "non-shortening" ones since $t_m$ and thus, $u_j \in S_{\mathcal{P}_{\mathsf{sur}}}^{\infty}$ will be reached eventually. Second, assume that the mission subgoal is the current one. Then according to Assumption 1 and the definition of the attraction function in Equation 3.11, the value of $\mathrm{pref}(\overline{\pi_1}(u_0 \dots u_k \dots u_l)) > \mathrm{pot}_{\mathcal{P}}(u, u_0 \dots u_k \dots u_l, h)$ for all prefixes $u_0 \dots u_k \dots u_l$ of the run $\rho$ such that $l \geq m$ for some $m \geq k$. Analogously to the former case, $u_j \in F_{\mathcal{P}}^{\infty}$ will be reached eventually. Thus the proof is complete. □

In general, the satisfaction of condition (ii) of Problem 1 cannot be guaranteed as repeated visits to the state maximizing Equation 3.5 might prevent the mission to be satisfied. However, we reach some level of optimality as discussed bellow.

In the attraction definition in Equation 3.10, the value of the state potential function $\mathrm{pot}_{\mathcal{P}}(u, u_0 \dots u_k, h)$ is computed in the product instead of the DTS. As a result, it is computed assuming that only sequences of transitions that do not cause an immediate, unrepairable violation of the formula can be followed from $s_k = \pi_1(u_k)$. If the current subgoal of the online planner is the surveillance subgoal, the following optimality statement can be made: A state of $\mathcal{P}$ maximizing the attraction in Equation 3.10 projects onto the state of $\mathcal{T}$ maximizing the cost function in Equation 3.5 taking into consideration only finite runs that do not cause an immediate violation of the formula. In contrast, if the current subgoal of the online planner is the mission one, we cannot claim the similar. First, the indicator function in the attraction in Equation 3.10 does not indicate whether a transition of the product automaton leads closer to $a_{\mathsf{sur}}$, it rather indicates whether it leads closer to both an accepting state and $a_{\mathsf{sur}}$. Second, the preference function in the attraction function in Equation 3.10 is computed for $\overline{\pi_1}(u_0 \dots u_k)$ instead for $\pi_1(u_0 \dots u_k)$. This is necessary for correctness of the algorithm, however, as a result, the value of $\mathrm{pref}(\overline{\pi_1}(u_0 \dots u_k))$ in the attraction in Equation 3.10 might be different than the corresponding value of $\mathrm{pref}(s_0 \dots s_k)$ in the cost function in Equation 3.5.

In case $F_{\mathcal{P}}^{\infty} = \{u' \in S_{\mathcal{P}} \mid u \in S_{\mathcal{P}_{\mathsf{sur}}}^{\infty} \text{ and } (u, u') \in T_{\mathcal{P}}\}$, the mission subgoal is reached always exactly one planning step after the surveillance subgoal is reached. Therefore, we can reach the optimality that was stated in the previous paragraph

for the surveillance subgoal also for the mission subgoal, since all the transitions from $S^{\infty}_{\mathcal{P}_{\mathsf{sur}}}$ are always "shortening" with respect to $F^{\infty}_{\mathcal{P}}$. In particular, this is the case if a Büchi automaton with the property that all the transitions leading to an accepting states are labeled with a set containing $a_{\mathsf{sur}}$, is used in the product automaton construction. For instance, a surveillance fragment of LTL defined in [CTB12] guarantees existence of such a BA. The fragment includes LTL formulas that require to repeatedly visit a surveillance proposition $a_{\mathsf{sur}}$ (called an optimizing proposition in [CTB12]) and to visit a given set of regions in between any two successive visits to states satisfying $a_{\mathsf{sur}}$. In addition, ordering constraints, request-response properties, and safety properties are allowed.

#### Complexity

The size of a BA for an LTL formula $\phi$ is $2^{\mathcal{O}(|\phi|)}$ in the worst case, where $|\phi|$ denotes the length of the formula $\phi$. However, note that the actual size of the BA is in practice often quite small. The size of the product $\mathcal{P}$ is $\mathcal{O}(|S| \cdot 2^{\mathcal{O}(|\phi|)})$. A simple modification of the Floyd-Warshall algorithm is employed to find the minimum weights between each pair of states in $\mathcal{O}(|\mathcal{P}|^3)$. The same complexity is reached for the computation of $F^{\infty}_{\mathcal{P}}, S^{\infty}_{\mathcal{P}_{\mathsf{sur}}}$, $w^{*}_{\mathcal{P}_{\mathsf{sur}}}$ and $w^{*}_{\mathcal{P}_{\phi}}$. The shortening indicators $I_{\mathcal{P}_{\mathsf{sur}}}, I_{\mathcal{P}_{\phi}}$ can be computed in linear time and space with respect to the size of $\mathcal{P}$. The overall complexity of Algorithm 1 is $\mathcal{O}((|S| \cdot 2^{\mathcal{O}(|\phi|)})^3)$. The complexity of the online planning algorithm highly depends on the complexity of the state potential and the preference functions. The set $\mathrm{Run}^{\mathcal{T}}_{\mathrm{fin}}(s, s_k, h)$ can be computed in $\mathcal{O}(d^h)$, where $d$ denotes the maximal out-degree of states of $\mathcal{P}$. If pot and pref functions took constant time to compute, the online planning algorithm would be in $\mathcal{O}(d \cdot d^h)$ per iteration.

### 3.1.4 Case Study

We implemented the framework with several concrete choices of the state potential and the preference function. In this section, we report on simulation results to illustrate employment of our approach.

We consider a data gathering robot in a grid-like partitioned environment modeled as the DTS depicted in Figure 3.1. The robot collects data packages of various, changing sizes (rewards) in the visited regions. The following is known about the reward dynamics: A non-negative natural reward can appear in a state with the current reward equal to 0. The probability of the fresh reward being from $\{0, \ldots, 15\}$ is 50% as well as from $\{16, \ldots, 60\}$, i.e., the smaller-sized data packages are more likely to occur. The reward drops by 1 every time unit as the data outdate. The visibility range $v$ is 6. For example, in Figure 3.1 the visibility region $\mathrm{Vis}(s_{\mathsf{init}})$ for the current state $s_{\mathsf{init}}$ is depicted as the blue-shaded area.

The mission assigned to the robot is to alternately visit the two transmitters (in green, labeled with propositions $a$, and $b$, respectively), while avoiding unsafe locations (in red, labeled with $u$). The surveillance proposition $a_{\mathsf{sur}}$ is true in both

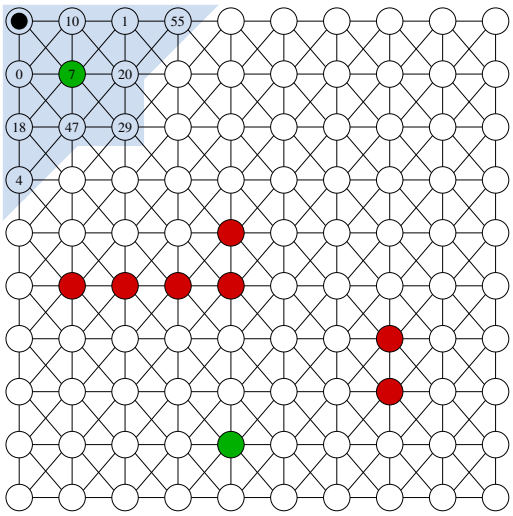



**Figure 3.1:** A weighted DTS representing the robot (illustrated as the black dot) motion model in a partitioned environment. Individual regions of the partitioned environment are depicted as states. Transmitters are in green (labeled with propositions $a$ and $b$, respectively), unsafe locations (labeled with $u$) are in red. The set of transitions contains every pair $(s, s')$ of vertically, horizontally or diagonally neighboring states. Weights of a horizontal and a vertical transition are 2, weight of a diagonal transition is 3.

transmitter regions. The LTL formula for the mission is

$$\phi \;=\; \mathbf{G}\left(a \Rightarrow \mathbf{X}\left(\neg a\,\mathbf{U}\,b\right)\right) \;\wedge\; \mathbf{G}\left(b \Rightarrow \mathbf{X}\left(\neg b\,\mathbf{U}\,a\right)\right) \wedge \mathbf{G}(\neg u) \;\wedge\; \mathbf{G}\,\mathbf{F}\,a_{\mathsf{sur}}.$$

In our simulations, we consider the planning horizon $h = 9$ and several variants of the state potential function and the preference function that are summarized in Table 3.1. The first state potential function $\mathrm{pot}_1$ is the maximal sum of rewards that can be collected on a finite run while taking into account the reward behavior assumptions described above. If the run visits a state more than once or a reward of a state drops below 0, we assume the reward there is 15. The second state potential function $\mathrm{pot}_2$ is defined as the maximal size of a single data package that can be collected on a finite run.

The respective ratio of the value of pref and the maximum value of pot is always non-decreasing with the time elapsed since last transmission and the value of pref overgrows the maximum value of pot when the elapsed time is 50. Intuitively, $\mathrm{pref}_1$ sets zero importance on going towards a transmitter if the last transmission occurred not more then 50 time units ago. On the other hand, $\mathrm{pref}_2$ rises quite slowly at the beginning and very quickly later. In contrast, the function $\mathrm{pref}_3$ grows very fast in the beginning and its growth slows down.

For each of 6 instances we executed 5 runs of 100 iterations of the online

**Table 3.1:** State potential and the preference functions used in the case study.

$$
\begin{array}{rcl}
\mathrm{pot}_1(s, s_0 \dots s_k, h) & = & \max\limits_{\sigma \in \mathrm{Run}^T_{\mathrm{fin}}(s, s_k, h)} \sum_{i=0}^{|\sigma|-1} f_1(\sigma(i), s_0 \dots s_k, \sigma) \\
\mathrm{pot}_2(s, s_0 \dots s_k, h) & = & \max\limits_{\sigma \in \mathrm{Run}^T_{\mathrm{fin}}(s, s_k, h)} \Big( \max\limits_{i=0,\dots,|\sigma|-1} f_2(\sigma(i), s_0 \dots s_k, \sigma) \Big), \\
f_{1,2}(\sigma(i), s_0 \dots s_k, \sigma) & = & \begin{cases} \mathrm{rew}(\sigma(i), s_0 \dots s_k) - w(\sigma^{\to i}) & \text{if this value } > 0, \sigma(i) \neq s_k \\ & \text{and } \forall j < i : \sigma(j) \neq \sigma(i) \\ f_1 = 15, f_2 = 0 & \text{otherwise} \end{cases} \\
\hline
\mathrm{pref}_1(s_0 \dots s_k) & = & \begin{cases} 0 & \text{if } w(s_{i_{\mathsf{sur}}} \dots s_k) \leq 50 \\ \mathrm{maxpot}(s_0 \dots s_k, h) + 1 & \text{otherwise} \end{cases} \\
\mathrm{pref}_2(s_0 \dots s_k) & = & \frac{1}{50^3} \cdot w(s_{i_{\mathsf{sur}}} \dots s_k)^3 \cdot \mathrm{maxpot}(s_0 \dots s_k, h) \\
\mathrm{pref}_3(s_0 \dots s_k) & = & \frac{1}{\sqrt[3]{50}} \cdot \sqrt[3]{w(s_{i_{\mathsf{sur}}} \dots s_k)} \cdot \mathrm{maxpot}(s_0 \dots s_k, h) \\
i_{\mathsf{sur}} & = & \max\{i \mid 0 \leq i \leq k, s_{i_{\mathsf{sur}}} \in S_{\mathsf{sur}}\} \\
\mathrm{maxpot}(s_0 \dots s_k, h) & = & \max\limits_{(s_k, s) \in T} \mathrm{pot}(s, s_0 \dots s_k, h)
\end{array}
$$

planner. The sizes of the data collected in time are depicted in Fig. 3.2. Table 3.2 shows the mean of average reward per transition and the time between consecutive surveys, respectively. As expected, the faster the preference function grows with time since the last survey, the smaller the reward per transition and the shorter the time between consecutive transmissions are. For $\mathrm{pref}_1$ and $\mathrm{pref}_2$, the difference in the reward per transition is not high, since in both cases the collection of rewards is preferred in the beginning, whereas $\mathrm{pref}_3$ is very steep and therefore drives the robot towards transmitter quickly. Function $\mathrm{pot}_1$ computing the maximal sum of rewards that can be collected gives, as expected, higher average and lower variance for both objectives comparing to $\mathrm{pot}_2$ that aims to collect big packages.

The experiments were run on Mac OS X 10.7.3 with 2.7 GHz Intel Core i5 and 4 GB DDR3 memory. The BA had 8 states (3 accepting) and it satisfied the condition for optimality from Section 3.1.3. The product had 800 states. Algorithm 1 took 6 seconds and one iteration of the online receding horizon control algorithm 1-2 milliseconds.

### 3.1.5 Conclusion

We proposed a general framework for control synthesis in an environment with dynamically changing locally sensed rewards. While a high-level surveillance mission is guaranteed to be accomplished, the user-defined priorities on trade-off between the surveillance frequency and the reward collection are taken into account. The system is modeled as a weighted deterministic transition system and although the weights are in this chapter interpreted as time durations of the transitions, they

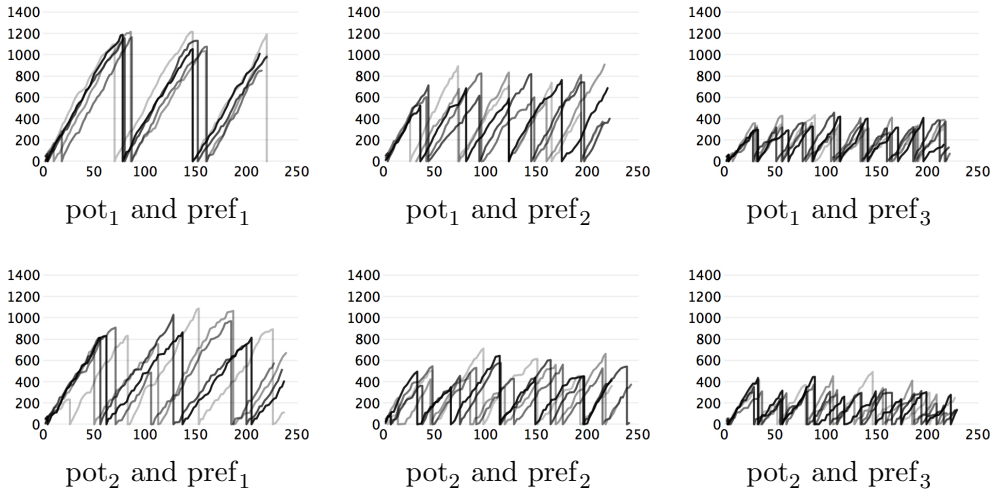



**Figure 3.2:** Total size of data collected since the last transmission with respect to time depicted for each executed run.

**Table 3.2:** Statistical results for the reward per transition (rew/$T$) and the time between consecutive surveys ($t$) for different choices of pot/pref functions (in the header). AVG is the mean of average computed on each run and VAR the mean of variance computed on each run. $\nu$ shows the percentage variance of the average among the runs.

| | | 1/1 | 1/2 | 1/3 | 2/1 | 2/2 | 2/3 |
|---|---|---|---|---|---|---|---|
| rew/$T$ | AVG | 33.8 | 33.7 | 28.2 | 30.8 | 29.2 | 25.9 |
| | $\nu$ | 2.4% | 4.4% | 6.0% | 3.9% | 5.8% | 8.3% |
| | VAR | 13.8 | 14.6 | 15.9 | 19.2 | 18.3 | 18.7 |
| $t$ | AVG | 73.4 | 46.0 | 26.7 | 66.2 | 41.9 | 26.4 |
| | $\nu$ | 2.4% | 8.0% | 2.1% | 8.4% | 7.7% | 3.0% |
| | VAR | 2.8 | 6.0 | 2.7 | 11.2 | 6.5 | 3.3 |

can be, in general interpreted, as any quantitative aspect, such as length or cost.

## 3.2 Deterministic Systems with Penalties

### 3.2.1 Motivation

In this section, we consider an analogous problem to the one discussed in Section 3.1. We assume that the system is modeled as a weighted deterministic transition system with time-varying, locally sensed values associated with every state. However, here we assume that the values are interpreted as penalties that

can be used to encode various dynamic features of the system. For example, consider a mobile robot involved in a surveillance mission in a dangerous area and under tight fuel and time constraints. The correctness requirement is a temporal logic specification, e.g., "Alternately keep visiting $A$ and $B$ and always avoid $C$", while the resource constraints translate to minimizing a cost function over the feasible trajectories. The concept of such dynamic values is well adopted, e.g., in reinforcement learning [Sze10] with applications in robotics, games or economics. Here, we consider probabilistic penalties defined as Markov chains that are used to model environmental phenomena with known statistics such as the traffic load or the value of a stock.

Unlike in Section 3.1, our goal is to optimize penalty collection over infinite time horizon. To be specific, given an LTL mission involving persistent surveillance, our goal is to minimize the expected average cumulative penalty incurred between consecutive visits of states under surveillance, while at the same time satisfying the additional temporal constraint. We design two algorithms that bring together concepts from automata-based model checking, game theory and receding horizon control. The first computes an offline, optimal control strategy that uses only the a priori known transition probabilities of the penalties' Markov chains, but does not exploit their actual values sensed in real time. Up to special cases, the optimal strategy requires infinite memory. We show that using simple feedback, the strategy can be implemented efficiently. The second proposed algorithm shows that by taking advantage of the local sensing, we can design an online, receding horizon control strategy that, while still satisfying the temporal specification, provides lower value of the optimization function. The online strategy is a heuristic that locally improves the offline strategy based on local sensing and simulation over a user-defined planning horizon. While we can prove optimality of the offline strategy among the strategies that disregard local sensing, it is intractable to construct or use an optimal strategy among those that utilize it. We also suggest a method to construct a whole class of online strategies with good expected behaviors.

This paper is related to [STBR11, WTM12a], which also focus on optimal control for weighted deterministic transition systems with temporal constraints. In [STBR11], the authors develop a control strategy that minimizes the maximum weight between consecutive visits to a given set of states, subject to constraints expressed as LTL formulas. In addition to weights, transitions in [WTM12a] are assigned with costs and a strategy is constructed that minimizes the weighted average cost, while satisfying an LTL formula. The proposed solution is related to our offline approach, where we disregard the local sensing of penalties and consider only their expected, static value. We address this correlation and our contribution over [WTM12a] in more detail in the following sections.

The results presented in this section are based on our results in [SvB13b, SvB15]. The rest of the section organized as follows. In Section 3.2.2, we describe Markov chains as models of penalties. The problem is formally stated in Sec. 3.2.3

and Sec. 3.2.4 contains our main results. Finally, in Sec. 3.2.5, we present the two case studies and discuss simulation results.

### 3.2.2 Markov Chains

**Definition 17.** *A Markov chain (MC) is a tuple* $\mathcal{M} = (G, P, p_{\mathsf{init}})$*, where* $G$ *is a nonempty finite set of states,* $P : G \times G \to [0,1]$ *is a transition probability function such that* $\sum_{g' \in G} P(g, g') = 1$ *for all* $g \in G$ *and* $p_{\mathsf{init}} : G \to [0,1]$ *is an initial distribution, i.e.,* $\sum_{g \in G} p_{\mathsf{init}}(g) = 1$*.*

A run of a Markov chain is an infinite sequence $g_0 g_1 g_2 \ldots \in G^\omega$ such that for every $i \geq 0$ it holds $P(g_i, g_{i+1}) > 0$. A finite run of $\mathcal{M}$ is a finite prefix of a run of $\mathcal{M}$. A Markov chain is called strongly connected if for every pair $g, g' \in G$ of states there exists a finite run from $g$ to $g'$. We call a Markov chain nontrivial if $|G| > 1$ implies that there exist $g, g' \in G$ such that $P(g, g') \in (0, 1)$.

In this work, we only consider strongly connected nontrivial Markov chains due to reasons explained later in Remark 3 and we interpret them as discrete stochastic processes [Nor98], i.e., time series of values that involve probabilistic indeterminacy. We assume that the set of states $G = \{g_0, g_1, \ldots g_n\}$ is an ordered finite set of non-negative real numbers and we refer to $G$ as the set of values. We use $\mathcal{M}(t)$ to denote the value (or state) of Markov chain $\mathcal{M}$ at time $t \in \mathbb{N}_0$. The function $P$ can be represented as a square matrix $A_{\mathcal{M}} = \{A_{ij}\}$, where $A_{ij} = P(g_i, g_j)$.

**Definition 18.** *The invariant distribution of a (strongly connected nontrivial) Markov chain* $\mathcal{M} = (G, P, p_{\mathsf{init}})$*,* $G = \{g_0, \ldots, g_n\}$*, is a vector* $\nu_{\mathcal{M}}$ *of size* $n+1$ *such that for every* $0 \leq i \leq n$ *it holds* $0 \leq \nu_{\mathcal{M}}(i) \leq 1$*,* $\sum_{i=0}^{n} \nu_{\mathcal{M}}(i) = 1$ *and* $\nu_{\mathcal{M}} \cdot A_{\mathcal{M}} = \nu_{\mathcal{M}}$*.*

Intuitively, $\nu_{\mathcal{M}}(i)$ is the probability of the Markov chain $\mathcal{M}$ being in state $g_i$ at any point of an execution. Note that the invariant distribution can be effectively computed using the above definition. The expected value of $\mathcal{M}$ is then

$$\mathcal{M}_E = \sum_{i=0}^{n} \nu_{\mathcal{M}}(i) \cdot g_i.$$

Assume that $\mathcal{M}(t) = g_i$ for some $t \geq 0, 0 \leq i \leq n$, and let $k \geq 0$. The simulated expected value

$$\mathcal{M}_{\mathrm{sim}}(t, g_i, k) = \sum_{j=0}^{n} (A_{\mathcal{M}}^k)_{ij} \cdot g_j$$

is the expected value of $\mathcal{M}$ at time $t + k$ assuming that its value is $g_i$ at time $t$.

### 3.2.3 Problem Formulation

Consider a DTS $\mathcal{T} = (S, T, AP, L)$ with initial state $s_{\mathsf{init}} \in S$ and a weight function $w \colon T \to \mathbb{R}^+$ as in Equation 3.1. The weight $w((s, s'))$ represents the amount of

time that the transition $(s, s') \in T$ takes and the system starts at time 0. We use $t_n$ to denote the point in time after the $n$-th transition of a run, i.e., initially $t_0 = 0$ and after a finite run $\sigma \in \mathrm{Run}^{\mathcal{T}}_{\mathrm{fin}}(s_{\mathsf{init}})$ of length $n+1$ the time is $t_n = w(\sigma)$.

We assume there is a dynamic *penalty* associated with every state of the transition system. The penalty in a state $s \in S$ is defined as a (strongly connected nontrivial) Markov chain $\mathcal{M}_s$. With a slight abuse of notations, we use

$$\mathrm{pen}(s, t) = \mathcal{M}_s(t)$$

to denote the value of the penalty in a state $s \in S$ at time $t \in \mathbb{N}_0$ and

$$\mathrm{pen}_E(s) = \mathcal{M}_{sE}$$

is the expected value of the penalty in state $s$. Assuming that the penalty in state $s$ is $x$ at time $t \in \mathbb{N}_0$,

$$\mathrm{pen}_{\mathrm{sim}}(s, t, x, k) = \mathcal{M}_{s\,\mathrm{sim}}(t, x, k)$$

is the simulated expected value of the penalty in state $s$ at time $t + k$. We use $\mathrm{pen}_{\max}$ to denote the maximum possible value of a penalty over all states. Upon the visit of a state, the corresponding penalty is incurred. The visit of the state does not affect the penalty's value or dynamics.

In every execution of the transition system $\mathcal{T}$, the probabilistic choices during the evolution of all penalties are resolved in some particular way that we call *penalty profile*. A penalty profile $\Delta$ determines, for a particular execution of the TS $\mathcal{T}$, the value of penalty in every state at every time moment. The penalty profile that is being followed in an execution is not known to us. Nevertheless, we can compute the probability $\Pr(\Delta, k)$ that a penalty profile $\Delta$ will be followed in the first $k$ time units of an execution based on the Markov chains $\mathcal{M}_s, s \in S$.

Motivated by robotic applications, where various sensors typically provide reasonable measurements only within certain range, we assume that the penalties are sensed only locally in close proximity from the current state. To be specific, we assume a *visibility range* $v \in \mathbb{N}$ is given. If the system is in a state $s \in S$ at time $t$, the penalty $\mathrm{pen}(s', t)$ of a state $s' \in S$ is observable if and only if $s' \in \mathrm{Vis}(s) = \{s' \in S \mid w^*(s, s') \leq v\}$. The set $\mathrm{Vis}(s)$ is also called the set of states visible from $s$. We consider the penalties to be an integral part of the DTS and thus, a strategy for the DTS might consider in its decision procedure not only the sequence of states that have been visited in the past but also the penalties incurred and observed in the meantime. Therefore, the run induced by a strategy $C$ for $\mathcal{T}$ may differ under different penalty profiles. We use $\rho_{C,\Delta}(s)$ to denote the run induced by a strategy $C$ under a penalty profile $\Delta$ starting from a state $s \in S$.

We assume that the specification is given as an LTL formula with persistent surveillance as defined in Section 2.2.4. Recall that a formula with persistent surveillance is

$$\phi = \varphi \wedge \mathbf{GF}\, a_{\mathsf{sur}},$$

where $\varphi$ is an arbitrary LTL formula over $AP$ and $a_{\mathsf{sur}} \in AP$ is a surveillance proposition. The formula states the system must satisfy a temporal constraint $\varphi$ and at the same time, it must infinitely many times survey states $S_{\mathsf{sur}} = \{s \in S \mid a_{\mathsf{sur}} \in L(s)\}$ labeled with the proposition $a_{\mathsf{sur}}$. We extend the notation from Section 2.2.4 and use $\sharp(\sigma)$ to denote the number of complete surveillance cycles in a finite run $\sigma$ such that $\mathrm{last}(\sigma) \in S_{\mathsf{sur}}$, otherwise $\sharp(\sigma)$ denotes the number of complete surveillance cycles in $\sigma$ plus one.

The long-term optimization objective is defined as follows. Let $V_{\mathcal{T},C} \colon S \to \mathbb{R}_0^+$ be a function such that $V_{\mathcal{T},C}(s)$ is the *expected average cumulative penalty per surveillance cycle (APPC)* incurred under a strategy $C$ for $\mathcal{T}$ starting from a state $s \in S$:

$$V_{\mathcal{T},C}(s) = \limsup_{k\to\infty} \sum_{\Delta} \Pr(\Delta, k) \cdot \frac{\sum_{i=0}^{k} \mathrm{pen}(\rho_{C,\Delta}(i), w(\rho_{C,\Delta}^{\to i}))}{\sharp(\rho_{C,\Delta}^{\to k})} \tag{3.12}$$

where $\rho_{C,\Delta} \in \mathrm{Run}^{\mathcal{T}}(s)$ is the run induced by $C$ starting from $s$ under a penalty profile $\Delta$. In the special case, when $a_{\mathsf{sur}} \in L(s)$ for every $s \in S$, the long-term optimization objective minimizes the expected average penalty incurred per stage that is a standard optimization objective in control theory [Ber12] as well as in game theory [AG11].

**Problem 2.** ***Given***

- *an initialized DTS $\mathcal{T} = (S, T, AP, L)$ with initial state $s_{\mathsf{init}} \in S$,*
- *a weight function $w$,*
- *an LTL formula $\phi$ over $AP$ with persistent surveillance,*
- *penalties defined by (strongly connected nontrivial) Markov chains $\mathcal{M}_s, s \in S$,*
- *a visibility range $v \in \mathbb{R}_{>0}, v \geq \max_{(s,s')\in T} w(s, s')$,*

***find*** *a control strategy $C \colon \mathrm{Run}_{\mathrm{fin}}^{\mathcal{T}} \to S$ such that*

*(i) $C$ satisfies $\phi$ starting from $s_{\mathsf{init}}$ and*

*(ii) among all strategies satisfying (i), $C$ minimizes the APPC value $V_{\mathcal{T},C}(s_{init})$ defined in Equation 3.12.*

Note that a strategy that solves Problem 2 is dependent on the penalty profiles and it defines an optimal control sequence for each penalty profile separately. However, due to the probabilistic nature of penalties, we are not able to predict their values in the future precisely, i.e., we are not able to determine the profile that is being followed during an execution. Hence, even if we were able to construct the solution to Problem 2, we do not have the means to use it. Therefore, we consider the following relaxed version of Problem 2. Consider strategies that are independent on the penalty profiles, i.e., $\rho_{C,\Delta}(s) = \rho_{C,\Delta'}(s)$ for any two penalty profiles $\Delta, \Delta'$ and a state $s$. Note that such strategies may still consider the Markov chains defining the penalties in states of the TS and their expected values.

For this type of strategies the value $V_{\mathcal{T},C}(s)$ for a state $s \in S$ from Equation 3.12 can be computed easily as follows:

$$V_{\mathcal{T},C}(s) = \limsup_{k\to\infty} \frac{\sum_{i=0}^{k} \mathrm{pen}_E(\rho_C(i))}{\sharp(\rho_C^{\to k})}. \tag{3.13}$$

**Problem 3.** ***Given***

- *an initialized DTS* $\mathcal{T} = (S, T, AP, L)$ *with initial state* $s_{\mathsf{init}} \in S$,
- *a weight function* $w$,
- *an LTL formula* $\phi$ *over* $AP$ *with persistent surveillance,*
- *penalties defined by (strongly connected nontrivial) Markov chains* $\mathcal{M}_s, s \in S$,
- *a visibility range* $v \in \mathbb{R}_{>0}, v \geq \max_{(s,s')\in T} w(s, s')$,

***find*** *a control strategy* $C \colon \mathrm{Run}_{\mathrm{fin}}^{\mathcal{T}} \to S$ *such that*

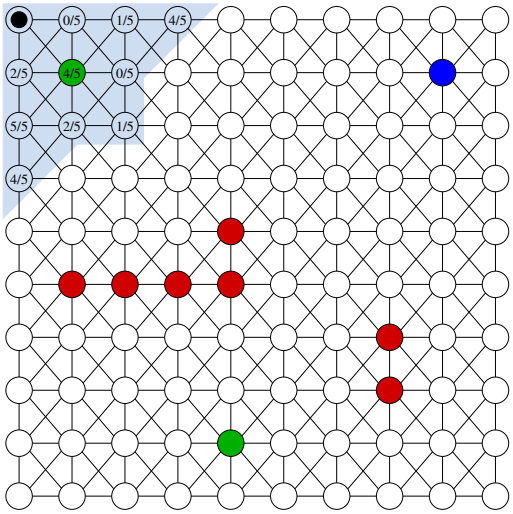
*(i)* $C$ *is independent on penalty profiles,*

*(ii)* $C$ *satisfies* $\phi$ *starting from* $s_{\mathsf{init}}$ *and*

*(iii) among all strategies satisfying (i) and (ii),* $C$ *minimizes the APPC value* $V_{\mathcal{T},C}(s_{init})$ *defined in Equation 3.13.*

In Section 3.2.4, we propose an algorithm to design a strategy that solves Problem 3. Since the resulting strategy is independent on penalty profiles, it does not take advantage of the local sensing of penalties. It is computed in an offline manner and we refer to it as the offline strategy or offline control. While the offline strategy minimizes the APPC value among strategies satisfying the formula that are independent on penalty profiles, there may exist strategies that are dependent on penalty profiles and while satisfying the formula, provide lower APPC value than the offline control. We construct such a strategy in Section 3.2.4 as well. Since the strategy is dependent on penalty profiles, it considers the penalties observed in real-time. This online control is constructed using the principles from receding horizon control, where we locally improve the offline control according to the penalties observed from the current state of the DTS and their simulation over the next $h$ time units, where $h \in \mathbb{N}_0$ is a user-defined *planning horizon*. Note that the variable planning horizon will be used differently here and in Section 3.1. The online strategy is a heuristic, and we also suggest a method to construct a whole class of strategies with similar properties. In Section 3.2.5, we evaluate all designed control strategies on illustrative case studies. All strategies synthesized in this work are infinite-memory in general, but can be implemented efficiently using simple technical improvements.

**Example 3.** Consider a robot whose motion in a grid-like partitioned environment is modeled by the transition system depicted in Figure 3.3a. The robot transports packages between two delivery locations, marked green in Fig. 3.3a. The blue state marks the robot's base location. There is a transition between every two vertically, horizontally, and diagonally neighboring states. The weight




(a)

| | 0 | 1/5 | 2/5 | 3/5 | 4/5 | 1 |
|---|---|---|---|---|---|---|
| 0 | 0 | 1.0 | 0 | 0 | 0 | 0 |
| 1/5 | 0 | 0 | 1.0 | 0 | 0 | 0 |
| 2/5 | 0 | 0 | 0 | 1.0 | 0 | 0 |
| 3/5 | 0 | 0 | 0 | 0 | 1.0 | 0 |
| 4/5 | 0 | 0 | 0 | 0 | 0 | 1.0 |
| 1 | $1-p(s)$ | 0 | 0 | 0 | 0 | $p(s)$ |

(b)

(c)

**Figure 3.3:** (a) Transition system modeling the robot (black dot) motion in a partitioned environment. There are two delivery locations shown in green, a base shown in blue, and unsafe locations shown in red. (b) Transition matrix $A_{\mathcal{M}_s}$ of the Markov chain $\mathcal{M}_s$ that defines the penalty in a state $s$. (c) Graphical representation of the function $p\colon S \to (0,1)$. The values range over the set $\{0.1,\ldots,0.9\}$. Darker shades indicate higher values.

of a horizontal and vertical transition is 2, for a diagonal transition it is 3. The Markov chain $\mathcal{M}_s$ defining the penalty in a state $s$ has the set of values $G = \{0, \frac{1}{5}, \frac{2}{5}, \frac{3}{5}, \frac{4}{5}, 1\}$, the initial distribution is the uniform distribution over $G$ and the transition matrix is of the form shown in Figure 3.3b. Intuitively, every penalty increases every time unit by $\frac{1}{5}$ and always when the penalty is 1, in the next time unit the penalty remains 1 with nonzero probability $p(s)$ or it drops to 0 with nonzero probability $1-p(s)$, where $p\colon S \to (0,1)$ is a function over states of the system, defined in Figure 3.3c. The visibility range $v$ is 6. For example, in Figure 3.3a the set $\mathrm{Vis}(s)$ of states visible from the state $s$, with corresponding penalties, is depicted as the blue-shaded area.

The mission for the robot is to transport packages between the two delivery locations (labeled with propositions $a$ and $b$, respectively) and infinitely many times return to the base (labeled with $c$), while avoiding unsafe locations (labeled with $d$). At the same time, we wish to minimize the expected average cumulative penalty incurred per transport. To model this requirement, we add the property

$a_{\mathsf{sur}}$ to the label set of both delivery locations. The corresponding LTL formula is

$$\mathbf{G}\,(a \Rightarrow \mathbf{X}\,(\neg a\,\mathbf{U}\,b)) \wedge \mathbf{G}\,(b \Rightarrow \mathbf{X}\,(\neg b\,\mathbf{U}\,a)) \wedge \mathbf{GF}\,c \wedge \mathbf{G}(\neg d) \wedge \mathbf{GF}\,a_{\mathsf{sur}}. \quad (3.14)$$

### 3.2.4 Problem Solution

In this section, we present algorithms to construct the offline and online control strategies. Both algorithms work with the product

$$\mathcal{P} = \mathcal{T} \times \mathcal{A} = (S_{\mathcal{P}}, T_{\mathcal{P}}, s_{\mathcal{P}\mathsf{init}}, AP, L_{\mathcal{P}}, F_{\mathcal{P}})$$

of the initialized DTS $\mathcal{T}$ and a Büchi automaton $\mathcal{A}$ for the LTL formula $\phi$, see Definition 14. We map the penalties from $\mathcal{T}$ to $\mathcal{P}$ by defining $\mathcal{M}_{(s,q)} = \mathcal{M}_s$ for every state $(s,q) \in S_{\mathcal{P}}$ and $\mathrm{pen}((s,q),t) = \mathrm{pen}(s,t)$ for every $t \in \mathbb{N}_0$. We also adopt the visibility range $v$ and the definition of the set $\mathrm{Vis}((s,q))$. When convenient, we use singletons such as $u, u', u_i$ to denote states of the product. We distinguish between control strategies for $\mathcal{P}$ that are dependent on penalty profiles and those that are not, i.e., between strategies for which the APPC value is computed directly using the definition in Equation 3.12 and those for which the value can be computed easily using Equation 3.13. The control strategies for $\mathcal{T}$ are computed as projections of control strategies for $\mathcal{P}$ with suitable properties.

We start with a formal definition of a probability measure that allows us to argue about possible sequences of penalties incurred in a run of the product. We follow with the description of the offline and online control strategies. We discuss their properties, complexity and usability.

#### Probability measure

Let $C_{\mathcal{P}}$ be a strategy for $\mathcal{P}$ that is independent on penalty profiles and let $(s,q) \in S_{\mathcal{P}}$. Let $\sigma \in \mathrm{Run}^{\mathcal{P}}_{\mathrm{fin}}((s,q))$ be a finite run induced by $C_{\mathcal{P}}$ starting from $(s,q)$ and let $\tau \in [0, \mathrm{pen}_{\max}]^+$ be a sequence of length $|\sigma|$ such that there exists a penalty profile for $\mathcal{P}$ for which the penalty

$$\mathrm{pen}(\sigma(i), w_{\mathcal{P}}(\sigma^{\to i})) = \tau(i)$$

for every $0 \leq i \leq |\sigma|$. We call $(\sigma, \tau)$ a finite pair. Analogously, an infinite pair $(\rho, \kappa)$ consists of the run $\rho \in \mathrm{Run}^{\mathcal{P}}((s,q))$ induced by the strategy $C_{\mathcal{P}}$ and an infinite sequence $\kappa \in [0, \mathrm{pen}_{\max}]^{\omega}$ such that there exists a penalty profile for $\mathcal{P}$ for which the penalty

$$\mathrm{pen}(\sigma(i), w_{\mathcal{P}}(\sigma^{\to i})) = \kappa(i)$$

for every $i \geq 0$. A cylinder set $\mathrm{Cyl}((\sigma,\tau))$ of a finite pair $(\sigma,\tau)$ is the set of all infinite pairs $(\rho,\kappa)$ for which $\tau$ is a prefix of $\kappa$.

Consider the $\sigma$-algebra generated by the set of cylinder sets of all finite pairs $(\sigma,\tau)$, where $\sigma \in \mathrm{Run}^{\mathcal{P}}_{\mathrm{fin}}((s,q))$. From classical concepts in probability theory [ADD00], there exists a unique probability measure $\Pr^{\mathcal{P},C_{\mathcal{P}}}_{(s,q)}$ on the $\sigma$-algebra

such that for a finite pair $(\sigma, \tau)$

$$\Pr_{(s,q)}^{\mathcal{P},C_{\mathcal{P}}}(\mathrm{Cyl}((\sigma,\tau)))$$

is the probability that the penalties incurred in the first $|\sigma|+1$ stages when applying the strategy $C_{\mathcal{P}}$ in $\mathcal{P}$ from the state $(s,q)$ are given by the sequence $\tau$, i.e.,

$$\mathrm{pen}(\sigma(i), w_{\mathcal{P}}(\sigma^{\to i})) = \tau(i)$$

for every $0 \leq i \leq |\sigma|$. This probability can be computed based on the Markov chains $\mathcal{M}_{(s,q)}, (s,q) \in S_{\mathcal{P}}$ and it is equal to the sum of probabilities $\Pr(\Delta, w_{\mathcal{P}}(\sigma))$ over all penalty profiles $\Delta$ under which the above equation is satisfied. For a set $X$ of infinite pairs, an element of the above $\sigma$-algebra, the probability $\Pr_{(s,q)}^{\mathcal{P},C_{\mathcal{P}}}(X)$ is the probability that under $C_{\mathcal{P}}$ starting from $(s,q)$ the infinite sequence of penalties incurred in the visited states is $\kappa$ for some $(\rho,\kappa) \in X$.

### Offline control

In this section, we construct a strategy $C$ for $\mathcal{T}$ that solves Problem 3 as a projection of a strategy $C_{\mathcal{P}}^{\mathsf{off}}$ for $\mathcal{P}$. All strategies in this subsection including $C_{\mathcal{P}}^{\mathsf{off}}$ are independent on penalty profiles, i.e., their APPC value can be computed using Equation 3.13.

In order for $C_{\mathcal{P}}^{\mathsf{off}}$ to project on a solution to Problem 3, the run induced by $C_{\mathcal{P}}^{\mathsf{off}}$ must visit the set $F_{\mathcal{P}}$ infinitely many times and at the same time, the APPC value $V_{\mathcal{P},C_{\mathcal{P}}^{\mathsf{off}}}(s_{\mathcal{P}\mathsf{init}})$ must be minimal among all strategies for $\mathcal{P}$ that are independent on penalty profiles and visit the set $F_{\mathcal{P}}$ infinitely many times. Hence, the strategy $C_{\mathcal{P}}^{\mathsf{off}}$ must lead from $s_{\mathcal{P}\mathsf{init}}$ to an ASCC. If the set $\mathrm{ASCC}(\mathcal{P})$ is empty, the formula $\phi$ cannot be satisfied in $\mathcal{T}$ and our algorithm terminates. Otherwise, for $\mathcal{U} \in \mathrm{ASCC}(\mathcal{P})$, we denote $V_{\mathcal{U}}^*((s,q))$ the minimum expected average cumulative penalty per surveillance cycle that can be achieved in $\mathcal{U}$ by a strategy independent on penalty profiles starting from $(s,q) \in S_{\mathcal{U}}$. Since $\mathcal{U}$ is strongly connected, this value is the same for all states in $S_{\mathcal{U}}$ and is denoted by $V_{\mathcal{U}}^*$. It is associated with a cycle $\mathsf{cyc}_{\mathcal{U}}^V = c_0 \dots c_m$ of $\mathcal{U}$ witnessing the value. We describe the algorithm to compute $V_{\mathcal{U}}^*$ and $\mathsf{cyc}_{\mathcal{U}}^V$ for $\mathcal{U}$ in the proof of Theorem 2. In the remainder of this section, $\mathcal{U} = (S_{\mathcal{U}}, T_{\mathcal{U}}, AP, L_{\mathcal{P}})$ is the ASCC of the product that minimizes $V_{\mathcal{U}}^*$ and $\mathsf{cyc}_{\mathcal{U}}^V = c_0 \dots c_m$ is the corresponding cycle.

Before constructing the strategy $C_{\mathcal{P}}^{\mathsf{off}}$, let us intuitively describe the main idea behind this strategy. Using the automata-based approach to model checking, one can construct a strategy $C_{\mathcal{P}}^{\mathsf{off},\phi}$ for $\mathcal{P}$ that visits at least one of the accepting states infinitely many times. On the other hand, using graph theory, we can design a strategy $C_{\mathcal{P}}^{\mathsf{off},V}$ that achieves the minimum APPC value among all strategies for $\mathcal{P}$ that are independent on penalty profiles and do not cause an immediate, unrepairable violation of $\phi$, i.e., $\phi$ is satisfiable from every state of the run induced by $C_{\mathcal{P}}^{\mathsf{off},V}$. However, we would like to have a strategy $C_{\mathcal{P}}^{\mathsf{off}}$ satisfying both properties

at the same time. To achieve that, we draw inspiration from a recent game theoretic technique proposed in [CD12]. Intuitively, we combine two strategies $C_{\mathcal{P}}^{\mathsf{off},\phi}$ and $C_{\mathcal{P}}^{\mathsf{off},V}$ to create a new strategy $C_{\mathcal{P}}^{\mathsf{off}}$. The strategy $C_{\mathcal{P}}^{\mathsf{off}}$ is played in rounds, where each round consists of two phases. In the first phase, we play the strategy $C_{\mathcal{P}}^{\mathsf{off},\phi}$ until an accepting state is reached. We say that the aim is to achieve the mission subgoal. The second phase applies the strategy $C_{\mathcal{P}}^{\mathsf{off},V}$. The aim is to maintain the expected average cumulative penalty per surveillance cycle in the current round, and we refer to it as the average subgoal. The number of steps for which we apply $C_{\mathcal{P}}^{\mathsf{off},V}$ is computed individually every time we enter the second phase of a round. The resulting offline control strategy is not a finite-memory strategy in general. Intuitively, we need to perform more and more steps in every round of the strategy. We discuss this matter in detail later in this section and suggest technical improvements that reduce the number of steps in every round and the usage of memory. The improvements result in a strategy that is equivalent to the offline strategy in the sense that it guarantees the satisfaction of the given LTL formula and has the same APPC value as the offline control strategy, but is no longer independent on penalty profiles.

Now, we design the strategies $C_{\mathcal{P}}^{\mathsf{off},\phi}$ and $C_{\mathcal{P}}^{\mathsf{off},V}$ that are then combined to create the strategy $C_{\mathcal{P}}^{\mathsf{off}}$. The strategy $C_{\mathcal{P}}^{\mathsf{off},\phi}$ is a memoryless strategy that from every $u \in S_{\mathcal{P}} \backslash F_{\mathcal{U}}$ that can reach the set $F_{\mathcal{U}}$, follows one of the finite runs with the minimum weight from $u$ to $F_{\mathcal{U}}$. Formally, for every $u \in S_{\mathcal{P}} \backslash F_{\mathcal{U}}$ with $w^*_{\mathcal{P}}(u, F_{\mathcal{U}}) < \infty$, we define

$$C_{\mathcal{P}}^{\mathsf{off},\phi}(u) = u' \text{ such that } w_{\mathcal{P}}(u,u') = w^*_{\mathcal{P}}(u, F_{\mathcal{U}}) - w^*_{\mathcal{P}}(u', F_{\mathcal{U}}).$$

The strategy $C_{\mathcal{P}}^{\mathsf{off},V}$ is a memoryless strategy given by the cycle $\mathsf{cyc}^V_{\mathcal{U}} = c_0 \dots c_m$. Similarly as for the above strategy $C_{\mathcal{P}}^{\mathsf{off},\phi}$, for a state $u \in S_{\mathcal{P}} \backslash \mathsf{cyc}^V_{\mathcal{U}}$ such that $w^*_{\mathcal{P}}(u, \mathsf{cyc}^V_{\mathcal{U}}) < \infty$, the strategy $C_{\mathcal{P}}^{\mathsf{off},V}$ follows one of the finite runs with the minimum weight to $\mathsf{cyc}^V_{\mathcal{U}}$, i.e.,

$$C_{\mathcal{P}}^{\mathsf{off},V}(u) = u' \text{ such that } w_{\mathcal{P}}(u,u') = w^*_{\mathcal{P}}(u, \mathsf{cyc}^V_{\mathcal{U}}) - w^*_{\mathcal{P}}(u', \mathsf{cyc}^V_{\mathcal{U}})$$

and for a state $c_i \in \mathsf{cyc}^V_{\mathcal{U}}$, it holds

$$C_{\mathcal{P}}^{\mathsf{off},V}(c_i) = c_{i+1 \bmod (m+1)}.$$

**Proposition 1.** For the strategy $C_{\mathcal{P}}^{\mathsf{off},V}$ and every state $u \in S_{\mathcal{U}}$ it holds that for every $\epsilon > 0$, there exists $j(\epsilon) \in \mathbb{N}$ such that if $C_{\mathcal{P}}^{\mathsf{off},V}$ is followed from the state $u$ until at least $j(\epsilon)$ surveillance cycles are completed, then the average cumulative penalty per surveillance cycle incurred in the performed finite run is at most $V^*_{\mathcal{U}} + \epsilon$ with probability at least $1 - \epsilon$. Formally:

$$\lim_{k\to\infty} \Pr_u^{\mathcal{U}, C_{\mathcal{P}}^{\mathsf{off},V}} \Big( \frac{\sum_{i=0}^{k} \mathrm{pen}(\rho(i), w_{\mathcal{P}}(\rho^{\to i}))}{\sharp(\rho^{\to k})} \leq V^*_{\mathcal{U}} \Big) = 1, \tag{3.15}$$

where $\rho$ is the run induced by $C_{\mathcal{P}}^{\mathsf{off},V}$ starting from $u$.

*Proof.* It holds that the product $\mathcal{P}$ with penalties defined as MCs can be translated into a Markov decision process with static penalties. Together with the fact that the cycle $\mathsf{cyc}^*_{\mathcal{U}}$ provides the minimum APPC value in the ASCC $\mathcal{U}$, it implies that Equation 3.15 is equivalent to the property of MDPs proved in [CD12] regarding the minimum expected penalty per stage. □

**Remark 3.** Here we explain why we require that the Markov chains defining the penalties of the DTS $\mathcal{T}$ be nontrivial. Assume there exists a state $u \in S_{\mathcal{P}}$ with a trivial Markov chain $\mathcal{M}_u$, i.e., the penalty in $u$ evolves deterministically, not probabilistically. If we visit $u$ infinitely many times in different (not necessarily consequent) points in time, the expected average penalty incurred in $u$ might differ from $\mathrm{pen}_E(u)$. That can cause violation of Proposition 1.

Finally, we are ready to define the strategy $C^{\mathsf{off}}_{\mathcal{P}}$. It is played in rounds, where each round consists of two phases, one for each subgoal. The first round starts at the beginning of the execution of the system in the initial state $s_{\mathcal{P}init}$ of $\mathcal{P}$. Let $i$ be the current round. In the first phase of the round the strategy $C^{\mathsf{off},\phi}_{\mathcal{P}}$ is applied until an accepting state of the ASCC $\mathcal{U}$ is reached. We use $k_i$ to denote the number of steps we played the strategy $C^{\mathsf{off},\phi}_{\mathcal{P}}$ in round $i$. Once the mission subgoal is fulfilled, the average subgoal becomes the current subgoal. In this phase, we play the strategy $C^{\mathsf{off},V}_{\mathcal{P}}$ until the number of completed surveillance cycles in the second phase of the current round is $l_i = \max\{j(\frac{1}{i}), i \cdot (k_i + |S_{\mathcal{U}}|) \cdot \mathrm{pen}_{\max}\}$, where $j(\frac{1}{i})$ is from Proposition 1.

**Theorem 2.** The offline control strategy $C$ that results from projecting the strategy $C^{\mathsf{off}}_{\mathcal{P}}$ from $\mathcal{P}$ to $\mathcal{T}$ solves Problem 3 and the corresponding APPC value is

$$V_{\mathcal{T},C}(s_{\mathsf{init}}) = V_{\mathcal{P},C^{\mathsf{off}}_{\mathcal{P}}}(s_{\mathcal{P}\mathsf{init}}) = V^*_{\mathcal{U}}.$$

*Proof.* To prove that the offline strategy $C$ satisfies the LTL formula $\phi$, we show that $C^{\mathsf{off}}_{\mathcal{P}}$ guarantees infinite number of visits of accepting states. Since the ASCC $\mathcal{U}$ is reachable from the initial state $s_{\mathcal{P}init}$ and from the construction of $C^{\mathsf{off},\phi}_{\mathcal{P}}$, it holds that every round of the strategy $C^{\mathsf{off}}_{\mathcal{P}}$ finishes after a finite number of steps and in every round an accepting state is visited.

To prove that the offline control strategy minimizes the APPC value among all strategies that satisfy the LTL formula $\phi$, we first present the algorithm to compute the minimum APPC value $V^*_{\mathcal{U}}$ that can be achieved in an ASCC $\mathcal{U}$ and a cycle $\mathsf{cyc}^V_{\mathcal{U}}$ of $\mathcal{U}$ witnessing the value. The idea is to reduce $\mathcal{U}$ to a TS $\mathcal{U}_{\mathsf{sur}}$ that contains only the states labeled with the surveillance proposition $a_{\mathsf{sur}}$ and then apply Karp's algorithm [Kar78] that finds a cycle with minimum value per edge also called the minimum mean cycle for a directed graph with values on edges. The value $V^*_{\mathcal{U}}$ and cycle $\mathsf{cyc}^V_{\mathcal{U}}$ are synthesized from the minimum mean cycle of $\mathcal{U}_{\mathsf{sur}}$.

Given an ASCC $\mathcal{U} = (S_{\mathcal{U}}, T_{\mathcal{U}}, AP, L_{\mathcal{P}})$ of $\mathcal{P}$, Algorithm 3 returns a DTS $\mathcal{U}_{\mathsf{sur}} = (S_{\mathcal{U}\mathsf{sur}}, \mathbf{T}_{\mathsf{sur}}, AP, L_{\mathcal{P}})$, a weight function $w_{\mathsf{sur}}$ and a function $\varsigma\colon \mathbf{T}_{\mathsf{sur}} \to \mathrm{Run}^{\mathcal{U}}_{\mathrm{fin}}$ with

**Algorithm 3** Reduction of an ASCC. We use . to denote the concatenation of two finite sequences and $\sum \mathrm{pen}_E(\sigma)$ is the sum of expected penalties $\mathrm{pen}_E(\sigma(i))$ for every state $\sigma(i)$ of a finite run $\sigma$.

Input: ASCC $\mathcal{U} = (S_\mathcal{U}, T_\mathcal{U}, AP, L_\mathcal{P})$ of $\mathcal{P}$, $w_\mathcal{P}$
Output: DTS $\mathcal{U}_{\mathsf{sur}} = (S_{\mathcal{U}_{\mathsf{sur}}}, \mathbf{T}_{\mathsf{sur}}, AP, L_\mathcal{P})$, weight function $w_{\mathsf{sur}}$ and function $\varsigma\colon \mathbf{T}_{\mathsf{sur}} \to \mathrm{Run}^{\mathcal{U}}_{\mathrm{fin}}$
let $X = (S_X, \mathbf{T}_X, AP, L_\mathcal{P})$ be a DTS equal to $\mathcal{U}$, $w_X = w_\mathcal{P}$ and $\varsigma_X\colon \mathbf{T}_X \to \mathrm{Run}^{\mathcal{U}}_{\mathrm{fin}}$ such that $\varsigma_X((u,u')) = u$ for every $(u,u') \in \mathbf{T}_X$
**while** $S_X \setminus S_{\mathcal{U}_{\mathsf{sur}}} \neq \emptyset$ **do**
    let $u \in S_X \setminus S_{\mathcal{U}_{\mathsf{sur}}}$
    **for all** $u_1, u_2 \in S_X, u_1 \neq u, u_2 \neq u, (u_1,u), (u,u_2) \in \mathbf{T}_X$ **do**
        **if** $(u_1,u_2) \notin \mathbf{T}_X$ **then**
            add $(u_1,u_2)$ to $\mathbf{T}_X$
            $\varsigma_X((u_1,u_2)) := \varsigma_X((u_1,u)).\varsigma_X((u,u_2))$
            $w_X((u_1,u_2)) := \sum \mathrm{pen}_E(\varsigma_X(u_1,u_2))$
        **else**
            **if** $\sum \mathrm{pen}_E(\varsigma_X((u_1,u_2))) \geq \sum \mathrm{pen}_E(\varsigma_X((u_1,u)).\varsigma_X((u,u_2)))$ **then**
                $\varsigma_X((u_1,u_2)) := \varsigma_X((u_1,u)).\varsigma_X((u,u_2))$
                $w_X((u_1,u_2)) := \sum \mathrm{pen}_E(\varsigma_X(u_1,u_2))$
            **end if**
        **end if**
    **end for**
    remove $u$ from $S_X$, and all adjacent transitions from $\mathbf{T}_X$
**end while**
return $X$, $w_X$ and $\varsigma_X$

the following properties. For the DTS $\mathcal{U}_{\mathsf{sur}}$ it holds that $(u,u') \in \mathbf{T}_{\mathsf{sur}}$ if and only if there exists a finite run in $\mathcal{U}$ from $u \in S_{\mathcal{U}_{\mathsf{sur}}}$ to $u' \in S_{\mathcal{U}_{\mathsf{sur}}}$ with one surveillance cycle, i.e., between $u$ and $u'$ no state labeled with $a_{\mathsf{sur}}$ is visited. Moreover, the run $\varsigma((u,u')) = u_0 \dots u_n$ is such that $u = u_0$ and $\sigma = u_0 \dots u_n u'$ is the finite run in $\mathcal{U}$ from $u$ to $u'$ with one surveillance cycle that minimizes the sum of expected penalties received during $\sigma$, denoted as the weight $w_{\mathsf{sur}}(\sigma)$, among all finite runs in $\mathcal{U}$ from $u$ to $u'$ with one surveillance cycle. The algorithm in Algorithm 3 builds $\mathcal{U}_{\mathsf{sur}}$ and the function $\varsigma$ by eliminating the states from $S_\mathcal{U} \setminus S_{\mathcal{U}_{\mathsf{sur}}}$ one by one, in arbitrary order. Figure 3.4 demonstrates elimination of one such state on an illustrative example.

We apply the Karp's algorithm to the oriented graph with vertices $S_{\mathcal{U}_{\mathsf{sur}}}$, edges $\mathbf{T}_{\mathsf{sur}}$ and values on edges $w_{\mathsf{sur}}$. Let $\mathsf{cyc}_{\mathcal{U}_{\mathsf{sur}}} = u_0 \dots u_m$ be the minimum mean cycle of this graph. We have

$$V^*_\mathcal{U} = \frac{1}{m+1} \sum_{i=0}^{m} \mathrm{pen}_E\big(\varsigma((u_i, u_{i+1 \bmod (m+1)}))\big),$$

$$\mathsf{cyc}^V_\mathcal{U} = \varsigma((u_0,u_1)).\ \dots\ .\varsigma((u_{m-1},u_m)).\varsigma((u_m,u_0)).$$

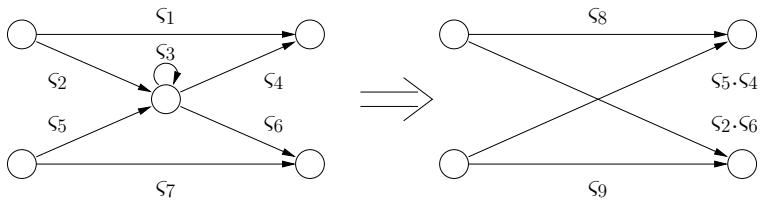



**Figure 3.4:** Elimination of one state of an ASCC during the algorithm in Algorithm 3. The finite run $\varsigma_8$ is the one of the runs $\varsigma_1$ and $\varsigma_2.\varsigma_4$ that minimizes the sum of expected penalties in the states of the run. Similarly, $\varsigma_9$ is one of the runs $\varsigma_7$ and $\varsigma_5.\varsigma_6$.

It can be easily shown that all states of the cycle $\mathsf{cyc}^V_{\mathcal{U}}$ are distinct. It follows that

$$\frac{1}{|\mathsf{cyc}^V_{\mathcal{U}} \cap S_{\mathcal{U}\mathsf{sur}}|} \sum_{i=0}^{m} \mathrm{pen}_E(c_i) = V^*_{\mathcal{U}}.$$

When the APPC value and the corresponding cycle is computed for every ASCC of $\mathcal{P}$, we choose the ASCC that minimizes the APPC value. We denote this ASCC $\mathcal{U} = (S_{\mathcal{U}}, T_{\mathcal{U}}, AP, L_{\mathcal{P}})$ and $\mathsf{cyc}^V_{\mathcal{U}} = c_0 \dots c_m$. Every strategy solving Problem 3 must achieve APPC value $V^*_{\mathcal{U}}$.

Since the offline strategy $C$ is a projection of the strategy $C^{\mathsf{off}}_{\mathcal{P}}$, we have $V_{\mathcal{T},C}(s_{init}) = V_{\mathcal{P},C^{\mathsf{off}}_{\mathcal{P}}}(s_{\mathcal{P}init})$. To show that $V_{\mathcal{P},C^{\mathsf{off}}_{\mathcal{P}}}(s_{\mathcal{P}init}) = V^*_{\mathcal{U}}$, let $\epsilon_i = \frac{1}{i}$ for round $i$. From Proposition 1 and the fact that $l_i = \max\{j(\frac{1}{i}), i\cdot(k_i+|S_{\mathcal{U}}|)\cdot \mathrm{pen}_{\max}\}$ it follows that the average penalty per surveillance cycle in $i$-th round after its completion is at most

$$\begin{aligned}
&\frac{k_i \cdot \mathrm{pen}_{\max} + |S_{\mathcal{U}}| \cdot \mathrm{pen}_{\max} + l_i(V^*_{\mathcal{U}} + \epsilon_i)}{l_i} \\
&\qquad \leq V^*_{\mathcal{U}} + \epsilon_i + \frac{1}{i} \qquad (l_i \geq i \cdot (k_i + |S_{\mathcal{U}}|) \cdot \mathrm{pen}_{\max}) \\
&\qquad = V^*_{\mathcal{U}} + \frac{2}{i}
\end{aligned}$$

with probability at least $1 - \frac{1}{i}$. Therefore, in the limit, the average cumulative penalty per surveillance cycle $V^*_{\mathcal{U}}$ with probability 1, independently on penalty profile. □

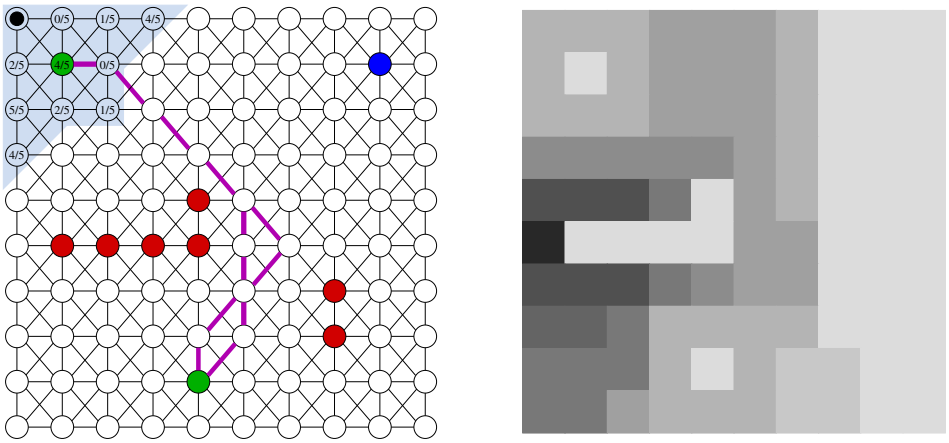
### Complexity

The size of a BA for an LTL formula $\phi$ is in $2^{\mathcal{O}(|\phi|)}$, where $|\phi|$ is the size of $\phi$ [GO01a]. However, the actual size of the BA is in practice often quite small. The size of the product $\mathcal{P}$ is in $\mathcal{O}(|S| \cdot 2^{\mathcal{O}(|\phi|)})$. To compute the minimum weights $w^*((s,q),(s',q'))$ between every two states of $\mathcal{P}$ we use Floyd-Warshall algorithm

taking $\mathcal{O}(|S_{\mathcal{P}}|^3)$ time. Tarjan's algorithm [Tar72] is used to compute the set $\mathrm{SCC}(\mathcal{P})$ in $\mathcal{O}(|S_{\mathcal{P}}| + |T_{\mathcal{P}}|)$ time. The reduction of an ASCC $\mathcal{U}$ can be computed in $\mathcal{O}(|S_{\mathcal{U}}| \cdot |T_{\mathcal{U}}|^2)$ time. The Karp's algorithm [Kar78] finds the optimal APPC value and corresponding cycle in $\mathcal{O}(|S_{\mathcal{U}\mathsf{sur}}| \cdot |\mathbf{T}_{\mathsf{sur}}|)$ time. The main pitfall of the algorithm is to compute the number $j(\frac{1}{i})$ of surveillance cycles needed in the second phase of the current round $i$ according to Proposition 1. Intuitively, we need to consider the finite run $\sigma^{\sharp k}$ induced by the strategy $C_{\mathcal{P}}^{\mathsf{off},V}$ from the current state that contains $k = 1$ surveillance cycles, and compute the sum of probabilities $\Pr_{(s,q)}^{\mathcal{P},C_{\mathcal{P}}}(\mathrm{Cyl}((\sigma^{\sharp k}, \tau)))$ for every $\tau$ with the average cumulative penalty per surveillance cycle less or equal to $V_{\mathcal{U}}^* + \frac{2}{i}$. If the total probability is at least $1 - \frac{1}{i}$, we set $j(\frac{1}{i}) = k$, otherwise we increase $k$ and repeat the process. For every $k$, there exist up to $\mathrm{pen}_{\max}$ to the power of $|\sigma^{\sharp k}|$ sequences $\tau$. This issue can be partially overcome using the rule presented below.

### Usability

The strategy $C_{\mathcal{P}}^{\mathsf{off}}$ is not a finite-memory strategy in general. The reason is that the number of surveillance cycles that we need to perform in the second phase of round $i$ is increasing with $i$. Note that in the special case when there exists a cycle $\mathsf{cyc}_{\mathcal{U}}^{V}$ of the SCC $\mathcal{U}$ corresponding to $V_{\mathcal{U}}^*$ that contains an accepting state, the memoryless strategy $C_{\mathcal{P}}^{\mathsf{off},V}$ for the average subgoal maps to a strategy of $\mathcal{T}$ solving Problem 3, which is therefore in the worst case finite-memory. To improve the memory usage we suggest the following technical improvements. Let $\overline{C_{\mathcal{P}}^{\mathsf{off}}}$ be the strategy for $\mathcal{P}$ that results from applying the following rule to the strategy $C_{\mathcal{P}}^{\mathsf{off}}$. Let $i$ be the current round and $k_i$ the number of steps in the first phase of the round. In the second phase we proceed as follows. After completion of every surveillance cycle, check whether the average penalty per surveillance cycle incurred in the current round of the execution is above $V_{\mathcal{U}}^* + \frac{2}{i}$, for the significance of this value see the proof of Theorem 2. If yes, continue with the second phase of round $i$, otherwise start new round $i+1$. Also, avoid performing the expensive computation of the value $j(\frac{1}{i})$ until it is necessary, i.e., only compute the value once the number of surveillance cycles performed in the second phase of the round $i$ is $i \cdot (k_i + |S_{\mathcal{U}}|) \cdot \mathrm{pen}_{\max}$ and the average penalty per surveillance cycle in the round $i$ is still above $V_{\mathcal{U}}^* + \frac{2}{i}$. Note that the strategy $\overline{C_{\mathcal{P}}^{\mathsf{off}}}$ is dependent on penalty profiles but it is equivalent to the strategy $C_{\mathcal{P}}^{\mathsf{off}}$ in the meaning that it provably guarantees infinite number of visits to the set $F_{\mathcal{P}}$ of accepting states and the APPC value of $\overline{C_{\mathcal{P}}^{\mathsf{off}}}$ is equal to the APPC value of $C_{\mathcal{P}}^{\mathsf{off}}$, i.e., it is $V_{\mathcal{U}}^*$. Formally, the strategy $\overline{C_{\mathcal{P}}^{\mathsf{off}}}$ may still require infinite memory. However, in our simulations in Section 3.2.5 we demonstrate that the memory usage and the amount of computation performed while using the strategy $\overline{C_{\mathcal{P}}^{\mathsf{off}}}$ is significantly decreased comparing to the strategy $C_{\mathcal{P}}^{\mathsf{off}}$. More specifically, the number of surveillance cycles performed in the second phase of each round drops dramatically and the value $j(\frac{1}{i})$ for round $i$ needs to be computed only rarely, if ever.




(a) (b)

**Figure 3.5:** (a) Transition system for the delivery system example. The projection of an optimal APPC cycle is shown in magenta. For convenience, in (b), we also show the graphical representation of the function $p\colon S \to (0,1)$ from Figure 3.3c.

**Remark 4.** By considering only the expected values of penalties in states of the TS, the penalties can be seen as static in our offline approach. This makes Problem 3 related to the problem formulated in [WTM12a] for systems with static costs. In [WTM12a], the optimality is achieved by persistently increasing the number of visits to states under surveillance that leads to neglecting the remaining part of the LTL specification over time. The authors in [WTM12a] disregard such a strategy as undesirable. In our case, the optimal strategy $C_{\mathcal{P}}^{\mathsf{off}}$ performs only as many surveillance cycles in every round as are needed for the expected average penalty per surveillance cycle to be close enough to the optimal value $V_{\mathcal{U}}^*$ with probability 1, see Proposition 1 and the proof of Theorem 2. This allows us to efficiently implement $C_{\mathcal{P}}^{\mathsf{off}}$ using feedback, see the discussion on usability above. We demonstrate in Section 3.2.5 that unlike the one in [WTM12a], our optimal strategy does not lead to the undesirable, ever-increasing number of visits of surveillance states.

**Example 4.** For the delivery system from Example 3, the Büchi automaton generated for the LTL formula in Equation 3.14 using [GO01b] has 16 states. The product $\mathcal{P}$ of the transition system and the automaton contains 2 ASCCs and the chosen, optimal ASCC $\mathcal{U}$ has 568 states. The projection of a cycle $\mathsf{cyc}_{\mathcal{U}}^{V}$ providing the minimum expected average cumulative penalty per surveillance cycle is depicted in magenta in Figure 3.5 and the APPC value associated with the cycle is $V_{\mathcal{U}}^* = 4.35$.

The offline control has the following structure. In the first phase of the first round, the aim of the strategy $C_{\mathcal{P}}^{\mathsf{off}}$ for the product is to reach an accepting state of the ASCC $\mathcal{U}$ using a finite run of the minimum possible weight. When projected

to the TS, the robot starts from the base location and moves to the top delivery location using a finite run of the minimum possible weight. The first phase is completed one step after the visit of the delivery location, when the product reaches an accepting state, in total of $k_1 = 8$ steps. The control proceeds to the second phase of the first round. In the product, we first reach the optimal APPC cycle $\mathsf{cyc}^V_{\mathcal{U}}$ as fast as possible and then follow the cycle until $l_1 = \max\{576, j(1)\}$ surveillance cycles are completed. At the beginning of the second phase, even though the robot might be in a state that lies on the cycle shown in magenta in Figure 3.5, the product is not yet on the cycle $\mathsf{cyc}^V_{\mathcal{U}}$. Therefore, in the TS the control does not yet follow the cycle depicted in Figure 3.5. In the product, the closest state of the optimal APPC cycle that can be reached is the one that projects to the bottom delivery location. That means, the robot follows one of the shortest finite runs from the top to the bottom delivery location. The round then proceeds with $l_1$ alternative visits of the two delivery locations by following the cycle shown in magenta in Figure 3.5. Once completed, the control proceeds to the second round. Every round $i \geq 2$ starts from either the top or bottom delivery location. In the first phase of the round, the robot first moves from the top, or bottom, delivery location to the base location and then continues delivering the packages by moving to the bottom, or top, delivery location, respectively. In both cases the first phase of the round ends after 15 steps, i.e., $l_i = 15$ for every $i \geq 2$. In the second phase, the robot keeps delivering packages between the top and the bottom delivery locations until the number of visits of the two locations is $l_i = \max\{i \cdot 583, j(\frac{1}{i})\}$.

Note that the robot follows a predetermined sequence of transitions given by the strategy $C^{\mathsf{off}}_{\mathcal{P}}$, without considering the penalties incurred or observed in real-time. Also, the robot visits the base location only during the first phase of rounds and as the number $l_i$ grows very fast with $i$, the time between consecutive visits to the base grows rapidly. By applying the rule from the above discussion on the usability of the offline control, we can visit the base more often by monitoring the amount of penalties incurred in the current round and decreasing the number of performed surveillance cycles. As we demonstrate in Section 3.2.5, the rule dramatically improves the visit rate of the base while not affecting the convergence to the optimal value $V^*_{\mathcal{U}}$.

### Online control

In this section, we construct a strategy for $\mathcal{T}$ that also provably guarantees the satisfaction of given LTL formula but provides better or equal APPC than the offline strategy. Hence, in the context of Problem 2 it is a better solution than the offline control. It is however dependent on penalty profiles, so it cannot be considered as a solution to Problem 3.

The online strategy is again computed as a projection of a suitable strategy $C^{\mathsf{on}}_{\mathcal{P}}$ for the product $\mathcal{P}$. We obtain $C^{\mathsf{on}}_{\mathcal{P}}$ by locally improving the offline strategy $C^{\mathsf{off}}_{\mathcal{P}}$ from the previous section. Intuitively, we compare applying $C^{\mathsf{off}}_{\mathcal{P}}$ for several

steps to reach a specific state or set of states of $\mathcal{P}$ to executing different local paths to reach the same state or set of states. We consider a finite set of finite runs leading to the state or set, containing the finite run induced by $C_{\mathcal{P}}^{\mathsf{off}}$. We then choose the one that is expected to minimize the average cumulative penalty per surveillance cycle incurred in the current round, taking into account the currently observed penalties within the visibility range and their simulation in the next $h$ time units, where $h \in \mathbb{N}$ is a user-defined planning horizon. The first transition of the chosen run is applied in the system. The process continues until the state, or set, is reached, and then it starts over again. For the same reasons as in the offline algorithm, the resulting strategy for $\mathcal{T}$ is an infinite-memory strategy. We again propose technical improvements to reduce memory usage and computational cost that result in a strategy for $\mathcal{T}$ that is equivalent to the online strategy, i.e., it guarantees the satisfaction of the given formula and provides the same APPC value. The online control strategy is a heuristic and we suggest a procedure to design a whole class of heuristic online strategies with similar properties.

In this section, we use the following notation. Let $\sigma_{\text{all}} \in \text{Run}_{\text{fin}}^{\mathcal{P}}(s_{\mathcal{P}init})$ denote the finite run executed by $\mathcal{P}$ so far. Let $i$ be the current round of strategy $C_{\mathcal{P}}^{\mathsf{on}}$ and $\sigma_i = u_{i,0} \dots u_{i,k}$ the finite run executed so far in this round, i.e., $u_{i,k}$ is the current state of $\mathcal{P}$. We use $t_{i,0}, \dots, t_{i,k}$ to denote the points in time when the states $u_{i,0}, \dots, u_{i,k}$ were visited, respectively.

The optimization function $f \colon \text{Run}_{\text{fin}}^{\mathcal{P}}(u_{i,k}) \to \mathbb{R}_0^+$ assigns every finite run $\sigma = u_0 \dots u_n$ starting from the current state value $f(\sigma)$ that is the expected average cumulative penalty per surveillance cycle that would be incurred in round $i$, if run $\sigma$ was to be executed next, i.e.,

$$f(\sigma) = \frac{\sum_{j=0}^{k} \text{pen}(u_{i,j}, t_{i,j}) + \sum_{j=1}^{n} \overline{\text{pen}}(u_j, t_{i,k} + w_{\mathcal{P}}(\sigma^{\to j}))}{\sharp(\sigma_i.\sigma^{1\to})}, \tag{3.16}$$

where

$$\overline{\text{pen}}(u_j, t_{i,k} + w_{\mathcal{P}}(\sigma^{\to j})) = \begin{cases} \text{pen}_{\text{sim}}(u_j, t_{i,k}, \text{pen}(u_j, t_{i,k}), w_{\mathcal{P}}(\sigma^{\to j})) \\ \qquad \text{if } u_j \in \text{Vis}(u_{i,k}), w_{\mathcal{P}}(\sigma^{\to j}) \leq h \\ \text{pen}_E(u_j) \\ \qquad \text{otherwise.} \end{cases}$$

Intuitively, if the penalty in state $u_j$ is visible in the current time moment and $u_j$ would have been visited within the next $h$ time units in run $\sigma$, the value $\overline{\text{pen}}(u_j, t_{i,k} + w_{\mathcal{P}}(\sigma^{\to j}))$ refers to the simulated expected penalty in $u_j$ at the time of its visit, as defined in Section 3.2.2. Otherwise, we do not simulate the penalty over time and consider only its expected value $\text{pen}_E(u_j)$.

For a set of states $X \subseteq S_{\mathcal{P}}$, we define a *shortening indicator function* $I_X \colon T_{\mathcal{P}} \to$

$\{0,1\}$ such that for $(u,u') \in T_{\mathcal{P}}$, we have

$$I_X\big((u,u')\big) = \begin{cases} 1 & \text{if } w^*_{\mathcal{P}}(u,X) > w^*_{\mathcal{P}}(u',X), \\ 0 & \text{otherwise.} \end{cases} \tag{3.17}$$

In words, the indicator has value 1 if the transition leads strictly closer to $X$, and 0 otherwise.

Now we are ready to formally define the strategy $C^{\mathsf{on}}_{\mathcal{P}}$. In the first phase of every round, we locally improve the strategy $C^{\mathsf{off},\phi}_{\mathcal{P}}$ computed in the previous section that aims to visit an accepting state of the chosen ASCC $\mathcal{U}$. In each step of the resulting strategy $C^{\mathsf{on},\phi}_{\mathcal{P}}$, we consider the set $\mathrm{Run}_\phi(u_{i,k})$ of all finite runs that start in the current state $u_{i,k}$ and lead to an accepting state from the set $F_{\mathcal{U}}$ with all transitions shortening in the indicator $I_{F_{\mathcal{U}}}$ defined according to Equation 3.17, i.e.,

$$\begin{aligned}\mathrm{Run}_\phi(u_{i,k}) = \{\sigma \in \mathrm{Run}^{\mathcal{P}}_{\mathrm{fin}}(u_{i,k}) \,|\mathrm{last}(\sigma) \in F_{\mathcal{U}} \text{ and} \\ \forall 0 \le j \le |\sigma|-1 \colon I_{F_{\mathcal{U}}}((\sigma(j),\sigma(j+1))) = 1\}.\end{aligned}$$

Let $\sigma \in \mathrm{Run}_\phi(u_{i,k})$ be the run that minimizes the optimization function $f$ from Equation 3.16. Then $C^{\mathsf{on},\phi}_{\mathcal{P}}(\sigma_{\mathrm{all}}) = \sigma(1)$. Just like in the offline algorithm, the strategy $C^{\mathsf{on},\phi}_{\mathcal{P}}$ is applied until a state from the set $F_{\mathcal{U}}$ is visited. In the second phase of strategy $C^{\mathsf{on}}_{\mathcal{P}}$, we locally improve the strategy $C^{\mathsf{off},V}_{\mathcal{P}}$ for the average subgoal from the previous section and we use $C^{\mathsf{on},V}_{\mathcal{P}}$ to denote the resulting strategy. At the beginning of the second phase of the current round $i$, we aim to reach the cycle $\mathsf{cyc}^V_{\mathcal{U}} = c_0 \dots c_m$ of the ASCC $\mathcal{U}$ and we use the same idea that is used in the first phase above. To be specific, we define $C^{\mathsf{on},V}_{\mathcal{P}}(\sigma_{\mathrm{all}}) = \sigma(1)$, where $\sigma$ is the run minimizing $f$ from the set

$$\begin{aligned}\mathrm{Run}^{\mathsf{init}}_V(u_{i,k}) = \{\sigma \in \mathrm{Run}^{\mathcal{P}}_{\mathrm{fin}}(u_{i,k}) \,|\mathrm{last}(\sigma) \in \mathsf{cyc}^V_{\mathcal{U}}, \text{ and} \\ \forall 0 \le j \le |\sigma|-1 \colon I_{\mathsf{cyc}^V_{\mathcal{U}}}((\sigma(j),\sigma(j+1))) = 1\}.\end{aligned}$$

Once a state $c_a \in \mathsf{cyc}^V_{\mathcal{U}}$ of the cycle is reached, we continue as follows. Let $c_b \in \mathsf{cyc}^V_{\mathcal{U}}$ be the first state labeled with $a_{\mathsf{sur}}$ that is visited from $c_a$ if we follow the cycle. Until we reach the state $c_b$, the optimal finite run $\sigma$ is chosen from the set

$$\begin{aligned}\mathrm{Run}_V(u_{i,k}) = \{\sigma \in \mathrm{Run}^{\mathcal{P}}_{\mathrm{fin}}(u_{i,k}) \,|\mathrm{last}(\sigma) = c_b, \text{ and} \\ \forall 0 \le j \le |\sigma|-1 \colon I_{c_b}((\sigma(j),\sigma(j+1))) = 1 \text{ or} \\ |\sigma_{c_a \to u_{i,k}}| + |\sigma| \le b - a + 2 \mod (m+1)\},\end{aligned}$$

where $\sigma_{c_a \to u_{i,k}}$ is the finite run already executed in $\mathcal{P}$ from state $c_a$ to the current state $u_{i,k}$. Intuitively, the set contains every finite run starting from the current state and leading to $c_b$ that either has all transitions shortening in $I_{c_b}$ or the length of the finite run is such that if we were to perform the finite run, the length of the

performed run from $c_a$ to $c_b$ would not be longer than following the cycle from $c_a$ to $c_b$. When state $c_b$ is reached, we restart the above procedure with $c_a = c_b$. The strategy $C_{\mathcal{P}}^{\mathsf{on},V}$ is performed until $l_i = \max\{j(\frac{1}{i}), i \cdot (k_i + |S_{\mathcal{U}}|) \cdot \mathrm{pen}_{\max}\}$ surveillance cycles are completed in the second phase of the current round $i$, where $k_i$ is the number of steps of the first phase and $j$ is from Proposition 1 (the value is given by the strategy $C_{\mathcal{P}}^{\mathsf{off}}$).

**Theorem 3.** The online control strategy $C$ that results from projecting strategy $C_{\mathcal{P}}^{\mathsf{on}}$ from $\mathcal{P}$ to $\mathcal{T}$ guarantees satisfaction of the LTL specification $\phi$. Moreover, it holds that the online control strategy provides lower or equal expected average cumulative penalty per surveillance cycle than the offline control strategy, i.e., $V_{\mathcal{T},C}(s_{\mathsf{init}}) = V_{\mathcal{P},C_{\mathcal{P}}^{\mathsf{on}}}(s_{\mathcal{P}\mathsf{init}}) \leq V_{\mathcal{U}}^{*}$.

*Proof.* To prove that the online control strategy satisfies the LTL specification, we show that under every penalty profile $\Delta$, the run $\rho_{C_{\mathcal{P}}^{\mathsf{on}},\Delta}$ induced by strategy $C_{\mathcal{P}}^{\mathsf{on}}$ visits the set $F_{\mathcal{P}}$ of accepting states infinitely many times. From the definitions of sets $\mathrm{Run}_{\phi}(u_{i,k})$, $\mathrm{Run}_{V}^{\mathsf{init}}(u_{i,k})$ and $\mathrm{Run}_{V}(u_{i,k})$ it follows that every round of $C_{\mathcal{P}}^{\mathsf{on}}$ ends after a finite number of steps and at least one accepting state is visited in every round. The inequality $V_{\mathcal{P},C_{\mathcal{P}}^{\mathsf{on}}}(s_{\mathcal{P}\mathsf{init}}) \leq V_{\mathcal{U}}^{*}$ follows directly from the design of the strategy $C_{\mathcal{P}}^{\mathsf{on}}$. □

As we show in Section 3.2.5, computation of the APPC value of the online strategy according to Equation 3.12 is intractable even for reasonably small examples. Nevertheless, we demonstrate that in some cases, it can be computed based on the specifics of the transition system, and in general, it can be estimated from statistical results. The case studies also demonstrate that even though the construction of the online strategy does not guarantee strict decrease in APPC value in theory, these strategies often result in a significant improvement.

### Complexity

To design the online strategy $C_{\mathcal{P}}^{\mathsf{on}}$, we first compute the strategy $C_{\mathcal{P}}^{\mathsf{off}}$. The complexity of one step of the online strategy is as follows. The cardinality of the set of finite runs $\mathrm{Run}_{\phi}(u_{i,k})$ grows exponentially with the minimum weight $w_{\mathcal{P}}^{*}(u_{i,k}, F_{\mathcal{U}})$. Analogously, the same holds for sets $\mathrm{Run}_{V}^{\mathsf{init}}(u_{i,k})$ and $\mathrm{Run}_{V}(u_{i,k})$, and the set $\mathsf{cyc}_{V}^{*}$ or one of its surveillance states. In the following discussion on the usability of the online control, we propose a simple rule to simplify the computations and effectively use the algorithm in real time. Also, the complexity of one step of the strategy $C_{\mathcal{P}}^{\mathsf{on}}$ grows exponentially with the user-defined planning horizon $h$ and the system-specific visibility range $v$. Hence, $h$ should be chosen wisely. One should also keep in mind that generally, the higher the planning horizon, the better local improvement.

### Usability

Just like the offline strategy, the online strategy is not finite-memory in general. To reduce memory usage, we construct new strategy $\overline{C_{\mathcal{P}}^{\mathsf{on}}}$ from $C_{\mathcal{P}}^{\mathsf{on}}$ by applying the rule to reduce the number of performed surveillance cycles in the second phase of every round that was introduced for the offline control in the previous section. Moreover, to simplify the computation in one step of the online control, we allow the user to define parameter $W_{\max} \geq \max\{w_{\mathcal{P}}((u,u')) \mid (u,u') \in T_{\mathcal{P}}\}$ that serves as follows. In the first phase of round $i$ and at the beginning of the second phase, when not yet on the optimal APPC cycle, we only consider prefixes of the finite runs from the sets $\mathrm{Run}_{\phi}(u_{i,k})$, $\mathrm{Run}_{V}^{\mathsf{init}}(u_{i,k})$ or $\mathrm{Run}_{V}(u_{i,k})$, and the set $\mathsf{cyc}_{V}^{*}$ of weight at most $W_{\max}$. In the second phase of a round, when the optimal cycle has already been reached, if the weight of the fragment of the cycle from $c_a$ to $c_b$ is more than $W_{\max}$, we first optimize the run from $c_a$ to an intermediate state $c_b'$ for which it holds that the weight of the fragment of the cycle from $c_a$ to $c_b'$ is at most $W_{\max}$ but highest possible. Finally, we postpone the computation of the value $j(\frac{1}{i})$ for round $i$ for as long as possible in the same manner as for the offline strategy. The strategy $\overline{C_{\mathcal{P}}^{\mathsf{on}}}$ is equivalent to $C_{\mathcal{P}}^{\mathsf{on}}$ in the meaning that it provably satisfies the LTL formula $\phi$ and has the same APPC value as the strategy $C_{\mathcal{P}}^{\mathsf{on}}$. The improvement of the usability of the online control is demonstrated in Section 3.2.5.

**Example 5.** The online control for the delivery system from Example 3 that locally improves the offline control described in Example 4 works as follows. Consider planning horizon $h = 9$. In every step of the first phase of every round, the robot considers all finite runs that start in the current state, continue to the base location and end in the delivery location that should be visited next. The robot performs the first transition of the run minimizing the function from Equation 3.16. When computing the value for a finite run, the penalties in states that are within the visibility range and would be visited within 9 time units are simulated. In the second phase, the robot locally improves the finite run leading from one delivery location to the other, while visiting at most as many states as there are on the cycle shown in Figure 3.5 between the two delivery locations. The number of surveillance cycles in the second phase of round $i$ is $l_i = \max\{i \cdot (k_i + 576), j(\frac{1}{i})\}$, where $k_i$ is the number of steps in the first phase of the round. Unlike in the offline control in Example 4, number $k_i$ might differ from round to round.

Note that the set of finite runs considered in one step of the online control can be very large, especially in the first phases of rounds when the finite runs are considerably long. We can use the parameter $W_{\max}$ introduced in the discussion on the usability of the online control above to decrease the number and weight of finite runs considered in every step of the control. By applying the rest of the rules from the discussion, we can also decrease the number of visits to delivery locations in every round and thus visit the base location more often.

**Remark 5.** The online control introduced in this section is in fact a heuristic. We can formulate other heuristics that would construct a strategy satisfying Theorem 3. For example, consider a strategy that is constructed from the offline strategy $C_{\mathcal{P}}^{\mathsf{off}}$ in the same way as the strategy $C_{\mathcal{P}}^{\mathsf{on}}$, i.e., deploying sets $\mathrm{Run}_{\phi}(u_{i,k})$ and $\mathrm{Run}_{V}^{\mathsf{init}}(u_{i,k})$, except in the second phase of every round, once a state $c_a \in \mathsf{cyc}_{\mathcal{U}}^{V}$ on the cycle is reached, the optimal finite run is chosen from the set defined as

$$\begin{aligned}\mathrm{Run}_V(u_{i,k}) = \{\sigma \in \mathrm{Run}_{\mathrm{fin}}^{\mathcal{P}}(u_{i,k}) \mid & \mathrm{last}(\sigma) = c_b, \text{ and} \\ & \forall 0 \leq j \leq |\sigma| - 1 \colon I_{c_b}((\sigma(j), \sigma(j+1))) = 1 \text{ or} \\ & |\sigma_{c_a \to u_{i,k}}| + |\sigma| \leq 2 \cdot \big(b - a + 2 \mod (m+1)\big)\},\end{aligned}$$

i.e., we consider all finite runs consisting only of transitions shortening in $I_{c_b}$ and all finite runs leading to the target state $c_b$ with length at most twice the length of following the cycle from $c_a$ to $c_b$. We refer to the projection of this strategy from $\mathcal{P}$ to $\mathcal{T}$ as the *modified online control*.

We can define other heuristics in a similar way by changing only the definition of the sets of finite runs $\mathrm{Run}_{\phi}(u_{i,k}), \mathrm{Run}_{V}(u_{i,k})$. However in order to guarantee satisfaction of Theorem 3, the sets must satisfy the following conditions. The definition of $\mathrm{Run}_{\phi}(u_{i,k})$ guarantees that an accepting state from $F_{\mathcal{U}}$ is always visited after a finite number of steps. The definition of $\mathrm{Run}_{V}(u_{i,k})$ guarantees a visit of the cycle $\mathsf{cyc}_{\mathcal{U}}^{V}$ after at most $|S_{\mathcal{U}}|$ steps and once a state $c_a \in \mathsf{cyc}_{\mathcal{U}}^{V}$ on the cycle is reached, the set $\mathrm{Run}_{V}(u_{i,k})$ guarantees visit of the state $c_b$ in a finite number of steps.

### 3.2.5 Implementation and Case Studies

#### Implementation

The framework presented in this section is implemented in our simulation tool ConTool [SMv13]. Input transition system can be defined in DOT language and then visualized using Graphviz [GN00]. The MCs defining the penalties in states are loaded from a text file. We use LTL2BA [GO01b] to generate a Büchi automaton for given LTL formula. The user can choose to simulate the offline, online or the modified online control from Remark 5. All three control strategies implement the rules introduced to reduce memory usage and computational costs. After specifying additional parameters such as visibility range, planning horizon and the parameter $W_{\max}$ for the online control, the simulation tool allows to observe the control one transition at a time.

#### Case study 1: Delivery system

The offline and online control strategies for the delivery system from Example 3 were described in Example 4 and 5, respectively. Here we report on the results we obtained from executing 10 runs of 30 rounds for all three types of control strategies using ConTool. In simulations of the online and modified online controls, we

used parameter $W_{\max} = 9$. We present the analysis of the average penalty per surveillance cycle at the end of every round in Figure 3.6. For the offline control, the value gradually converges to the (optimal) APPC value 4.35 of the offline control, marked as a red line in Figure 3.6. On the other hand, for both online and modified online control strategies, the average is below 4.35 due to the local improvement based on local sensing. Due to the size and density of the transition system, it is intractable to compute the exact APPC value as defined in Equation 3.12 for the online and modified online control. Nevertheless, from Figure 3.6 we can observe that the average penalty per surveillance cycle stabilizes in timely manner at approximately 4.16 for the online control and 4.05 for the modified online control.

The number of surveillance cycles performed in the second phase of every round $i$ using the rules from Section 3.2.4 was always less than $i \cdot (k_i + 568)$ for all three types of control, i.e., the second phase always ended due to the fact that the average incurred in the round was below the threshold $V^*_{\mathcal{U}} + \frac{2}{i}$. That means, we were never forced to compute the value $j(\frac{1}{i})$. The maximum number of surveillance cycles performed in the second phase of a round of offline control strategy was 636, average was only 29 and median was 8. Using online control strategy, the maximum number of surveillance cycles performed in the second phase of a round was 10, the average was 2 and the median was 1. Similarly for the modified online strategy, the maximum was 9 and both average and median were 1, i.e., for all three control strategies the rules from Section 3.2.4 reduced the number of performed surveillance cycles in every round substantially from thousands to only tens or few hundreds and thus allowed to visit the base location much more often. Moreover, the number of surveillance cycles in the second phase of a round did not evolve monotonically, rather randomly.

We ran the simulations on a Lenovo laptop with Windows 7, Intel Pentium CPU 2.00 GHz and 3GB RAM. The offline strategy was computed in 45 seconds on average. One step of the online and modified online control took 150 milliseconds and 7.5 seconds on average, with 100 milliseconds and 12 seconds deviation, respectively.

#### Case study 2: Stock market

The second case study we use to evaluate our framework models a simple stock market and a broker that performs one action on the market at a time. He can sell or buy stocks, or decide to wait. We assume that the system can be modeled as the transition system depicted in Figure 3.7a. The system starts in state $s_0$. The broker decides his next action in state $s_1$ and he can choose from 5 different buying and selling orders. All transitions have weight 1. In Figure 3.7b, we list the transition matrices of the Markov chains that define penalties in states of the TS. The initial distribution is always the uniform distribution over the possible values of the penalty in a given state. Only the states $s_2, \ldots, s_6$ can have non-zero penalty modeling the fact that only buying and selling stocks has any value.

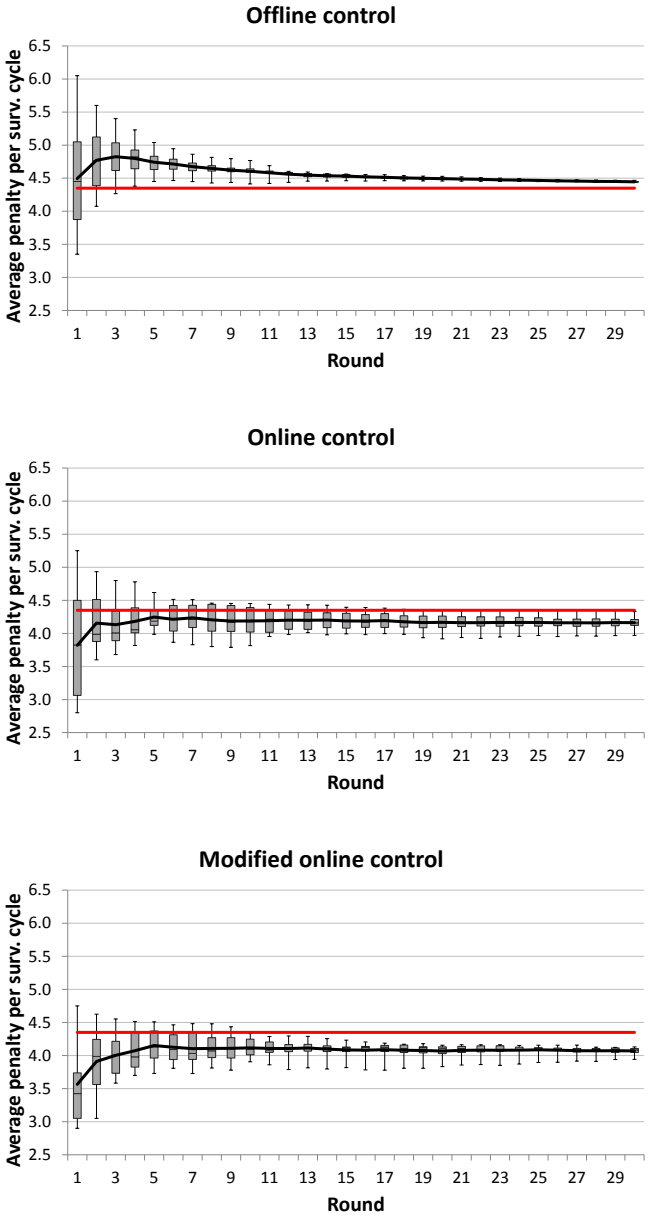
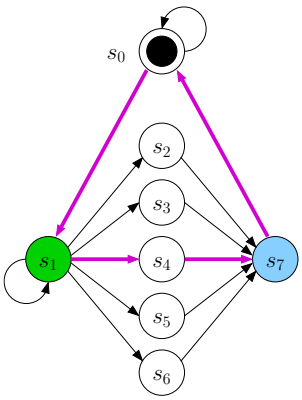



**Figure 3.6:** Evolution of the average penalty per surveillance cycle obtained in simulations of the offline, online and modified online control for the delivery system case study. For each control, the statistics is built on 10 individual runs of 30 rounds each. The red line marks the optimal APPC value and the black line shows the mean over executed runs.




**(a)**

$$A_0 = A_1 = A_7 = \begin{matrix} \\ 0 \end{matrix} \begin{pmatrix} 1.0 \end{pmatrix} \qquad A_2 = \begin{matrix} \\ 2 \end{matrix} \begin{pmatrix} 1.0 \end{pmatrix}$$

$$A_3 = \begin{matrix} & 1 & 2 \\ 1 \\ 2 \end{matrix} \begin{pmatrix} 0.8 & 0.2 \\ 0.2 & 0.8 \end{pmatrix} \qquad A_4 = \begin{matrix} & 0 & 1 & 2 & 3 \\ 0 \\ 1 \\ 2 \\ 3 \end{matrix} \begin{pmatrix} 0.1 & 0.5 & 0.3 & 0.1 \\ 0.1 & 0.5 & 0.3 & 0.1 \\ 0.1 & 0.5 & 0.3 & 0.1 \\ 0.1 & 0.5 & 0.3 & 0.1 \end{pmatrix}$$

$$A_5 = \begin{matrix} & 0 & 1 & 2 & 3 \\ 0 \\ 1 \\ 2 \\ 3 \end{matrix} \begin{pmatrix} 0.2 & 0.8 & 0 & 0 \\ 0 & 0.2 & 0.8 & 0 \\ 0 & 0 & 0.2 & 0.8 \\ 0.8 & 0 & 0 & 0.2 \end{pmatrix} \qquad A_6 = \begin{matrix} & 0 & 1 & 2 & 3 \\ 0 \\ 1 \\ 2 \\ 3 \end{matrix} \begin{pmatrix} 0 & 1.0 & 0 & 0 \\ 0 & 0 & 1.0 & 0 \\ 0 & 0 & 0 & 1.0 \\ 0.9 & 0 & 0 & 0.1 \end{pmatrix}$$

**(b)**

**Figure 3.7:** (a) Transition system that models a broker acting on a simple stock market. In state $s_1$, in green, the broker chooses one of the five buying or selling orders. All transitions have weight 1. The optimal cycle with respect to the expected average penalty per surveillance cycle is shown in magenta. (b) Matrices of Markov chains that define penalties in states of the transition system. Matrix $A_i$ describes penalty in state $s_i$.

Finally, state $s_7$, in blue, can be seen as an evaluation state, where the gains and losses are counted. The visibility range $v$ is 4, i.e., the broker can always observe penalties in all states.

The mission for the broker is to make an infinite number of orders and, at the same time, to minimize the penalty incurred per order. We model this requirement with LTL formula

$$\mathbf{GF}\, a \ \wedge\ \mathbf{GF}\, a_{\mathsf{sur}},$$

where $a$ and $a_{\mathsf{sur}}$ are true in state $s_7$. The Büchi automaton generated for the formula using [GO01b] has 4 states, the product has one ASCC with 25 states. The optimal APPC cycle projected to the transition system is shown in magenta in Figure 3.7a and the optimal APPC value is 1.4.

We present statistical results that we obtained by running 10 runs of 30 rounds for each of the offline, online and modified online control strategies. In all simulations we used planning horizon $h = 9$ and we did not use the parameter $W_{\max}$. In Figure 3.8, we plot the average penalty per surveillance cycle at the end of every round. The red line marks the APPC value of the offline control 1.4. For the offline control, the obtained value converges to the optimal APPC value fairly fast. We can observe considerable improvement for both online and modified online control. The reason is the following. In the offline control, when in the second phase of a round the broker always chooses the action leading to state $s_4$ that has the minimum expected value $\mathrm{pen}_E$. On the other hand, in the same situation when using online control, the broker always chooses the next action according to the simulated expected values of penalties $\mathrm{pen}_{\mathrm{sim}}$ rather than their expected values $\mathrm{pen}_E$. Finally, using the modified online control, the broker is allowed to wait up to three time units in state $s_1$ and only then decide to buy or sell. Note that the penalty in state $s_6$ gradually increases from 0 to 3 and then with probability 90% it drops to 0 again. The broker waits in $s_1$ until the penalty in state $s_6$ has value 3 and then moves to $s_6$. Just like for the first case study, it is intractable to compute the APPC value for the online and modified online control. However, from the discussion above, we can conclude that the APPC value of the modified online control is 0.3 and this fact can also be observed in Figure 3.8. Based on Figure 3.8, the APPC value of the online control strategy is approximately 0.64.

The number of surveillance cycles in the second phase of every round using the rules in Section 3.2.4 was always below $i \cdot (k_i + 25)$ for all three strategies, i.e., every round ended with the average penalty per surveillance cycle in the round dropping below the threshold $V^*_{\mathcal{U}} + \frac{2}{i}$ and we never needed to compute the value $j(\frac{1}{i})$. The maximum number of surveillance cycles performed in the second phase of a round was 288 for the offline control, 12 for the online control and 5 for the modified online control, and the median was 1 in all three cases. Hence, the improvement of the rules in Section 3.2.4 is again remarkable. In all three cases, the number of surveillance cycles in rounds did not evolve monotonically, rather randomly.

We ran the simulations on a Lenovo laptop with Windows 7, Intel Pentium

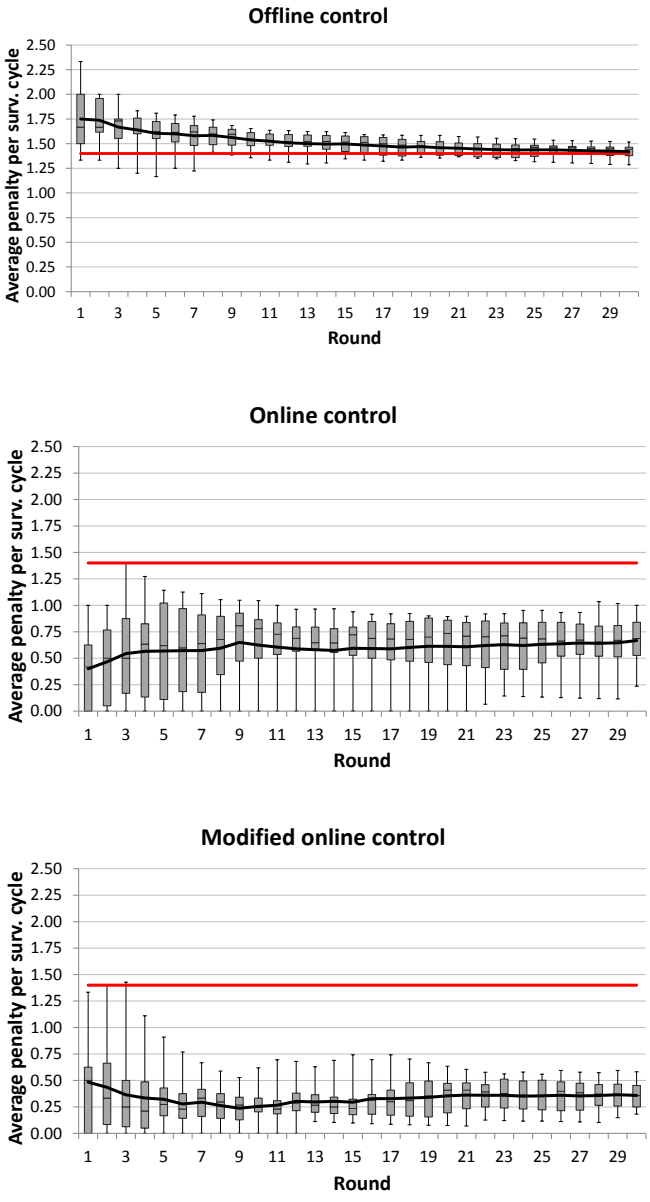



**Figure 3.8:** Evolution of the average penalty per surveillance cycle attained in simulations of the offline, online and modified online control for the stock market case study. For each control strategy, the statistics is built on 10 individual runs of 30 rounds each. The red line marks the optimal APPC value and the black line shows the mean over executed runs.

CPU 2.00 GHz and 3GB RAM. The offline strategy took 0.5 seconds on average to compute. One step of the online and modified online control strategies always took under one millisecond.

### 3.2.6 Conclusion

In this section, we considered the problem of synthesizing an optimal control strategy for a deterministic transition system under temporal logic constraints. We assumed real-valued penalties with probabilistic behaviors in the states of the system. We constructed a control strategy that, while guaranteeing satisfaction of an LTL formula, minimizes the expected average penalty per visit of a desired set of states. We also presented (a class of) control strategies that use local sensing of the penalties in real time and simulation of their values over finite horizon to improve the average penalty per visit of the set of states in every execution of the system.

## 3.3 Probabilistic Systems with Penalties

### 3.3.1 Motivation

In this section, we consider a variation of the control problem discussed in Section 3.2. While in the previous section, we considered a deterministic system with probabilistic penalties incurred with every transition to a new state, here we consider a probabilistic system modeled as a Markov decision process introduced in Definition 7, where every transition incurs a static real-valued penalty. Given an LTL formula with persistent surveillance, our goal is again to synthesize a control strategy that minimizes the average expected penalty between visits of states under surveillance subject to the temporal logic constraints.

The above problem was previously investigated in [DSBR11]. Using dynamic programming techniques, the authors design a solution that is sub-optimal in the general case. The solution becomes optimal if the MDP $\mathcal{M}$ satisfies certain conditions described in [DSBR11] and also for a fragment of LTL specified in [CTB12]. Using an adaptation of the framework presented in Section 3.2 to MDPs, in this section we design a control strategy that is optimal in the general case, i.e., for any MDP and any LTL formula. Just like the algorithms presented in Section 3.2, the designed control builds on recent results from game theory [CD12].

The results presented in this section are based on our results in [SvB13a]. The rest of the section is organized as follows. We start with formally stating the problem in Section 3.3.2. The solution together with discussion on its properties and complexity is presented in Section 3.3.3. Finally, Section 3.3.4 contains experimental results.

### 3.3.2 Problem Formulation

Consider an initialized MDP $\mathcal{M} = (S, Act, P, AP, L)$ with initial state $s_{\mathsf{init}} \in S$ and a penalty function $\mathrm{pen}\colon S \times Act \to \mathbb{R}_0^+$ that specifies the penalty $\mathrm{pen}(s, \alpha)$ associated with applying an action $\alpha \in Act$ in a state $s \in S$. Finally, consider a specification given as an LTL formula with persistent surveillance as defined in Section 2.2.4. Recall that a formula with persistent surveillance is

$$\phi = \varphi \wedge \mathbf{GF}\, a_{\mathsf{sur}}, \tag{3.18}$$

where $\varphi$ is an arbitrary LTL formula over $AP$ and $a_{\mathsf{sur}} \in AP$ is a surveillance proposition associated with states $S_{\mathsf{sur}} = \{s \in S \mid a_{\mathsf{sur}} \in L(s)\}$. We extend the notation from Section 2.2.4 and use $\sharp(\sigma)$ to denote the number of complete surveillance cycles in a finite run $\sigma$ such that $\mathrm{last}(\sigma) \in S_{\mathsf{sur}}$, otherwise $\sharp(\sigma)$ denotes the number of complete surveillance cycles in $\sigma$ plus one.

For a strategy $C$ for $\mathcal{M}$, we define the average expected cumulative penalty per surveillance cycle (APPC) in an analogous way as in Section 3.2. APPC in the MDP $\mathcal{M}$ under a strategy $C$ as a function $V_{\mathcal{M},C}\colon S \to \mathbb{R}_0^+$ such that for a state $s \in S$

$$V_{\mathcal{M},C}(s) = \limsup_{k\to\infty} \sum_{\rho_C \in \mathrm{Run}^{\mathcal{M},C}(s)} \Pr_s^{\mathcal{M},C}(\mathrm{Cyl}(\rho_C^{\to k}) \cdot \frac{\sum_{i=0}^{k} \mathrm{pen}(\rho_C(i), C(\rho_C^{\to i}))}{\sharp(\rho_C^{\to k})}. \tag{3.19}$$

The problem we consider in this section can be formally stated as follows.

**Problem 4.** ***Given***

- *an initialized MDP $\mathcal{M} = (S, Act, P, AP, L)$ with initial state $s_{\mathsf{init}} \in S$,*
- *an LTL formula $\phi$ over $AP$ with persistent surveillance,*
- *penalties* $\mathrm{pen}\colon S \times Act \to \mathbb{R}_0^+$,

***find*** *a control strategy $C\colon \mathrm{Run}_{\mathrm{fin}}^{\mathcal{M}} \to Act$ such that*

*(i) $C$ satisfies $\phi$ with probability 1 starting from $s_{\mathsf{init}}$ and*

*(ii) among all strategies satisfying (i), $C$ minimizes the APPC value $V_{\mathcal{M},C}(s_{\mathsf{init}})$ defined in Equation 3.19.*

In the special case when every state of $\mathcal{M}$ is a surveillance state, Problem 4 aims to find a strategy that minimizes the average expected penalty per stage among all strategies almost-surely satisfying $\phi$. The optimization problem of minimizing the average expected penalty per stage also known as average cost per stage (ACPS) in an MDP, without considering any correctness specification, is a well studied problem in optimal control [Ber12]. It holds that there always exists a stationary strategy that minimizes the ACPS value starting from the initial state. In our approach to Problem 4, we use techniques for solving the ACPS problem to find a strategy that minimizes the APPC value. Similarly as in Section 3.2, we construct a control strategy solving Problem 4 as a combination of a strategy that ensures the almost-sure satisfaction of the specification $\phi$ and a strategy that guarantees the minimum APPC value among all strategies that do not cause immediate unrepairable violation of $\phi$.

### 3.3.3 Solution

In this section, we describe the solution to Problem 4 in detail. The algorithm works with the synchronous product of the MDP $\mathcal{M}$ and a deterministic Rabin automaton $\mathcal{A}$ for the formula $\phi$. We first define the product and discuss the construction of an optimal strategy for the product on an intuitive level. We follow with the construction of the two strategies for the formula satisfaction and APPC optimization, respectively. Finally, the two strategies are combined to obtain an optimal strategy for the product that maps to the desired strategy for $\mathcal{M}$.

#### Product Construction

To employ the automata-based approach to control synthesis as described in Section 2.5, we translate the LTL formula $\phi$ to a deterministic Rabin automaton $\mathcal{A}$ using techniques discussed in Section 2.2. Next, we construct the synchronous product of the MDP $\mathcal{M}$ and the DRA $\mathcal{A}$.

**Definition 19.** *Let $\mathcal{M} = (S, Act, P, AP, L)$ be an initialized MDP with initial state $s_{\mathsf{init}} \in S$ and $\mathcal{A} = (Q, 2^{AP}, \delta, q_0, Acc)$ be a DRA. The synchronous product of $\mathcal{M}$ and $\mathcal{A}$ is the initialized MDP*

$$\mathcal{P} = \mathcal{M} \times \mathcal{A} = (S_{\mathcal{P}}, Act, P_{\mathcal{P}}, AP_{\mathcal{P}}, L_{\mathcal{P}}),$$

*where $S_{\mathcal{P}} = S \times Q$, $P_{\mathcal{P}}((s,q),\alpha,(s',q')) = P(s,\alpha,s')$ if $q' = \delta(q, L(s))$ and 0 otherwise, $AP_{\mathcal{P}} = AP \cup Q$, $L_{\mathcal{P}}((s,q)) = L(s) \cup \{q\}$. The initial state of $\mathcal{P}$ is $s_{\mathcal{P}\mathsf{init}} = (s_{\mathsf{init}}, q_0)$.*

The product $\mathcal{P}$ can be viewed as an initialized MDP with a Rabin accepting condition. Therefore, we adopt the definitions of a run $\rho$, a finite run $\sigma$, and sets $\mathrm{Run}^{\mathcal{P}}((s,q))$, $\mathrm{Run}^{\mathcal{P}}$, $\mathrm{Run}^{\mathcal{P}}_{\mathrm{fin}}((s,q))$ and $\mathrm{Run}^{\mathcal{P}}_{\mathrm{fin}}$ from Section 2.3.2. Similarly, a strategy $C_{\mathcal{P}}$ for $\mathcal{P}$, runs $\rho_{C_{\mathcal{P}}}, \sigma_{C_{\mathcal{P}}}$ induced by $C_{\mathcal{P}}$, and the probability measure $\Pr^{\mathcal{P},C_{\mathcal{P}}}_{(s,q)}$ are defined in the same way as for an MDP. We also adopt the definitions of an end component and maximal end component. Finally, we adopt the penalties by letting $\mathrm{pen}_{\mathcal{P}}((s,q),\alpha) = \mathrm{pen}(s,\alpha)$.

Using the projection $\pi_1$ on the first component, every (finite) run of $\mathcal{P}$ projects to a (finite) run of $\mathcal{M}$ and vice versa, for every (finite) run of $\mathcal{M}$, there exists a (finite) run of $\mathcal{P}$ that projects to it. Analogous correspondence exists between strategies for $\mathcal{P}$ and $\mathcal{M}$. It holds that the projection of a finite-memory strategy for $\mathcal{P}$ is also finite-memory. More importantly, for the product $\mathcal{P}$ of $\mathcal{M}$ and $\mathcal{A}$, the probability of satisfying the accepting condition $Acc$ of $\mathcal{A}$ under a strategy $C_{\mathcal{P}}$ for $\mathcal{P}$ starting from the initial state $s_{\mathcal{P}\mathsf{init}}$, i.e.,

$$\Pr^{\mathcal{P},C_{\mathcal{P}}}_{s_{\mathcal{P}\mathsf{init}}}\Big(\bigvee_{(E,F)\in Acc} (\mathbf{FG}(\neg E) \wedge \mathbf{GF}\, F)\Big),$$

is equal to the probability of satisfying the formula $\phi$ in the MDP $\mathcal{M}$ under the projected strategy $C$ starting from the initial state $s_{\mathsf{init}}$.

**Definition 20.** *Let $\mathcal{P} = (S_\mathcal{P}, Act, P_\mathcal{P}, AP_\mathcal{P}, L_\mathcal{P})$ be the product of an MDP $\mathcal{M}$ and a DRA $\mathcal{A}$. An accepting end component (AEC) of $\mathcal{P}$ is defined as an end component $\mathcal{N} = (S_\mathcal{N}, Act_\mathcal{N}, P_\mathcal{P}, AP_\mathcal{P}, L_\mathcal{P})$ of $\mathcal{P}$ for which there exists a pair $(E, F)$ in the acceptance condition of $\mathcal{A}$ such that $L_\mathcal{P}(S_\mathcal{N}) \cap E = \emptyset$ and $L_\mathcal{P}(S_\mathcal{N}) \cap F \neq \emptyset$. We say that $\mathcal{N}$ is accepting with respect to the pair $(E, F)$.*

*An AEC $\mathcal{N}$ is called maximal (MAEC) if there is no AEC $\mathcal{N}'$ such that $\mathcal{N}' \neq \mathcal{N}$, $S_\mathcal{N} \subseteq S_{\mathcal{N}'}$, $Act_\mathcal{N}((s,q)) \subseteq Act_{\mathcal{N}'}((s,q))$ for every $(s,q) \in S_\mathcal{P}$ and $\mathcal{N}$ and $\mathcal{N}'$ are accepting with respect to the same pair. We use* $\mathrm{AEC}(\mathcal{P})$ *and* $\mathrm{MAEC}(\mathcal{P})$ *to denote the set of all accepting end components and maximal accepting end components of $\mathcal{P}$, respectively.*

Note that MAECs that are accepting with respect to the same pair are always disjoint. However, MAECs that are accepting with respect to different pairs can intersect.

### Optimal strategy for the product – intuition

Using the correspondence between strategies for $\mathcal{P}$ and $\mathcal{M}$, an optimal strategy $C$ for $\mathcal{M}$ is found as a projection of a strategy $C_\mathcal{P}$ for $\mathcal{P}$ which almost-surely satisfies the accepting condition $Acc$ of $\mathcal{A}$ and at the same time, minimizes the APPC value $V_{\mathcal{P},C_\mathcal{P}}(s_{\mathcal{P}\mathsf{init}})$ among all strategies for $\mathcal{P}$ that almost-surely satisfy $Acc$.

From the discussion above it follows that a necessary condition for almost-sure satisfaction of the accepting condition $Acc$ by a strategy $C_\mathcal{P}$ for $\mathcal{P}$ is that there exists a set $\mathsf{maec} \subseteq \mathrm{MAEC}(\mathcal{P})$ of MAECs such that $C_\mathcal{P}$ leads the product from the initial state to $\mathsf{maec}$. Moreover, for every MAEC $\mathcal{N} \in \mathrm{MAEC}(\mathcal{P})$, the minimum APPC value $V^*_\mathcal{N}((s,q))$ that can be obtained in $\mathcal{N}$ starting from a state $(s,q) \in S_\mathcal{N}$ is equal for all the states of $\mathcal{N}$ and we denote this value $V^*_\mathcal{N}$. The strategy $C_\mathcal{P}$ is constructed in two steps.

First, we find a set $\mathsf{maec}^*$ of MAECs of $\mathcal{P}$ and a strategy $C_0$ that leads $\mathcal{P}$ from the initial state to the set $\mathsf{maec}^*$. We require that $C_0$ and $\mathsf{maec}^*$ minimize the weighted average of the values $V^*_\mathcal{N}$ for $\mathcal{N} \in \mathsf{maec}^*$. The strategy $C_\mathcal{P}$ applies $C_0$ from the initial state until $\mathcal{P}$ enters the set $\mathsf{maec}^*$.

Second, we solve the problem of how to control the product once a state of an MAEC $\mathcal{N} \in \mathsf{maec}^*$ is visited. Intuitively, we combine two finite-memory strategies, $C^\phi_\mathcal{N}$ for the almost-sure satisfaction of the accepting condition $Acc$ and $C^V_\mathcal{N}$ for maintaining the average expected cumulative penalty per surveillance cycle. To satisfy both objectives, the strategy $C_\mathcal{P}$ is played in rounds. In each round, we first apply the strategy $C^\phi_\mathcal{N}$ and then the strategy $C^V_\mathcal{N}$, each for a specific (finite) number of steps.

### Finding an optimal set of MAECs

Let $\mathrm{MAEC}(\mathcal{P})$ be the set of all MAECs of the product $\mathcal{P}$ that can be computed as follows. For every pair $(E, F) \in Acc$, we create a new MDP from $\mathcal{P}$ by removing

all its states with label in $B$ and the corresponding actions. For the new MDP, we use one of the algorithms in [dA97, CY95, CH11] to compute the set of all its MECs. Finally, for every MEC, we check whether it contains a state with label in $G$.

In this section, the aim is to find a set $\mathsf{maec}^* \subseteq \mathrm{MAEC}(\mathcal{P})$ and a strategy $C_0$ for $\mathcal{P}$ that satisfy conditions formally stated below. Since the strategy $C_0$ will only be used to enter the set $\mathsf{maec}^*$, it is constructed as a partial function.

**Definition 21.** *Let $\mathcal{M} = (S, Act, P, AP, L)$ be an MDP. A partial strategy for $\mathcal{M}$ is a partial function $\zeta \colon \mathrm{Run}^{\mathcal{M}}_{\mathrm{fin}} \to Act$, where if $\zeta(\sigma)$ is defined for $\sigma \in \mathrm{Run}^{\mathcal{M}}_{\mathrm{fin}}$, then $\zeta(\sigma) \in Act(\mathrm{last}(\sigma))$.*

A partial memoryless strategy for $\mathcal{M}$ can also be considered as a partial function $\zeta \colon S \to Act$ or a subset $\zeta \subseteq S \times Act$. The set $\mathrm{Run}^{\mathcal{M},\zeta}$ of runs of $\mathcal{M}$ under $\zeta$ contains all infinite runs of $\mathcal{M}$ that follow $\zeta$ and all those finite runs $\sigma$ of $\mathcal{M}$ under $\zeta$ for which $\zeta(\mathrm{last}(\sigma))$ is not defined. A finite run of $\mathcal{M}$ under $\zeta$ is then a finite prefix of a run under $\zeta$. The probability measure $\Pr^{\mathcal{M},\zeta}_s$ is defined in the same manner as in Section 2.3.2. We also extend the semantics of LTL formulas to finite words. For example, a formula $\mathbf{FG}\,\phi$ is satisfied by a finite word if in some non-empty suffix of the word $\phi$ always holds.

The conditions on $\mathsf{maec}^*$ and $C_0$ are as follows. First, the partial strategy $C_0$ leads $\mathcal{P}$ to the set $\mathsf{maec}^*$, i.e.,

$$\Pr^{\mathcal{P},C_0}_{s_{\mathcal{P}\mathsf{init}}}(\mathbf{FG}\,(\bigcup_{\mathcal{N}\in\mathsf{maec}^*} S_{\mathcal{N}})) = 1. \tag{3.20}$$

Second, we require that $\mathsf{maec}^*$ and $C_0$ minimize the value

$$\sum_{\mathcal{N}\in\mathsf{maec}^*} \Pr^{\mathcal{P},C_0}_{s_{\mathcal{P}\mathsf{init}}}(\mathbf{FG}\,S_{\mathcal{N}}) \cdot V^*_{\mathcal{N}}. \tag{3.21}$$

The procedure to compute the optimal APPC value $V^*_{\mathcal{N}}$ for an MAEC $\mathcal{N}$ of $\mathcal{P}$ is described in the next section. Assume we already computed this value for each MAEC of $\mathcal{P}$. The algorithm to find the set $\mathsf{maec}^*$ and partial strategy $C_0$ is based on an algorithm for stochastic shortest path (SSP) problem. The SSP problem is one of the basic optimization problems for MDPs. Given an initialized MDP and its state $t$, the goal is to find a strategy under which the MDP almost-surely reaches the state $t$, so called terminal state, while minimizing the expected cumulative penalty. If there exists at least one strategy almost-surely reaching the terminal state, then there exists a stationary optimal strategy. For details and algorithms see [Ber12].

The partial strategy $C_0$ and the set $\mathsf{maec}^*$ are computed as follows. First, we create a new MDP $\mathcal{P}'$ from $\mathcal{P}$ by considering only those states of $\mathcal{P}$ that can reach the set $\mathrm{MAEC}(\mathcal{P})$ with probability 1 and their corresponding actions. The MDP $\mathcal{P}'$ can be computed using backward reachability from the set $\mathrm{MAEC}(\mathcal{P})$. If $\mathcal{P}'$

does not contain the initial state $s_{\mathcal{P}\mathsf{init}}$, there exists no solution to Problem 4. Otherwise, we add a new state $t$ and for every MAEC $\mathcal{N} \in \text{MAEC}(\mathcal{P}') = \text{MAEC}(\mathcal{P})$, we add a new action $\alpha_{\mathcal{N}}$ to $\mathcal{P}'$. From each state $(s,q) \in S_{\mathcal{N}}, \mathcal{N} \in \text{MAEC}(\mathcal{P}')$, we define a transition under $\alpha_{\mathcal{N}}$ to $t$ with probability 1 and set its penalty to $V^*_{\mathcal{N}}$. All other penalties in the MDP are set to 0. Finally, we solve the SSP problem for $\mathcal{P}'$ and the state $t$ as the terminal state. Let $C_{SSP}$ be the resulting stationary optimal strategy for $\mathcal{P}'$. For every $(s,q) \in S_{\mathcal{P}}$, we define $C_0((s,q)) = C_{SSP}((s,q))$ if the action $C_{SSP}((s,q))$ does not lead from $(s,q)$ to $t$, $C_0((s,q))$ is undefined otherwise. The set $\mathsf{maec}^*$ is the set of all MAECs $\mathcal{N}$ for which there exists a state $(s,q)$ such that $C_{SSP}((s,q)) = \alpha_{\mathcal{N}}$.

**Proposition 2.** The set $\mathsf{maec}^*$ and the partial stationary strategy $C_0$ resulting from the above algorithm satisfy the conditions in Equation 3.20 and Equation 3.21.

*Proof.* Both conditions follow directly from the fact that the strategy $C_{SSP}$ is an optimal solution to the SSP problem for $\mathcal{P}'$ and $t$. □

### Optimizing APPC value in an MAEC

In this section, we compute the minimum APPC value $V^*_{\mathcal{N}}$ that can be attained in an MAEC $\mathcal{N} \in \text{MAEC}(\mathcal{P})$ and construct the corresponding strategy for $\mathcal{N}$.

Essentially, we reduce the problem of computing the minimum APPC value to the problem of computing the minimum ACPS value by reducing $\mathcal{N}$ to an MDP such that every state of the reduced MDP is labeled with the surveillance proposition $a_{\mathsf{sur}}$.

Let $\mathcal{N} = (S_{\mathcal{N}}, Act_{\mathcal{N}}, P_{\mathcal{P}}, AP_{\mathcal{P}}, L_{\mathcal{P}})$ be an MAEC of $\mathcal{P}$. Since it is an MAEC, there exists a state $(s,q) \in S_{\mathcal{N}}$ with $a_{\mathsf{sur}} \in L_{\mathcal{P}}((s,q))$. Let $S_{\mathcal{N}_{\mathsf{sur}}}$ denote the set of all such states in $S_{\mathcal{N}}$. We reduce $\mathcal{N}$ to an MDP

$$\mathcal{N}_{\mathsf{sur}} = (S_{\mathcal{N}_{\mathsf{sur}}}, \mathbf{Act}_{\mathsf{sur}}, P_{\mathsf{sur}}, AP_{\mathcal{P}}, L_{\mathcal{P}})$$

using Algorithm 4. For the sake of readability, we use singletons such as $u$ instead of pairs such as $(s,q)$ to denote the states of $\mathcal{N}$. The MDP $\mathcal{N}_{\mathsf{sur}}$ is constructed from $\mathcal{N}$ by eliminating states from $S_{\mathcal{N}} \backslash S_{\mathcal{N}_{\mathsf{sur}}}$ one by one in arbitrary order. The actions $\mathbf{Act}_{\mathsf{sur}}$ are partial stationary strategies for $\mathcal{N}$ in which we remember all the states and actions we eliminated. Later we prove that the transition probability $P_{\mathsf{sur}}(u, \zeta, u')$ for states $u, u' \in S_{\mathcal{N}_{\mathsf{sur}}}$ and an action $\zeta \in \mathbf{Act}_{\mathsf{sur}}(u)$ is the probability that in $\mathcal{N}$ under the partial stationary strategy $\zeta$, if we start from the state $u$, the next state that will be visited from the set $S_{\mathcal{N}_{\mathsf{sur}}}$ is the state $u'$, i.e., the first surveillance cycle is completed by visiting $u'$. The penalty $\text{pen}_{\mathsf{sur}}(u, \zeta)$ is the expected cumulative penalty gained in $\mathcal{N}$ using partial stationary strategy $\zeta$ from $u$ until we reach a state in $S_{\mathcal{N}_{\mathsf{sur}}}$.

In Figure 3.9, we demonstrate the reduction on an example using the notation introduced in Algorithm 4. On the left side, we see a part of an MAEC $\mathcal{N}$ with five

**Algorithm 4** Reduction of an MAEC $\mathcal{N}$ to $\mathcal{N}_{\mathsf{sur}}$

Input: $\mathcal{N} = (S_{\mathcal{N}}, Act_{\mathcal{N}}, P_{\mathcal{P}}, AP_{\mathcal{P}}, L_{\mathcal{P}})$, penalties $\text{pen}_{\mathcal{P}}$
Output: $\mathcal{N}_{\mathsf{sur}} = (S_{\mathcal{N}\mathsf{sur}}, \mathbf{Act}_{\mathsf{sur}}, P_{\mathsf{sur}}, AP_{\mathcal{P}}, L_{\mathcal{P}})$, penalties $\text{pen}_{\mathsf{sur}}$
let $X = (S_X, \mathbf{Act}_X, P_X, AP_{\mathcal{P}}, L_{\mathcal{P}})$ be an MDP and $\text{pen}_X$ penalties such that
- $S_X := S_{\mathcal{N}}$,
- for $u \in S_X : \mathbf{Act}_X(u) := \{\zeta_\alpha \mid \zeta_\alpha = \{(u, \alpha)\}, \alpha \in Act_{\mathcal{N}}(u)\}$,
- for $u, u' \in S_X, \zeta \in \mathbf{Act}_X : P_X(u, \zeta, u') := P_{\mathcal{P}}(u, \zeta(u), u')$,
- for $u \in S_X, \zeta \in \mathbf{Act}_X : \text{pen}_X(u, \zeta) := \text{pen}_{\mathcal{P}}(u, \zeta(u))$

**while** $S_X \backslash S_{\mathcal{N}\mathsf{sur}} \neq \emptyset$ **do**
  let $u \in S_X \backslash S_{\mathcal{N}\mathsf{sur}}$
  **for all** $\zeta \in \mathbf{Act}_X(u)$ **do**
    **if** $P_X(u, \zeta, u) < 1$ **then**
      **for all** $u_{from} \in S_X, \zeta_{old} \in \mathbf{Act}_X(u_{from})$ **do**
        **if** $P_X(u_{from}, \zeta_{old}, u) > 0$ and $\zeta_{old}, \zeta$ do not conflict **then**
          $\zeta_{new} := \zeta_{old} \cup \zeta$
          add $\zeta_{new}$ to $\mathbf{Act}_X(u_{from})$
          for every $u_{to} \in S_X$:

$$P_X(u_{from}, \zeta_{new}, u_{to}) := P_X(u_{from}, \zeta_{old}, u_{to}) + P_X(u_{from}, \zeta_{old}, u) \cdot \frac{P_X(u, \zeta, u_{to})}{1 - P_X(u, \zeta, u)}$$

$$\text{pen}_X(u_{from}, \zeta_{new}) := \text{pen}_X(u_{from}, \zeta_{old}) + P_X(u_{from}, \zeta_{old}, u) \cdot \frac{\text{pen}_X(u, \zeta)}{1 - P_X(u, \zeta, u)}$$

          remove $\zeta_{old}$ from $\mathbf{Act}_X(u_{from})$
        **end if**
      **end for**
    **end if**
    remove $\zeta$ from $\mathbf{Act}_X(u)$
  **end for**
  remove $u$ from $S_X$
**end while**
return $X$

states and two actions. First, we build an MDP $X = (S_X, \mathbf{Act}_X, P_X, AP_{\mathcal{P}}, L_{\mathcal{P}})$ and penalties $\text{pen}_X$ from $\mathcal{N}$ by transforming every action of every state to a partial stationary strategy with a single pair given by the state and the action. The MDP $X$ is used in the algorithm as an auxiliary MDP to store the current version of the reduced system. Assume we want to reduce the state $u$. We consider all "incoming" and "outgoing" actions of $u$ and combine them pairwise as follows. There is only one outgoing action from $u$ in $X$, namely $\zeta$, and only one incoming action, namely action $\zeta_{old}$ of state $u_{from}$. Since $\zeta$ and $\zeta_{old}$ do not conflict as partial stationary strategies on any state of $\mathcal{N}$, we merge them to create a new partial stationary strategy $\zeta_{new}$ that is an action of $u_{from}$. The transition probability $P_X(u_{from}, \zeta_{new}, u_{to})$ for a state $u_{to}$ of $X$ is computed as the sum of the transition probability $P_X(u_{from}, \zeta_{old}, u_{to})$ of transiting from $u_{from}$ to $u_{to}$ using the old action

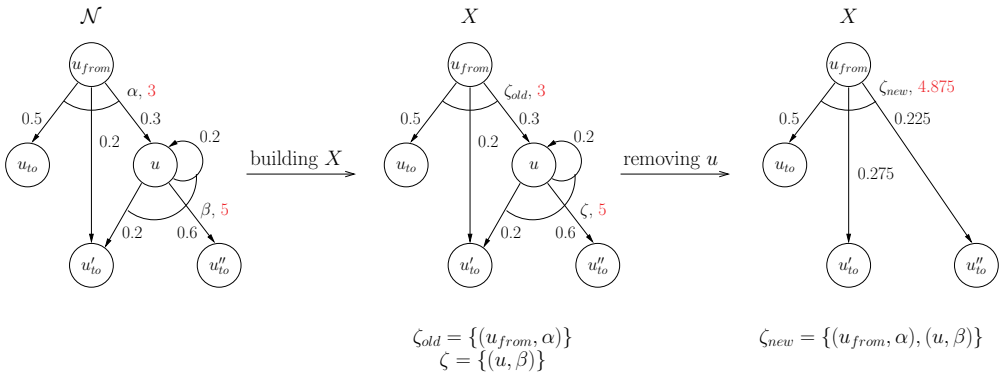



**Figure 3.9:** Illustration of Algorithm 4. A part of an MAEC $\mathcal{N}$ is shown in the left. An auxiliary MDP $X$ is constructed by transforming actions of $\mathcal{N}$ to partial stationary strategies. The MDP $X$ after eliminating the state $u$ is shown on the right. The penalties associated with actions are depicted in red.

$\zeta_{old}$ and the probability of entering $u_{to}$ by first transiting from $u_{from}$ to $u$ using $\zeta_{old}$ and from $u$ eventually reaching $u_{to}$ using $\zeta$. The penalty $\mathrm{pen}_X(u_{from}, \zeta_{new})$ is the expected cumulative penalty gained starting from $u_{from}$ by first applying action $\zeta_{old}$ and if we transit to $u$, applying $\zeta$ until a state different from $u$ is reached. Now that we considered every pair of an incoming and outgoing action of $u$, the state $u$ and its incoming and outgoing actions are reduced. The modified MDP $X$ is depicted on the right side of Figure 3.9.

**Proposition 3.** Let $\mathcal{N} = (S_{\mathcal{N}}, Act_{\mathcal{N}}, P_{\mathcal{P}}, AP_{\mathcal{P}}, L_{\mathcal{P}})$ be an MAEC with penalties $\mathrm{pen}_{\mathcal{P}}$ and $\mathcal{N}_{\mathsf{sur}} = (S_{\mathcal{N}_{\mathsf{sur}}}, \mathbf{Act}_{\mathsf{sur}}, P_{\mathsf{sur}}, AP_{\mathcal{P}}, L_{\mathcal{P}})$ and $\mathrm{pen}_{\mathsf{sur}}$ its reduction resulting from Algorithm 4. The minimum APPC value that can be attained in $\mathcal{N}_{\mathsf{sur}}$ starting from any of its states is the same and we denote it $V^*_{\mathcal{N}_{\mathsf{sur}}}$. There exists a stationary strategy $C^V_{\mathcal{N}_{\mathsf{sur}}}$ for $\mathcal{N}_{\mathsf{sur}}$ that attains this value regardless of the starting state in $\mathcal{N}_{\mathsf{sur}}$. Both $V^*_{\mathcal{N}_{\mathsf{sur}}}$ and $C^V_{\mathcal{N}_{\mathsf{sur}}}$ can be computed as a solution to the ACPS problem for $\mathcal{N}_{\mathsf{sur}}$. It holds that $V^*_{\mathcal{N}} = V^*_{\mathcal{N}_{\mathsf{sur}}}$ and from $C^V_{\mathcal{N}_{\mathsf{sur}}}$, one can construct a finite-memory strategy $C^V_{\mathcal{N}}$ for $\mathcal{N}$ which regardless of the starting state in $\mathcal{N}$ attains the optimal APPC value $V^*_{\mathcal{N}}$.

*Proof.* We prove the following correspondence between $\mathcal{N}$ and $\mathcal{N}_{\mathsf{sur}}$. For every $u, u' \in S_{\mathcal{N}_{\mathsf{sur}}}$ and $\zeta \in \mathbf{Act}_{\mathsf{sur}}(u)$, it holds that $\zeta$ is a well-defined partial stationary strategy for $\mathcal{N}$. The transition probability $P_{\mathsf{sur}}(u, \zeta, u')$ is the probability that in $\mathcal{N}$, when applying $\zeta$ starting from $u$, the first surveillance cycle is completed by visiting $u'$, i.e.,

$$P_{\mathsf{sur}}(u, \zeta, u') = \Pr_u^{\mathcal{N},\zeta}(\mathbf{X}(\neg S_{\mathcal{N}_{\mathsf{sur}}} \mathbf{U}\, u')).$$

The penalty $\mathrm{pen}_{\mathsf{sur}}(u, \zeta)$ is the expected cumulative penalty gained in $\mathcal{N}$ when applying $\zeta$ starting from $u$ until the first surveillance cycle is completed. On the

other hand, for every partial stationary strategy $\zeta$ for $\mathcal{N}$ such that

$$\Pr_u^{\mathcal{N},\zeta}(\mathbf{F}\, S_{\mathcal{N}_{\mathsf{sur}}}) = 1$$

for some $u \in S_{\mathcal{N}\mathsf{sur}}$, there exists an action $\zeta' \in \mathbf{Act}_{\mathsf{sur}}(u)$ such that the action $\zeta'$ corresponds to the partial stationary strategy $\zeta$ in the above sense, i.e.,

$$P_{\mathsf{sur}}(u, \zeta', u') = \Pr_u^{\mathcal{N},\zeta}(\mathbf{X}(\neg S_{\mathcal{N}_{\mathsf{sur}}} \mathbf{U}\, u'))$$

for every $u' \in S_{\mathcal{N}_{\mathsf{sur}}}$, and the penalty $\mathrm{pen}_{\mathsf{sur}}(u, \zeta')$ is the expected cumulative penalty gained in $\mathcal{N}$ when we apply $\zeta$ starting from $u$ until we reach a state in $S_{\mathcal{N}_{\mathsf{sur}}}$.

To prove the first part of the correspondence above, we prove the following invariant of Algorithm 4. Let $X = (S_X, \mathbf{Act}_X, P_X, AP_{\mathcal{P}}, L_{\mathcal{P}})$ be the MDP and $\mathrm{pen}_X$ the penalty function from the algorithm after the initialization, before the first iteration of the while cycle. It is easy to see that all actions of $X$ are well-defined partial stationary strategies. For the transition probabilities, it holds that

$$P_X(u_{from}, \zeta, u_{to}) = \Pr_{u_{from}}^{\mathcal{N},\zeta}(\mathbf{X}(\neg S_X \mathbf{U}\, u_{to}))$$

for every $u_{from}, u_{to} \in S_X$ and $\zeta \in \mathbf{Act}_X(u_{from})$. The penalty $\mathrm{pen}_X(u_{from}, \zeta)$ is the expected cumulative penalty gained in $\mathcal{N}$ starting from $u_{from}$ when applying $\zeta$ until we reach a state in $S_X$. We show that these conditions also hold after every iteration of the while cycle.

Let $X$ and $\mathrm{pen}_X$ satisfy the conditions above and let $u \in S_X \backslash S_{\mathcal{N}\mathsf{sur}}$. By removing the state $u$ from $S_X$, we obtain a new version of the MDP $X'$ and the function $\mathrm{pen}_{X'}$. Note that $S_{X'} \cup \{u\} = S_X$. Let $u_{from} \in S_{X'}$ be a state of $X'$ and $\zeta_{new} \in \mathbf{Act}_{X'}(u_{from})$ be its action such that $\zeta_{new}$ has changed in the process of removing the state $u$. The action $\zeta_{new}$ is a well-defined partial stationary strategy because it must have been created as a union of an action $\zeta_{old}$ of $u_{from}$ and an action $\zeta$ of $u$, both from the previous version $X$, which do not conflict on any state from $S_X$.

For a state $u_{to} \in S_{X'}$, we prove that

$$P_{X'}(u_{from}, \zeta_{new}, u_{to}) = \Pr_{u_{from}}^{\mathcal{N},\zeta_{new}}(\mathbf{X}(\neg S_{X'} \mathbf{U}\, u_{to})).$$

Since $\zeta_{new} = \zeta_{old} \cup \zeta$, the probability in $\mathcal{N}$ when applying $\zeta_{new}$ starting from $u_{from}$ of reaching the state $u_{to}$ as the next state in $S_{X'}$ is the probability of reaching it as the next state in $S_X$ when using $\zeta_{old}$ from $u_{from}$, plus the probability of reaching $u$ as the next state in $S_X$ from $u_{from}$ using $\zeta_{old}$ and then eventually reaching the state $u_{to}$ from $u$ using $\zeta$. This means

$$\begin{aligned}
&\Pr_{u_{from}}^{\mathcal{N},\zeta_{new}}(\mathbf{X}(\neg S_{X'} \mathbf{U}\, u_{to})) = \\
&= \Pr_{u_{from}}^{\mathcal{N},\zeta_{old}}(\mathbf{X}(\neg S_X \mathbf{U}\, u_{to})) + \Pr_{u_{from}}^{\mathcal{N},\zeta_{old}}(\mathbf{X}(\neg S_X \mathbf{U}\, u)) \cdot \Pr_u^{\mathcal{N},\zeta}(\mathbf{F}\, u_{to}) \\
&= P_X(u_{from}, \zeta_{old}, u_{to}) + P_X(u_{from}, \zeta_{old}, u) \cdot \Big( \sum_{i=0}^{\infty} P_X(u, \zeta, u)^i \cdot P_X(u, \zeta, u_{to}) \Big) \\
&= P_X(u_{from}, \zeta_{old}, u_{to}) + P_X(u_{from}, \zeta_{old}, u) \cdot \frac{P_X(u, \zeta, u_{to})}{1 - P_X(u, \zeta, u)}
\end{aligned}$$

which is exactly as defined in Algorithm 4.

Similarly, we prove that $\mathrm{pen}_{X'}(u_{from}, \zeta_{new})$ is the expected cumulative penalty gained in $\mathcal{N}$ starting from $u_{from}$ when applying $\zeta_{new}$ until we reach a state in $S_{X'}$. As $\zeta_{new} = \zeta_{old} \cup \zeta$, it is the expected cumulative penalty of reaching a state in $S_X$ by using $\zeta_{old}$ plus, in the case we reach $u$, the expected cumulative penalty of eventually reaching a state in $S_{X'}$, i.e., other than $u$, using $\zeta$. To be specific, we have

$$\mathrm{pen}_X(u_{from}, \zeta_{old}) + P_X(u_{from}, \zeta_{old}, u) \cdot \Big( \sum_{i=0}^{\infty} P_X(u, \zeta, u)^i \cdot (1 - P_X(u, \zeta, u)) \cdot (i+1) \Big) \cdot \mathrm{pen}_X(u, \zeta)$$

$$=$$

$$\mathrm{pen}_X(u_{from}, \zeta_{old}) + P_X(u_{from}, \zeta_{old}, u) \cdot \frac{\mathrm{pen}_X(u, \zeta)}{1 - P_X(u, \zeta, u)},$$

just as defined in Algorithm 4. This completes the proof of the first part of the correspondence between $\mathcal{N}$ and $\mathcal{N}_{\mathsf{sur}}$.

The second part of the correspondence between $\mathcal{N}$ and $\mathcal{N}_{\mathsf{sur}}$ follows directly from the fact that, in the process of removing a state $u \in S_X \backslash S_{\mathcal{N}_{\mathsf{sur}}}$, we consider all combinations of actions of $u$ which eventually reach a state different from $u$, with all actions of all states $u_{from}$ having an action under which $u$ is reached with non-zero probability.

From the correspondence between $\mathcal{N}$ and $\mathcal{N}_{\mathsf{sur}}$ it follows that in $\mathcal{N}_{\mathsf{sur}}$, there exists a finite run between every two states. Therefore, the minimum APPC value that can be obtained in $\mathcal{N}_{\mathsf{sur}}$ from any of its states is the same and it is denoted by $V^*_{\mathcal{N}_{\mathsf{sur}}}$. Since every state of $\mathcal{N}_{\mathsf{sur}}$ is a surveillance state, the APPC problem for $\mathcal{N}_{\mathsf{sur}}$ is equivalent to solving the ACPS problem for $\mathcal{N}_{\mathsf{sur}}$. Using one of the algorithms in [Ber12], we obtain a stationary strategy $C^V_{\mathcal{N}_{\mathsf{sur}}}$ that attains the APPC value $V^*_{\mathcal{N}_{\mathsf{sur}}}$ regardless of the starting state. From the correspondence between $\mathcal{N}$ and $\mathcal{N}_{\mathsf{sur}}$ it also follows that $V^*_{\mathcal{N}_{\mathsf{sur}}} = V^*_{\mathcal{N}}$.

Now we construct the strategy $C^V_{\mathcal{N}}$ for $\mathcal{N}$ and show that it attains the minimum APPC value $V^*_{\mathcal{N}}$ regardless of the initial state. Intuitively, the strategy $C^V_{\mathcal{N}}$ is constructed to lead to a single EC of $\mathcal{N}$ that provides the minimum APPC value and that is the EC encoded by the strategy $C^V_{\mathcal{N}_{\mathsf{sur}}}$ for $\mathcal{N}_{\mathsf{sur}}$.

Let $S_{def} \subseteq S_{\mathcal{N}}$ be the set of all states $u \in S_{\mathcal{N}}$ for which there exists a surveillance state $u_{\mathsf{sur}} \in S_{\mathcal{N}_{\mathsf{sur}}}$ such that the partial strategy $C^V_{\mathcal{N}_{\mathsf{sur}}}(u_{\mathsf{sur}})$ for $\mathcal{N}$ is defined on the state $u$. We compute a partial strategy $\zeta_{\mathsf{init}}$ that leads from every state from $S_{\mathcal{N}} \backslash S_{def}$ to the set $S_{def}$ as follows. Let $\mathcal{N}'$ be an MDP that is created from $\mathcal{N}$ by adding a new state $t$ and a new action $\alpha_{def}$. From every state $u \in S_{def}$, we define a new transition under $\alpha_{def}$ to $t$ with probability 1 and penalty 0. Let $C_{SSP}$ be a stationary optimal strategy for the SSP problem for $\mathcal{N}'$ and $t$ as the terminal state. We define $\zeta_{\mathsf{init}}(u) = C_{SSP}(u)$ for every $u \in S_{\mathcal{N}} \backslash S_{def}$.

The strategy $C^V_{\mathcal{N}}$ is a then finite-memory strategy

$$C^V_{\mathcal{N}} = (M, \mathsf{next}, \mathsf{trans}, \mathsf{start}),$$

where $M = S_{\mathcal{N}_{\mathsf{sur}}} \cup \{\mathsf{init}\}$ is the set of modes, $\mathsf{trans}\colon M \times S_{\mathcal{N}} \to M$ is the transition function such that for every $m \in M, u \in S_{\mathcal{N}}$

$$\mathsf{trans}(m,u) = \begin{cases} m & \text{if } u \notin S_{\mathcal{N}\mathsf{sur}}, \\ u & \text{otherwise.} \end{cases}$$

The function $\mathsf{next}\colon M \times S_{\mathcal{N}} \to Act_{\mathcal{N}}$ that selects an action to be applied in $\mathcal{N}$ is for $m \in M, u \in S_{\mathcal{N}}$ defined as

$$\mathsf{next}(m,u) = \begin{cases} \big(C^V_{\mathcal{N}_{\mathsf{sur}}}(m)\big)(u) & \text{if } m \in S_{\mathcal{N}_{\mathsf{sur}}} \\ \zeta_{\mathsf{init}}(u) & \text{otherwise.} \end{cases}$$

Finally, $\mathsf{start}\colon S_{\mathcal{N}} \to S_{\mathcal{N}_{\mathsf{sur}}}$ selecting the starting mode for $u \in S_{\mathcal{N}}$ is defined as

$$\mathsf{start}(u) = \begin{cases} u & \text{if } u \in S_{\mathcal{N}_{\mathsf{sur}}}, \\ m & \text{where } \big(C^V_{\mathcal{N}_{\mathsf{sur}}}(m)\big)(u) \\ & \text{is defined,} \\ \mathsf{init} & \text{otherwise.} \end{cases}$$

The strategy attains the APPC value $V^*_{\mathcal{N}}$ since it only simulates the strategy $C^V_{\mathcal{N}_{\mathsf{sur}}}$ by unwrapping the corresponding partial strategies. □

The following property of the strategy $C^V_{\mathcal{N}}$ is crucial for the correctness of our approach to Problem 4.

**Proposition 4.** For every $(s,q) \in S_{\mathcal{N}}$, it holds that

$$\lim_{k\to\infty} \Pr^{\mathcal{N},C^V_{\mathcal{N}}}_{(s,q)}(\{\rho \mid \frac{\mathrm{pen}_{\mathcal{P}}(\rho^{\to\sharp k})}{k} \leq V^*_{\mathcal{N}}\}) = 1,$$

where $\mathrm{pen}_{\mathcal{P}}(\rho^{\to\sharp k})$ denotes the cumulative penalty gained in the first $k$ surveillance cycles of a run $\rho \in \mathrm{Run}^{\mathcal{N}}((s,q))$. Hence, for every $\epsilon > 0$, there exists $j(\epsilon) \in \mathbb{N}$ such that if the strategy $C^V_{\mathcal{N}}$ is applied from a state $(s,q) \in S_{\mathcal{N}}$ for at least $j(\epsilon)$ surveillance cycles, then the average expected cumulative penalty per surveillance cycle in these surveillance cycles is at most $V^*_{\mathcal{N}} + \epsilon$ with probability at least $1 - \epsilon$.

*Proof.* In [CH11] the authors prove that a strategy solving the ACPS problem for an MDP satisfies a property analogous to the one in the proposition. Especially, for the strategy $C^V_{\mathcal{N}_{\mathsf{sur}}}$ for the reduced MDP $\mathcal{N}_{\mathsf{sur}}$, it holds that for any state $(s,q) \in S_{\mathcal{N}_{\mathsf{sur}}}$

$$\lim_{k\to\infty} \Pr^{\mathcal{N}_{\mathsf{sur}},C^V_{\mathcal{N}_{\mathsf{sur}}}}_{(s,q)}(\{\rho \mid \frac{\mathrm{pen}_{\mathcal{N}_{\mathsf{sur}}}(\rho^{\to k})}{k} \leq V^*_{\mathcal{N}_{\mathsf{sur}}}\}) = 1,$$

where $\mathrm{pen}_{\mathcal{N}_{\mathsf{sur}}}(\rho^{\to k})$ denotes the cumulative penalty gained in the first $n$ stages of a run $\rho \in \mathrm{Run}^{\mathcal{N}_{\mathsf{sur}}}((s,q))$. The proposition then follows directly from the construction of the strategy $C^V_{\mathcal{N}}$ from the strategy $C^V_{\mathcal{N}_{\mathsf{sur}}}$. □

### Almost-sure acceptance in an MAEC

Here we design a strategy for an MAEC $\mathcal{N} \in \text{MAEC}(\mathcal{P})$ that guarantees almost-sure satisfaction of the acceptance condition $Acc$ of $\mathcal{A}_\phi$. Let $(E, F)$ be a pair in $Acc$ such that $\mathcal{N}$ is accepting with respect to $(E, F)$, i.e., $L_{\mathcal{P}}(S_{\mathcal{N}}) \cap E = \emptyset$ and $L_{\mathcal{P}}(S_{\mathcal{N}}) \cap F \neq \emptyset$. There exists a stationary strategy $C^{\phi}_{\mathcal{N}}$ for $\mathcal{N}$ under which a state with label in $F$ is reached with probability 1 regardless of the starting state, i.e.,

$$\Pr^{\mathcal{N}, C^{\phi}_{\mathcal{N}}}_{(s,q)}(\mathbf{F}\, F) = 1 \tag{3.22}$$

for every $(s, q) \in S_{\mathcal{N}}$. The existence of such a strategy follows from the fact that $\mathcal{N}$ is an EC [BK08]. Moreover, we construct $C^{\phi}_{\mathcal{N}}$ to minimize the expected cumulative penalty before reaching a state in $S_{\mathcal{N}} \cap S \times F$.

The strategy $C^{\phi}_{\mathcal{N}}$ is found as follows. Let $\mathcal{N}'$ be an MDP that is created from $\mathcal{N}$ by adding a new state $t$ and a new action $\alpha_F$. From every state $(s, q) \in S_{\mathcal{N}} \cap S \times F$, we define a new transition under $\alpha_F$ to $t$ with probability 1 and penalty 0. Let $C_{SSP}$ be a stationary optimal strategy for the SSP problem for $\mathcal{N}'$ and $t$ as the terminal state. For a state $(s, q) \in S_{\mathcal{N}}$, we define $C^{\phi}_{\mathcal{N}}((s, q)) = C_{SSP}((s, q))$ if the state $(s, q)$ does not have a label in $F$, otherwise $C^{\phi}_{\mathcal{N}}((s, q)) = \alpha$ for some $\alpha \in Act_{\mathcal{N}}((s, q))$.

**Proposition 5.** The strategy $C^{\phi}_{\mathcal{N}}$ for $\mathcal{N}$ resulting from the above algorithm almost-surely reaches the set $S_{\mathcal{N}} \cap S \times F$ and minimizes the expected cumulative penalty before reaching the set, regardless of the initial state.

*Proof.* It follows directly from the fact that $C_{SSP}$ optimally solves the SSP problem for the MDP $\mathcal{N}'$ and $t$. □

### Optimal strategy for the product

Finally, we are ready to construct the strategy $C_{\mathcal{P}}$ for the product $\mathcal{P}$ that projects to an optimal solution for $\mathcal{M}$.

First, starting from the initial state $s_{\mathcal{P}\mathsf{init}}$, $C_{\mathcal{P}}$ applies the strategy $C_0$ until a state of an MAEC in the set $\mathsf{maec}^*$ is reached. Let $\mathcal{N} \in \mathsf{maec}^*$ denote the MAEC and let $(E, F) \in Acc$ be a pair from the accepting condition of $\mathcal{A}_\phi$ such that $\mathcal{N}$ is accepting with respect to $(E, F)$.

Now, the strategy $C_{\mathcal{P}}$ starts to play the rounds. Each round consists of two phases. First, play the strategy $C^{\phi}_{\mathcal{N}}$ until a state with label in $F$ is reached. Let us denote $k_i$ the number of steps we play $C^{\phi}_{\mathcal{N}}$ in $i$-th round. The second phase applies the strategy $C^{V}_{\mathcal{N}}$ until the number of completed surveillance cycles in the second phase of the current round is $l_i$. The number $l_i$ is any natural number for which

$$l_i \geq \max\{j(\tfrac{1}{i}), i \cdot k_i \cdot \text{pen}_{\mathcal{P}max}\},$$

where $j(\frac{1}{i})$ is from Proposition 4 and $\text{pen}_{\mathcal{P}max}$ is the maximum value of the penalties $\text{pen}_{\mathcal{P}}$. After applying the strategy $C^V_{\mathcal{N}}$ for $l_i$ surveillance cycles, we proceed to the next round $i+1$.

**Theorem 4.** The strategy $C_{\mathcal{P}}$ almost-surely satisfies the accepting condition $Acc$ of $\mathcal{A}_\phi$ and at the same time, $C_{\mathcal{P}}$ minimizes the APPC value $V_{\mathcal{P},C_{\mathcal{P}}}(s_{\mathcal{P}\mathsf{init}})$ among all strategies for $\mathcal{P}$ almost-surely satisfying $Acc$.

*Proof.* From Proposition 2 it follows that when applying the strategy $C_0$ from the initial state $s_{\mathcal{P}\mathsf{init}}$, the set $\mathsf{maec}^*$ is reached with probability 1.

Assume that $\mathcal{P}$ enters MAEC $\mathcal{N} \in \mathsf{maec}^*$ that is accepting with respect to a pair $(E, F) \in Acc$. Let $i$ be the current round of $C_{\mathcal{P}}$ and $\epsilon_i = \frac{1}{i}$. According to Proposition 5, a state with a label in $F$ is almost-surely reached. In addition, using Proposition 4, the average expected cumulative penalty per surveillance cycle in the $i$-th round is at most

$$\begin{aligned}
\frac{k_i \cdot \text{pen}_{\mathcal{N}max} + l_i(V^*_{\mathcal{N}} + \epsilon_i)}{l_i} &= \frac{(k_i + l_i)V^*_{\mathcal{N}} + k_i \cdot \text{pen}_{\mathcal{N}max} - k_i V^*_{\mathcal{N}} + l_i\epsilon_i}{k_i + l_i} \\
&= V^*_{\mathcal{N}} + \epsilon_i + \frac{k_i \cdot \text{pen}_{\mathcal{N}max}}{l_i} \\
&\leq V^*_{\mathcal{N}} + \epsilon_i + \frac{1}{i} \qquad (l_i \geq i \cdot k_i \cdot \text{pen}_{\mathcal{N}max}) \\
&= V^*_{\mathcal{N}} + \frac{2}{i}
\end{aligned}$$

with probability at least $1 - \frac{1}{i}$. Therefore, in the limit, in the MAEC $\mathcal{N}$, we both satisfy the LTL specification and reach the optimal APPC value with probability 1. Together with the fact that $\mathsf{maec}^*$ and $C_0$ satisfy the condition in Equation 3.21, we have that $C_{\mathcal{P}}$ is an optimal strategy for $\mathcal{P}$. □

### Complexity

The size of a Rabin automaton for an LTL formula $\phi$ is in the worst case doubly exponential in the size of the set $AP$. However, studies such as [KB06] show that in practice, for many LTL formulas, automata are much smaller and manageable.

Once the product $\mathcal{P}$ is built, we compute the set MAEC($\mathcal{P}$) by running $|Acc|$-times an algorithm for MEC decomposition, which is polynomial in the size of $\mathcal{P}$. The size of the set MAEC($\mathcal{P}$) is in the worst case $|Acc| \cdot |S_{\mathcal{P}}|$. For each MAEC $\mathcal{N}$, we compute its reduction $\mathcal{N}_{\mathsf{sur}}$ using Algorithm 4 in time $\mathcal{O}(|S_{\mathcal{N}}| \cdot |Act_{\mathcal{N}}|^{\mathcal{O}(|S_{\mathcal{N}}|)})$. The optimal APPC value $V^*_{\mathcal{N}}$ and an optimal finite-memory strategy $C^V_{\mathcal{N}}$ are then found in time polynomial in the size of the reduced MDP.

The algorithm for finding the strategy $C_0$ and the optimal set $\mathsf{maec}^*$ are again polynomial in the size of $\mathcal{P}$. Similarly, computing a stationary strategy $C^\phi_{\mathcal{N}}$ for an MAEC $\mathcal{N} \in \mathsf{maec}^*$ is polynomial in the size of $\mathcal{N}$.

**Usability**

As was proved above, the presented solution to Problem 4 is correct and complete. However, the resulting optimal strategy $C_{\mathcal{P}}$ for $\mathcal{P}$, and hence the projected strategy $C$ for $\mathcal{M}$ as well, is not a finite-memory strategy in general. The reason is that in the second phase of every round $i$, the strategy $C^V_{\mathcal{N}}$ is applied for $l_i$ surveillance cycles and $l_i$ is generally growing with $i$.

This, however, does not prevent the solution to be effectively used. The following simple rule can be applied to avoid performing all $l_i \geq \max\{i{\cdot}k_i{\cdot}\mathrm{pen}_{\mathcal{P}max}, j(\frac{1}{i})\}$ surveillance cycles in every round $i$. When the computation is in the second phase of round $i$ and the product is in an MAEC $\mathcal{N} \in \mathsf{maec}^*$, after completion of every surveillance cycle, we can check whether the average cumulative penalty per surveillance cycle in round $i$ is at most $V^*_{\mathcal{N}} + \frac{2}{i}$. If yes, we can proceed to the next round $i+1$, otherwise continue with the second phase of round $i$. As the simulation results in Section 3.3.4 show, the use of this simple rule dramatically decreases the number of performed surveillance cycles in almost every round.

On the other hand, the complexity of the resulting strategy $C$ for $\mathcal{M}$ can be reduced from non-finite-memory to finite-memory in the following case. Assume that for every $\mathcal{N} \in \mathsf{maec}^*$, the optimal APPC strategy $C^V_{\mathcal{N}}$ leads to an EC that contains a state from $F$, where $\mathcal{N}$ is accepting with respect to the pair $(E, F) \in \mathit{Acc}$. In this case, the optimal strategy $C_{\mathcal{P}}$ can be defined as a finite-memory strategy that first applies the strategy $C_0$ to reach a state of an MAEC $\mathcal{N} \in \mathsf{maec}^*$, and from that point on, only applies the strategy $C^V_{\mathcal{N}}$.

### 3.3.4 Case Study

We implemented the solution presented in Section 3.3.3 for a persistent surveillance robotics example. In this section, we report on the simulation results.

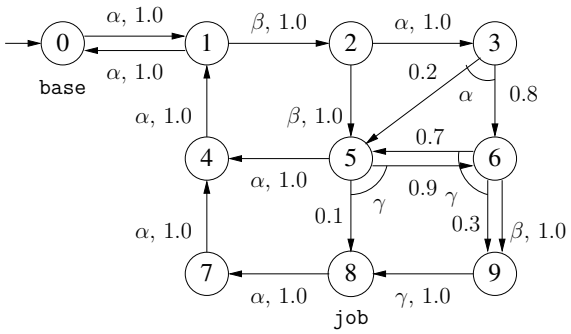
Consider a mobile robot moving in a partitioned environment. The motion of the robot is modeled by the initialized MDP $\mathcal{M}$ shown in Figure 3.10a. The set $AP$ of atomic propositions contains two propositions `base` and `job`. As depicted in Figure 3.10a, state 0 is the base location and state 8 is the job location. At the job location, the robot performs some work, and at the base, it reports on its job activity.

The robot's mission is to visit both base and job location infinitely many times. In addition, at least one job must be performed after every visit of the base, before the base is visited again. The corresponding LTL formula is

$$\phi = \mathbf{GF}\,\mathtt{base} \wedge \mathbf{GF}\,\mathtt{job} \wedge \mathbf{G}\big(\mathtt{base} \Rightarrow \mathbf{X}(\neg\mathtt{base}\,\mathbf{U}\,\mathtt{job})\big).$$

While satisfying the formula, we want to minimize the expected average penalty between two consecutive jobs, i.e., the surveillance proposition $a_{\mathsf{sur}} = \mathtt{job}$.

In the simulation, we use a Rabin automaton $\mathcal{A}$ for the formula that has 5 states and the accepting condition contains 1 pair. The product $\mathcal{P}$ of the MDP $\mathcal{M}$ and $\mathcal{A}$ has 50 states and one MAEC $\mathcal{N}$ of 19 states. The optimal set of MAECs




**(a)**

| | Condition | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $C_{\mathsf{init}}$ | | $\alpha$ | – | – | – | – | – | – | – | – | – |
| $C_{p1}$ | before `job` | $\alpha$ | $\beta$ | $\alpha$ | $\alpha$ | $\alpha$ | $\gamma$ | $\gamma$ | $\alpha$ | $\alpha$ | $\gamma$ |
| | after `job` | $\alpha$ | $\alpha$ | $\alpha$ | $\alpha$ | $\alpha$ | $\gamma$ | $\gamma$ | $\alpha$ | $\alpha$ | $\gamma$ |
| $C_{p2}$ | | $\alpha$ | $\beta$ | $\alpha$ | $\alpha$ | $\alpha$ | $\gamma$ | $\gamma$ | $\alpha$ | $\alpha$ | $\gamma$ |

**(b)**

**Figure 3.10:** (a) Initialized MDP $\mathcal{M}$ with initial state 0. The penalties of applying $\alpha, \beta, \gamma$ in any states are 5, 10, 1, respectively, e.g., $\mathrm{pen}(1, \alpha) = 5$. (b) Definitions of strategies $C_{\mathsf{init}}, C_{p1}, C_{p2}$ for $\mathcal{M}$, the projections of strategies $C_0, C^{\phi}_{\mathcal{N}}$, $C^{V}_{\mathcal{N}}$ for $\mathcal{P}$, respectively. The condition "before `job`" means that the corresponding prescription is used if the job location has not yet been visited since the last visit of the base. Similarly, the prescription with condition "after `job`" is used if the job location was visited at least once since the last visit of the base.

$\mathsf{maec}^* = \{\mathcal{N}\}$. The optimal APPC value $V^*_{\mathcal{N}} = 40.5$. In Figure 3.10b, we list the projections of strategies $C_0, C^{\phi}_{\mathcal{N}}$, $C^{V}_{\mathcal{N}}$ for $\mathcal{P}$ to strategies $C_{\mathsf{init}}, C_{p1}, C_{p2}$ for $\mathcal{M}$, respectively. The optimal strategy $C$ for $\mathcal{M}$ is then defined as follows. Starting from the initial state 0, apply strategy $C_{\mathsf{init}}$ until a state is reached, where $C_{\mathsf{init}}$ is no longer defined. Start round number 1. In $i$-th round, proceed as follows. In the first phase of the round, apply strategy $C_{p1}$ until the base is reached and then for one more step (the product $\mathcal{P}$ has to reach a state from the Rabin pair). Let $k_i$ denote the number of steps in the first phase of round $i$. In the second phase, use strategy $C_{p2}$ for $l_i = \max\{i \cdot k_i \cdot 10, j(\frac{1}{i})\}$ surveillance cycles, i.e., until the number of jobs performed by the robot is $l_i$. We also use the rule described in Section 3.3.3 to shorten the second phase, if possible.

Let us summarize the statistical results we obtained for 5 executions of the strategy $C$ for $\mathcal{M}$, each of 100 rounds. The number $k_i$ of steps in the first phase of a round $i > 1$ was always 5 because in such case, the first phase starts at

the job location and the strategy $C_{p1}$ needs to be applied for exactly 4 steps to reach the base. Therefore, in every round $i > 1$, the number $l_i$ is at least $50 \cdot i$, e.g., in round 100, $l_i \geq 5000$. However, using the rule described in Section 3.3.3, the average number of jobs per round was 130 and the median was only 14. In particular, the number was not increasing with the round. On the contrary, it appears to be independent from the history of the execution. In addition, at most 2 rounds in each of the executions finished only at the point, when the number of jobs performed by the robot in the second phase reached $l_i$. The average APPC value attained after 100 rounds was 40.56.

In contrast to our solution, the algorithm proposed in [DSBR11] does not find an optimal strategy for $\mathcal{M}$. Regardless of the initialization of the algorithm, it always results in a sub-optimal strategy, namely the strategy $C_{p1}$ from Figure 3.10b that has APPC value 50.5.

### 3.3.5 Conclusion

In this section, we focused on the problem of designing a control strategy for an MDP to guarantee satisfaction of an LTL formula with surveillance task, and at the same time, to minimize the expected average cumulative penalty between visits of surveillance states. This problem was previously addressed in [DSBR11], where the authors propose a sub-optimal solution based on dynamic programming. In contrast to this work, we exploit recent results from theoretical computer science, namely game theory and probabilistic model checking, to provide an optimal solution to this control problem.

# Chapter 4

# Control for Dynamic Systems

In this chapter, instead of focusing on control of finite discrete models, we consider the more general problem of LTL control for infinite dynamic systems. The complexity of dynamic systems vary depending on whether continuous- or discrete-time is considered and depending on the form of the corresponding differential or difference equation, respectively. As the use of formal methods might lead to increased computational complexity, in this chapter we focus on discrete-time systems with linear dynamics and stochastic uncertainty. While linear dynamics is suitable to describe many real systems, it is simple enough to analyze. Motivated by the hierarchical approach, we design an iterative abstraction-refinement algorithm to compute the set of all initial states from which the a given LTL formula can be satisfied with probability 1 and construct the corresponding strategies.

We start with a motivation for the considered problem and follow with the formal problem formulation. The proposed solution is presented next, including illustrative examples. Finally, the solution is demonstrated on a case study.

## 4.1 Linear Stochastic Systems

### 4.1.1 Motivation

In this work, we focus on the problem of finding the set of all initial states of a linear stochastic system from which a given constraint can be satisfied, and synthesizing the corresponding witness control strategies. In particular, we consider properties expressed as formulas of GR(1) fragment of LTL that offers polynomial computational complexity as compared to the doubly exponential one of general LTL, while being expressive enough to describe most of the usually considered temporal properties, see Section 2.2. We require the formula to be satisfied almost-surely, i.e., with probability 1. The almost-sure satisfaction is the strongest probability guarantee one can achieve while accounting for the stochasticity of the dynamics.

In our proposed approach, we iteratively construct and refine a discrete ab-

straction of the system and solve the synthesis problem for the abstract model. The discrete model considered in this work is a $2^1/_2$-player game introduced in Section 2.3.3. Every iteration of our algorithm produces a partial solution given as a partition of the state space into three categories. The first is a set of satisfying initial states together with corresponding witness strategies. The second is a set of non-satisfying initial states, i.e., those from which the system cannot be controlled to satisfy the specification with probability 1. Finally, some parts of the state space may remain undecided due to coarse abstraction. As the abstraction gets more precise, more states are being decided with every iteration of the algorithm. The designed solution is partially correct. That means, we guarantee soundness, i.e., almost sure satisfaction of the formula by all controlled trajectories starting in the satisfying initial set and non-existence of a satisfying control strategy for non-satisfying initial states. On the other hand, completeness is not ensured. If a weaker abstraction model, such as 2 player games, was used, there would be no soundness guarantee on the non-satisfying initial states and no completeness guarantees. We provide a practical implementation of the algorithm that ends after a predefined number of iterations.

The main novelty of our work is the abstraction-refinement of a dynamic system using a $2^1/_2$-player game. While abstraction-refinement exists for discrete systems such as non-deterministic and probabilistic systems [HJM03, CCD14, KKNP10, CHJM05], and some classes of hybrid systems [HNP$^+$11, NO14], to the best of our knowledge, the approach that we present in this paper is the first attempt to construct abstraction-refinement of stochastic systems with continuous state and control spaces in the form of $2^1/_2$-player games. The game theoretic solutions are necessary to determine what needs to be refined, and the dynamics of the linear-stochastic systems determine the refinement steps. Thus both game theoretic aspects and the dynamics of the system play a crucial role in the refinement step, see Remark 6. Also, the almost-sure analysis allows us to effectively build the game using only polytopic operators and thus avoiding the expensive and often approximative computation of integrals.

This paper is closely related to [YTv$^+$12, GLB12, LAB12, ADDB11, WTM12b]. Our computation of the abstraction is inspired from [YTv$^+$12], which, however, does not consider stochastic dynamics and does not perform refinement. The latter issue is addressed in [GLB12] for non-stochastic dynamics and specifications with finite-time semantics in the form of syntactically co-safe LTL formulas. The exact problem that we formulate in this paper was also considered in [LAB12], but for finite-time specifications in the form of probabilistic Computation Tree Logic (PCTL) formulas and for the particular case when the control space is finite. Also, in [LAB12], the abstraction is constructed in the form of an interval-valued MDP, which is less expressive than the game considered here. An uncontrolled version of the abstraction problem for a stochastic system was considered in [ADDB11], where the finite system was in form of a Markov set chain. In [WTM12b], the authors consider the problem of controlling uncertain MDPs from LTL specifi-

cation. When restricted to almost sure satisfaction, uncertain MDPs have the same expressivity as the games considered here. To obtain a control strategy, the authors of [WTM12b] use dynamic programming (value iteration), as opposed to games.

The results presented in this section are based on our results in [SKC+15]. The rest of the section is organized as follows. We formulate the problem and outline the approach in Section 4.1.2. The abstraction, game, and refinement algorithms are presented in Section 4.1.3. Finally, in Section 4.1.4, we present a case study and simulation results.

### 4.1.2 Problem Formulation

In this work, we assume the system is modeled as a linear stochastic system $\mathcal{L}$ described with an equation of the form

$$\mathcal{L}: x_{t+1} = Ax_t + Bu_t + w_t,$$

and a set of atomic propositions $AP$ is given as a finite set of linear predicates over its state space $\mathcal{X}$, see Definition 12.

**Problem 5.** ***Given***

- *a linear stochastic system $\mathcal{L}$,*
- *a finite set of atomic propositions $AP$,*
- *a GR(1) formula $\phi$ over alphabet $AP$,*

***find***

*(i) the set $\mathcal{X}_{init} \subseteq \mathcal{X}$ of all states $x \in \mathcal{X}$ such that there exists a control strategy $C_x \colon \mathrm{Run}^{\mathcal{L}}_{\mathrm{fin}}(x) \to \mathcal{U}$ that satisfies $\phi$ with probability 1,*

*(ii) the corresponding strategies $C_x$ for $x \in \mathcal{X}_{init}$.*

The solution we propose for Problem 5 can be summarized as follows. First, we abstract the linear stochastic system $\mathcal{L}$ using a $2^1/_2$-player game based on the partition of the state space $\mathcal{X}$ given by linear predicates $\Pi$. The game is built only using polytopic operations on the state space and control space. We analyze the game and identify those partition elements of the state space $\mathcal{X}$ that provably belong to the solution set $\mathcal{X}_{init}$, as well as those that do not contain any state from $\mathcal{X}_{init}$. The remaining parts of the state space still have the potential to contribute to the set $\mathcal{X}_{init}$ but are not decided yet due to coarse abstraction. In the next step, the partition of state space $\mathcal{X}$ is refined using deep analysis of the constructed game. Given the new partition, we build a new game and repeat the analysis. The approach can be graphically represented as shown in Figure 4.1.

We prove that the result of every iteration is a partial solution to Problem 5. In other words, the computed set of satisfying initial states as well as the set of non-satisfying initial states are correct. Moreover, they are improved or maintained with every iteration as the abstraction gets more precise. This allows us to efficiently use the proposed algorithm for a fixed number of iterations. Finally,

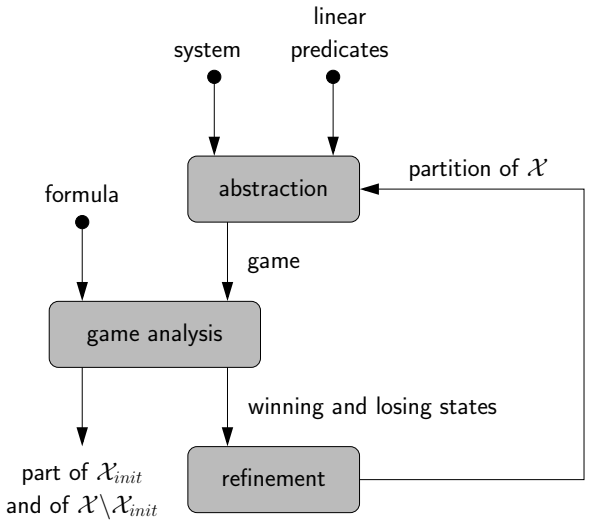



**Figure 4.1:** Graphical representation of the proposed solution to Problem 4.1.2.

we prove that if the algorithm terminates then the result is indeed the solution to Problem 5.

The main difficulty of the approach is the abstraction-refinement of $2^1/_2$-player game. Abstraction-refinement has been considered for discrete systems [KKNP10, HJM03, CCD14, CHJM05], and also for some classes of hybrid systems [HNP$^+$11, NO14]. However, in all of these approaches, even if the original system is considered to be probabilistic, the distributions are assumed to be discrete and given, and are not abstracted away during the refinement. The key challenge is the extension of abstraction-refinement approach to continuous stochastic systems, where the transition probabilities in the abstract discrete model need to be abstracted. We show that by exploiting the nature of the considered dynamic systems we can develop an abstraction-refinement approach for our problem, see Remark 6.

### 4.1.3 Problem Solution

In this section, we describe the proposed solution in detail and present necessary proofs. We start with the abstraction procedure that consists of two steps. The linear stochastic system $\mathcal{L}$ is first abstracted using a non-deterministic transition system which is then extended to a $2^1/_2$-player game. The game analysis section then describes how to identify parts of the solution to Problem 4.1.2. The procedure for refinement is presented last.

Recall the set $\mathcal{X}_{\mathsf{out}}$ defined in Equation 2.5 that contains all states outside of the state space $\mathcal{X}$ that can be reached within one step in system $\mathcal{L}$. Note that $\mathcal{X}_{\mathsf{out}}$ is generally not a polytope, but it can be represented as a finite set of polytopes $\{\mathcal{X}_{i_{\mathsf{out}}}\}_{i_{\mathsf{out}} \in I_{\mathsf{out}}}$, or $\{\mathcal{X}_{i_{\mathsf{out}}}\}$ for short. In the following sections, we use

**Table 4.1:** Definitions of polytopic operators Post (posterior), Pre (predecessor), PreR (robust predecessor), PreP (precise predecessor), Attr (attractor) and AttrR (robust attractor), where $\mathcal{X}' \subseteq \mathcal{X}, \mathcal{U}' \subseteq \mathcal{U}$ are polytopes, and $\{\mathcal{X}_j\}_{j\in J}$ is a set of polytopes in $\mathcal{X}$. The algorithms to compute all the operators are listed in Appendix C.

$$
\begin{aligned}
\mathrm{Post}(\mathcal{X}', \mathcal{U}') &= \{x \in \mathbb{R}^N \mid \exists x' \in \mathcal{X}', \exists u \in \mathcal{U}', \exists w \in \mathcal{W} : x = Ax' + Bu + w\} \\
\mathrm{Pre}(\mathcal{X}', \mathcal{U}', \{\mathcal{X}_j\}_{j\in J}) &= \{x \in \mathcal{X}' \mid \exists u \in \mathcal{U}' : \mathrm{Post}(x,u) \cap \bigcup_{j\in J} \mathcal{X}_j \text{ is non-empty}\} \\
\mathrm{PreR}(\mathcal{X}', \mathcal{U}', \{\mathcal{X}_j\}_{j\in J}) &= \{x \in \mathcal{X}' \mid \exists u \in \mathcal{U}' : \mathrm{Post}(x,u) \subseteq \bigcup_{j\in J} \mathcal{X}_j\} \\
\mathrm{PreP}(\mathcal{X}', \mathcal{U}', \{\mathcal{X}_j\}_{j\in J}) &= \{x \in \mathcal{X}' \mid \exists u \in \mathcal{U}' : \mathrm{Post}(x,u) \subseteq \bigcup_{j\in J} \mathcal{X}_j \text{ and} \\
&\qquad \forall j \in J : \mathrm{Post}(x,u) \cap \mathcal{X}_j \text{ is non-empty}\} \\
\mathrm{Attr}(\mathcal{X}', \mathcal{U}', \{\mathcal{X}_j\}_{j\in J}) &= \{x \in \mathcal{X}' \mid \forall u \in \mathcal{U}' : \mathrm{Post}(x,u) \cap \bigcup_{j\in J} \mathcal{X}_j \text{ is non-empty}\} \\
\mathrm{AttrR}(\mathcal{X}', \mathcal{U}', \{\mathcal{X}_j\}_{j\in J}) &= \{x \in \mathcal{X}' \mid \forall u \in \mathcal{U}' : \mathrm{Post}(x,u) \subseteq \bigcup_{j\in J} \mathcal{X}_j\}
\end{aligned}
$$

various polytopic operators to analyze the dynamic system. All these operators are formally defined in Table 4.1 and their computation is described in detail in Appendix C.

### Abstraction

The abstraction consists of two steps. First, the linear stochastic system is abstracted using a non-deterministic transition system which is then extended to a $2^1/_2$-player game.

**Definition 22** (NTS)**.** *A non-deterministic transition system (NTS) is a tuple $\mathcal{N} = (S, Act, T, AP, L)$, where $S$ is a non-empty finite set of states, $Act$ is a non-empty finite set of actions, and $T\colon S \times Act \to 2^S$ is a non-deterministic transition function, $AP$ is a non-empty finite set of atomic propositions and $L\colon S \to 2^{AP}$ is a labeling function.*

In order to build an NTS abstraction for $\mathcal{L}$, we assume we are given a partition $\{\mathcal{X}_i\}_{i\in I}$, or $\{\mathcal{X}_i\}$ for short, of the state space $\mathcal{X}$. Initially, the partition is given by the set of linear predicates $AP$, i.e., it is the partition given by the equivalence relation $\sim_{AP}$ defined as

$$x \sim_{AP} x' \iff \forall (a\colon cx \le d) \in AP\colon \big(cx \le d \Leftrightarrow cx' \le d\big).$$

In the later iterations of the algorithm, the partition is given by the refinement procedure. The construction below builds on the approach from [YTv+12].

We use $\mathcal{N}_{\{\mathcal{X}_i\}} = (S_\mathcal{N}, Act_\mathcal{N}, T_\mathcal{N}, AP, L_\mathcal{N})$ to denote the NTS corresponding to partition $\{\mathcal{X}_i\}$ defined as follows. The states of $\mathcal{N}_{\{\mathcal{X}_i\}}$ are given by the partition

of the state space $\mathcal{X}$ and the outer part $\mathcal{X}_{\mathsf{out}}$, i.e., $S_{\mathcal{N}} = \{\mathcal{X}_i\} \cup \{\mathcal{X}_{i_{\mathsf{out}}}\}$. Let $\mathcal{X}_i \in \{\mathcal{X}_i\} \subset S_{\mathcal{N}}$ be a state of the NTS, a polytope in $\mathcal{X}$. We use $\sim_i$ to denote the equivalence relation on $\mathcal{U}$ such that $u \sim_i u'$ if for every state $\mathcal{X}_j \in \{\mathcal{X}_i\} \cup \{\mathcal{X}_{i_{\mathsf{out}}}\}$, it holds that $\mathrm{Post}(\mathcal{X}_i, u) \cap \mathcal{X}_j$ is non-empty if and only if $\mathrm{Post}(\mathcal{X}_i, u') \cap \mathcal{X}_j$ is non-empty. Intuitively, two control inputs are equivalent with respect to $\mathcal{X}_i$, if from $\mathcal{X}_i$ the system $\mathcal{L}$ can transit to the same set of partition elements of $\mathcal{X}$ and $\mathcal{X}_{\mathsf{out}}$. The partition $\mathcal{U}/\sim_i$ is then the set of all actions of the NTS $\mathcal{N}_{\{\mathcal{X}_i\}}$ that are allowed in state $\mathcal{X}_i$. We use $\mathcal{U}_i^J$ to denote the union of those partition elements from $\mathcal{U}/\sim_i$ that contain control inputs that lead the system from $\mathcal{X}_i$ to polytopes $\mathcal{X}_j, j \in J \subseteq I \cup I_{\mathsf{out}}$, i.e.,

$$\begin{aligned}\mathcal{U}_i^J = \{u \in \mathcal{U} \mid &\forall j \in J : \mathrm{Post}(\mathcal{X}_i, u) \cap \mathcal{X}_j \text{ is non-empty and} \\ &\forall j \notin J : \mathrm{Post}(\mathcal{X}_i, u) \cap \mathcal{X}_j \text{ is empty}\}. \end{aligned} \tag{4.1}$$

The set $\mathcal{U}_i^J$ can be computed using only polytopic computations as described in Appendix C.1. For a state $\mathcal{X}_i \in \{\mathcal{X}_i\} \subset S_{\mathcal{N}}$ and action $\mathcal{U}_{i'}^J \in Act_{\mathcal{N}}$, we let

$$T_{\mathcal{N}}(\mathcal{X}_i, \mathcal{U}_{i'}^J) = \begin{cases} \{\mathcal{X}_j \mid j \in J\} & \text{if } i = i', \\ \emptyset & \text{otherwise.} \end{cases}$$

For states $\mathcal{X}_{i_{\mathsf{out}}} \in \{\mathcal{X}_{i_{\mathsf{out}}}\} \subset S_{\mathcal{N}}$, no actions or transitions are defined. Finally, the labeling function $L_{\mathcal{N}}$ for the NTS $\mathcal{N}_{\{\mathcal{X}_i\}}$ is defined as

$$L_{\mathcal{N}}(\mathcal{X}_i) = \{(a\colon cx \leq d) \in AP \mid \forall x \in \mathcal{X}_i\colon cx \leq d\}. \tag{4.2}$$

Since the NTS does not capture the probabilistic aspect of the linear stochastic system, we build a $2\frac{1}{2}$-player game on top of the NTS. Let $\mathcal{X}_i$ be a polytope within the state space $\mathcal{X}$ of $\mathcal{L}$, a state of $\mathcal{N}_{\{\mathcal{X}_i\}}$. When $\mathcal{L}$ is in a particular state $x \in \mathcal{X}_i$ and a control input $u \in \mathcal{U}_i^J$ is to be applied, we can compute the probability distribution over the set $\{\mathcal{X}_j\}_{j \in J}$ that determines the probability of the next state of $\mathcal{L}$ being in $\mathcal{X}_j, j \in J$, using the distribution of the random vector for uncertainty. The evolution of the system can thus be seen as a game, where Player 1 acts in states $\mathcal{X}_i \in S_{\mathcal{N}}$ of the NTS and chooses actions from $Act_{\mathcal{N}}$, and Player 2 determines the exact state within the polytope $\mathcal{X}_i$ and thus chooses the probability distribution according to which a transition in $\mathcal{L}$ is made. This intuitive game construction implies that Player 2 has a possibly infinite number of actions. On the other hand, in Problem 5 we are interested in satisfying the GR(1) specification with probability 1 and in the theory of finite discrete probabilistic models, it is a well-studied phenomenon that in almost-sure analysis, the exact probabilities in admissible probability distributions of the model are not relevant. It is only important to know supports of such distributions, see e.g., [BK08]. That means that in our case we do not need to consider all possible probability distributions as actions for Player 2, but it is enough to consider that Player 2 chooses support for the probability distribution that will be used to make a

transition. For a polytope $\mathcal{X}_i \in S_\mathcal{N}$ and $\mathcal{U}_i^J \in Act_\mathcal{N}$, we use $\mathrm{Supp}(\mathcal{X}_i, \mathcal{U}_i^J)$ to denote the set of all subsets $J' \subseteq J$ for which there exist $x \in \mathcal{X}_i, u \in \mathcal{U}_i^J$ such that the next state $x' = Ax + Bu + w$ of $\mathcal{L}$ belongs to $\mathcal{X}_j, j \in J'$ with non-zero probability and with zero probability to $\mathcal{X}_j, j \notin J'$, i.e.,

$$\mathrm{Supp}(\mathcal{X}_i, \mathcal{U}_i^J) = \{J' \subseteq J \mid \mathrm{PreP}(\mathcal{X}_i, \mathcal{U}_i^J, \{\mathcal{X}_j\}_{j \in J'}) \text{ is non-empty}\}, \tag{4.3}$$

where PreP is the precise predecessor operator from Table 4.1.

Given the NTS $\mathcal{N}_{\{\mathcal{X}_i\}}$, the $2\frac{1}{2}$-player game $\mathcal{G}_{\{\mathcal{X}_i\}} = (S_1 \cup S_2, Act, P, AP, L)$ is defined as follows. Player 1 states $S_1 = \{\mathcal{X}_i\} \cup \{\mathcal{X}_{i_{\mathsf{out}}}\}$ are the states $S_\mathcal{N}$ of the NTS and Player 1 actions are the actions $Act_\mathcal{N}$ of $\mathcal{N}_{\{\mathcal{X}_i\}}$. Player 2 states are given by the choice of an action in a Player 1 state, i.e., $S_2 = \{\mathcal{X}_i\} \times \{\mathcal{U}_i^J\}$. The Player 2 actions available in a state $(\mathcal{X}_i, \mathcal{U}_i^J)$ are the elements of the set $\mathrm{Supp}(\mathcal{X}_i, \mathcal{U}_i^J)$ defined in Equation 4.3. For Player 1, the transition probability function $P$ defines non-zero probability transitions only for triples of the form $\mathcal{X}_i, \mathcal{U}_i^J, (\mathcal{X}_i, \mathcal{U}_i^J)$ and for such it holds $P(\mathcal{X}_i, \mathcal{U}_i^J)((\mathcal{X}_i, \mathcal{U}_i^J)) = 1$. For Player 2, the function $\delta$ defines the following transitions:

$$P\big((\mathcal{X}_i, \mathcal{U}_i^J), J'\big)\big(\mathcal{X}_j\big) = \begin{cases} \frac{1}{|J'|} & \text{if } J' \in \mathrm{Supp}(\mathcal{X}_i, \mathcal{U}_i^J) \\ & \text{and } j \in J', \\ 0 & \text{otherwise.} \end{cases}$$

The definition reflects the fact that once Player 2 chooses the support, the exact transition probabilities are irrelevant and without loss of generality, we can consider them to be uniform. For a Player 1 state $\mathcal{X}_i$, the labeling function $L(\mathcal{X}_i) = L_\mathcal{N}(\mathcal{X}_i)$ is defined in the same way as in Equation 4.2. For Player 2 states, the labeling function always returns an empty set.

**Example 6.** Let $\mathcal{L}$ be a linear stochastic system of the form given in Definition 12, where

$$A = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, B = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix},$$

the state space is $\mathcal{X} = \{x \in \mathbb{R}^2 \mid 0 \leq x(1) \leq 4, 0 \leq x(2) \leq 2\}$, the control space is $\mathcal{U} = \{u \in \mathbb{R}^2 \mid -1 \leq u(1), u(2) \leq 1\}$, and the random vector takes values in polytope $\mathcal{W} = \{w \in \mathbb{R}^2 \mid -0.1 \leq w(1), w(2) \leq 0.1\}$. Let $AP$ contain a single linear predicate $a\colon\ x(1) \leq 2$. In Figure 4.2a, polytopes $\mathcal{X}_1$ and $\mathcal{X}_2$ form the partition of $\mathcal{X}$ given by $AP$, and polytopes $\mathcal{X}_3, \mathcal{X}_4, \mathcal{X}_5, \mathcal{X}_6$ form the rest of the one step reachable set of system $\mathcal{L}$, i.e., $\mathcal{X}_{\mathsf{out}}$. The game $\mathcal{G}_{\{\mathcal{X}_i\}}$ given by this partition has 6 states and 18 actions. In Figure 4.2b, we visualize part of the transition function as follows. In Player 1 state $\mathcal{X}_1$, if Player 1 chooses, e.g., action $\mathcal{U}_1^{\{1,2,5\}}$ that leads from $\mathcal{X}_1$ to polytopes $\mathcal{X}_1, \mathcal{X}_2, \mathcal{X}_5$, the game is in Player 2 state $(\mathcal{X}_1, \mathcal{U}_1^{\{1,2,5\}})$ with probability 1. Actions of Player 2 are the available supports of the action over

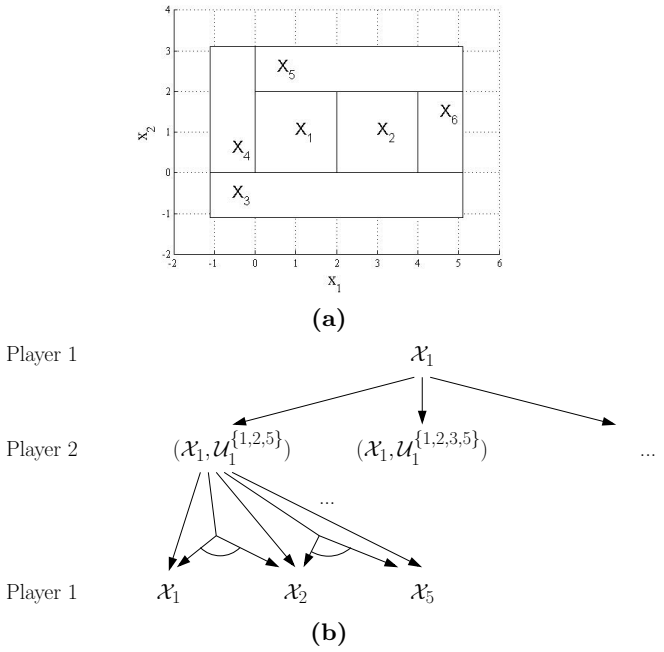



**Figure 4.2:** (a) Partition of state space $\mathcal{X}$ of system $\mathcal{L}$ in Example 6 given by linear predicates in $AP$. Polytopes $\mathcal{X}_3, \ldots, \mathcal{X}_6$ form the set $\mathcal{X}_{\mathsf{out}}$. (b) Part of the transition function of the game $\mathcal{G}_{\{\mathcal{X}_i\}}$ constructed in Example 6.

the set $\{\mathcal{X}_1, \mathcal{X}_2, \mathcal{X}_5\}$, which are in this case all non-empty subsets. If Player 2 chooses, e.g., support $\{\mathcal{X}_1, \mathcal{X}_2\}$, the game is in Player 1 state $\mathcal{X}_1$ or $\mathcal{X}_2$ with equal probability 0.5.

The following proposition proves that the game $\mathcal{G}_{\{\mathcal{X}_i\}}$ simulates the linear stochastic system $\mathcal{L}$.

**Proposition 6.** Let $\rho$ be a run of the linear stochastic system $\mathcal{L}$. Then there exists a play $\rho'$ of the game $\mathcal{G}_{\{\mathcal{X}_i\}}$ such that $\rho(n) \in \rho'(2n-1)$ for every $n \geq 1$.

*Proof.* The play $\rho'$ is defined as follows. The states $\rho'(2n-1) = \mathcal{X}_i$ such that $\rho(n) \in \mathcal{X}_i$. The states $\rho'(2n) = (\mathcal{X}_i, \mathcal{U}_i^J)$ such that there exist $u \in \mathcal{U}_i^J, w \in \mathcal{W}$ for which $\rho(n+1) = A\rho(n) + Bu + w$. □

On the other hand, since $\mathcal{G}_{\{\mathcal{X}_i\}}$ is only an abstraction of the system $\mathcal{L}$, it may contain plays that do not correspond to any trace of the system.

### Game analysis

Let $\mathcal{G}_{\{\mathcal{X}_i\}}$ be the $2^1/_2$-player game constructed for the linear stochastic system $\mathcal{L}$ and partition $\{\mathcal{X}_i\}$ of its state space using the procedure from above. In this

section, we identify partition elements from $\{\mathcal{X}_i\}$ which are part of the solution set of initial states $\mathcal{X}_{init}$ as well as those that do not contain any satisfying initial states from $\mathcal{X}$.

First, we compute the almost-sure winning set $S_{\mathsf{yes}}$ in game $\mathcal{G}_{\{\mathcal{X}_i\}}$ with respect to the GR(1) formula $\phi$ from Problem 5, i.e.,

$$S_{\mathsf{yes}} = \mathsf{Almost}^{\mathcal{G}_{\{\mathcal{X}_i\}}}(\phi). \tag{4.4}$$

We proceed as follows. Let $\mathcal{A} = (Q, 2^{AP}, \delta^{\mathcal{A}}, q_0, (E, F))$ be a deterministic $\omega$-automaton with Büchi implication acceptance condition for the GR(1) formula $\phi$ constructed as described in Section 2.2.3. We consider the $2\frac{1}{2}$-player game

$$\mathcal{P}_{\{\mathcal{X}_i\}} = (S_1^{\mathcal{P}} \cup S_2^{\mathcal{P}}, Act, P^{\mathcal{P}}, AP, L^{\mathcal{P}})$$

that is the synchronous product of $\mathcal{G}_{\{\mathcal{X}_i\}}$ and $\mathcal{A}$, where $S_1^{\mathcal{P}} = S_1 \times Q$, $S_2^{\mathcal{P}} = S_2 \times \mathcal{A}$, and for every $(\mathcal{X}_i, q) \in S_1^{\mathcal{P}}$ and $\mathcal{U}_i^J \in Act$ we have

$$P^{\mathcal{P}}\big((\mathcal{X}_i, q), \mathcal{U}_i^J\big)\big((\mathcal{X}_i, \mathcal{U}_i^J), q'\big)\big) = \begin{cases} P\big(\mathcal{X}_i, \mathcal{U}_i^J\big)\big((\mathcal{X}_i, \mathcal{U}_i^J)\big) \\ \quad \text{if } \delta^{\mathcal{A}}(q, L(\mathcal{X}_i)) = q', \\ 0 \\ \quad \text{otherwise,} \end{cases}$$

and similarly, for all $((\mathcal{X}_i, \mathcal{U}_i^J), q) \in S_2^{\mathcal{P}}$ and $J' \in Act$ we have

$$P^{\mathcal{P}}\big(((\mathcal{X}_i, \mathcal{U}_i^J), q), J'\big)\big((\mathcal{X}_j, q')\big) = \begin{cases} P\big((\mathcal{X}_i, \mathcal{U}_i^J), J'\big)(\mathcal{X}_j) \\ \quad \text{if } q = q', \\ 0 \\ \quad \text{otherwise.} \end{cases}$$

When constructing the product game, we only consider those states from $S_1 \times Q$ and $S_2 \times Q$ that are reachable from some $(\mathcal{X}_i, q_0)$, where $q_0$ is the initial state of the automaton $\mathcal{A}$. Finally, we consider Büchi implication acceptance condition $(E^{\mathcal{P}}, F^{\mathcal{P}})$, where $E^{\mathcal{P}} = (S_1^{\mathcal{P}} \cup S_2^{\mathcal{P}}) \times E$ and $F^{\mathcal{P}} = (S_1^{\mathcal{P}} \cup S_2^{\mathcal{P}}) \times F$.

**Proposition 7.** The set $S_{\mathsf{yes}}$ defined in Equation 4.4 consists of all $\mathcal{X}_i \in S_1$ for which $(\mathcal{X}_i, q_0) \in S_{\mathsf{yes}}^{\mathcal{P}}$, where the set

$$S_{\mathsf{yes}}^{\mathcal{P}} = \mathsf{Almost}^{\mathcal{P}_{\{\mathcal{X}_i\}}}((E^{\mathcal{P}}, F^{\mathcal{P}})) \tag{4.5}$$

can be computed using algorithm in Appendix B.

*Proof.* Follows directly from the construction of the game $\mathcal{P}_{\{\mathcal{X}_i\}}$ above and the results of [CdAH11]. □

The next proposition proves that the polytopes from $S_{\mathsf{yes}}$ are part of the solution to Problem 5.

**Proposition 8.** For every $\mathcal{X}_i \in S_{\mathsf{yes}}$, there exists a finite-memory strategy $C_{\mathcal{X}_i}$ for $\mathcal{L}$ such that every run of $\mathcal{L}$ under strategy $C_{\mathcal{X}_i}$ that starts in any $x \in \mathcal{X}_i$ satisfies the GR(1) formula $\phi$ with probability 1.

*Proof.* Let $\mathcal{X}_i \in S_{\mathsf{yes}}$ and let $C_{\mathcal{G}_{\{\mathcal{X}_i\}}}$ be a finite-memory almost-sure winning strategy for Player 1 from state $\mathcal{X}_i$ in game $\mathcal{G}_{\{\mathcal{X}_i\}}$, see Section 2.3.3. Let $C_{\mathcal{X}_i}$ be a strategy for $\mathcal{L}$ defined as follows. For a finite run $\sigma \in \mathrm{Run}^{\mathcal{L}}_{\mathrm{fin}}(\mathcal{X}_i)$, let $C_{\mathcal{X}_i}(\sigma) = u$, where $u \in C_{\mathcal{G}_{\{\mathcal{X}_i\}}}(\sigma_{\mathcal{G}_{\{\mathcal{X}_i\}}})$, where $\sigma_{\mathcal{G}_{\{\mathcal{X}_i\}}}$ is finite play of the game such that $\sigma(n) \in \sigma_{\mathcal{G}_{\{\mathcal{X}_i\}}}(2n)$ for every $1 \leq n \leq |\sigma|$. Since $C_{\mathcal{G}_{\{\mathcal{X}_i\}}}$ for game $\mathcal{G}_{\{\mathcal{X}_i\}}$ is almost-sure winning from state $\mathcal{X}_i$ with respect to $\phi$, i.e., every play of the game that starts in $\mathcal{X}_i$ almost-surely satisfies $\phi$, the analogous property holds for $C_{\mathcal{X}_i}$ and runs in $\mathcal{L}$. □

Next, we consider the set $S_{\mathsf{no}}$ of Player 1 states in game $\mathcal{G}_{\{\mathcal{X}_i\}}$ defined as follows:

$$S_{\mathsf{no}} = S_1 \setminus \mathsf{Almost}^{\mathcal{G}^{\mathsf{coop}}_{\{\mathcal{X}_i\}}}(\phi). \tag{4.6}$$

Intuitively, $S_{\mathsf{no}}$ is the set of states, where even if Player 2 cooperates with Player 1, $\phi$ can still not be satisfied with probability 1.

**Proposition 9.** The set $S_{\mathsf{no}}$ defined in Equation 4.6 consists of all $\mathcal{X}_i \in S_1$ for which $(\mathcal{X}_i, q_0) \in S^{\mathcal{P}}_{\mathsf{no}}$, where

$$S^{\mathcal{P}}_{\mathsf{no}} = S^{\mathcal{P}}_1 \setminus \mathsf{Almost}^{\mathcal{P}^{\mathsf{coop}}_{\{\mathcal{X}_i\}}}((E^{\mathcal{P}}, F^{\mathcal{P}})). \tag{4.7}$$

*Proof.* Follows directly from the construction of the product game $\mathcal{P}_{\{\mathcal{X}_i\}}$. □

We prove that no state $x \in \mathcal{X}_i$ for $\mathcal{X}_i \in S_{\mathsf{no}}$ is part of the solution to Problem 5.

**Proposition 10.** For every $\mathcal{X}_i \in S_{no}$ and $x \in \mathcal{X}_i$, there does not exist a strategy $C_x$ for $\mathcal{L}$ such that every run of $\mathcal{L}$ under $C_x$ starting in $x$ satisfies $\phi$ with probability 1.

*Proof.* Intuitively, from the construction of the game $\mathcal{G}_{\{\mathcal{X}_i\}}$, Player 2 represents the unknown precise state of the system $\mathcal{L}$ within in the abstraction, i.e., he makes the choice of a state inside each polytope $\mathcal{X}_i$ at each step. Therefore, if $\phi$ cannot be almost-surely satisfied from $\mathcal{X}_i$ in the game even if the two players cooperate, in $\mathcal{L}$ it translates to the fact that $\phi$ cannot be almost-surely satisfied from any $x \in \mathcal{X}_i$ even if we consider strategies that can moreover change inside each $\mathcal{X}_i$ arbitrarily at any moment. □

Finally, consider the set

$$S_? = S_1 \setminus (S_{\mathsf{yes}} \cup S_{\mathsf{no}}). \tag{4.8}$$

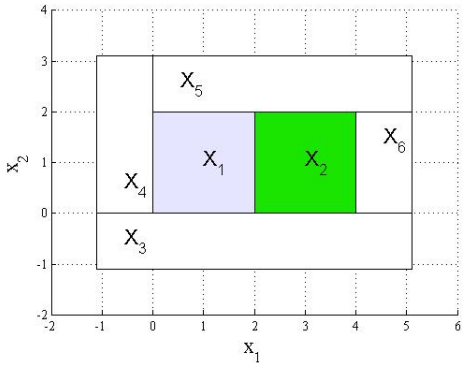



**Figure 4.3:** Solution of the game in Example 7. The polytopes, i.e., Player 1 states, that belong to sets $S_{\mathsf{yes}}, S_{\mathsf{no}}, S_?$ are shown in green, white and light blue, respectively.

These are the polytopes within the state space of $\mathcal{L}$ that have not been decided as satisfying or non-satisfying due to coarse abstraction. Alternatively, from Propositions 7 and 9, and Equation 4.8, we can define the set $S_?$ as the set of all $\mathcal{X}_i \in S_1$, for which $(\mathcal{X}_i, q_0) \in S_?^{\mathcal{P}}$, where

$$S_?^{\mathcal{P}} = S_1^{\mathcal{P}} \backslash (S_{\mathsf{yes}}^{\mathcal{P}} \cup S_{\mathsf{no}}^{\mathcal{P}}). \tag{4.9}$$

**Proposition 11.** For every $\mathcal{X}_i \in S_?$ it holds that the product game $\mathcal{P}_{\{\mathcal{X}_i\}}$ can be won cooperatively starting from the Player 1 state $(\mathcal{X}_i, q_0)$. Analogously, for every $(\mathcal{X}_i, q) \in S_?^{\mathcal{P}}$ it holds that the product game $\mathcal{P}_{\{\mathcal{X}_i\}}$ can be won cooperatively starting from $(\mathcal{X}_i, q)$.

*Proof.* The proposition follows directly from Equations 4.8 and 4.9, and Propositions 7 and 9. □

**Example 7.** Recall the linear stochastic system $\mathcal{L}$ from Example 6 and consider GR(1) formula $\mathbf{F}\neg a$ over the set $AP$ that requires to eventually reach a state $x \in \mathcal{X}$ such that $x(1) \geq 2$. The deterministic $\omega$-automaton for the formula has only two states, $q_0$ and $q_1$. The automaton remains in the initial state $q_0$ until polytope $\mathcal{X}_2$ is visited in $\mathcal{L}$. Then it transits to state $q_1$ and remains there forever. The Büchi implication condition $(E, F)$ is $E = \{q_0\}, F = \{q_1\}$. The solution of the game $\mathcal{G}_{\{\mathcal{X}_i\}}$ constructed in Example 6 with respect to the above formula is depicted in Figure 4.3.

If the set $S_{\mathsf{no}}$ contains all Player 1 states of the game $\mathcal{G}_{\{\mathcal{X}_i\}}$, the GR(1) formula $\phi$ cannot be satisfied in the system $\mathcal{L}$ and our algorithm terminates. If set $S_?$ is empty, the algorithm terminates and returns the union of all polytopes from $S_{\mathsf{yes}}$ as the solution to Problem 5. The corresponding satisfying strategies are synthesized

as described in the proof of Proposition 8. Otherwise, we continue the algorithm by computing a refined partition of the state space $\mathcal{X}$ as described in the next section.

**Refinement**

Refinement is a heuristic that constructs a new partition of $\mathcal{X}$, a subpartition of $\{\mathcal{X}_i\}$, that is used in the next iteration of the overall algorithm. We design two refinement procedures, called positive and negative, that aim to enlarge the combined volume of polytopes in the set $S_{\mathsf{yes}}$ and $S_{\mathsf{no}}$, respectively, or equivalently, to reduce the combined volume of polytopes in the set $S_?$. Based on Propositions 7 and 9, both procedures are formulated over the product game $\mathcal{P}_{\{\mathcal{X}_i\}}$ and reach their respective goals through refining polytopes $\mathcal{X}_i$ for which $(\mathcal{X}_i, q) \in S_?^{\mathcal{P}}$ for some $q \in Q$.

In this section, we use $J_{\mathsf{yes}}^q$ to denote the set of all indices $i \in I$ for which $(\mathcal{X}_i, q) \in S_{\mathsf{yes}}^{\mathcal{P}}$, and $J_?^q, J_{\mathsf{no}}^q$ are defined analogously. In the two refinement procedures, every polytope $\mathcal{X}_i$ can be partitioned into a set of polytopes in iterative manner, as $(\mathcal{X}_i, q) \in S_?^{\mathcal{P}}$ can hold for multiple $q \in Q$. Therefore, given a partition of $\mathcal{X}_i$, the refinement of $\mathcal{X}_i$ according to a polytope $\mathcal{B}$ refers to the partition of $\mathcal{X}_i$ that contains all intersections and differences of elements of the original partition of $\mathcal{X}_i$ and polytope $\mathcal{B}$.

In the positive refinement, we explore the following property of states in $S_?^{\mathcal{P}}$. In Proposition 11, we proved that the product game $\mathcal{P}_{\{\mathcal{X}_i\}}$ can be won cooperatively from every $(\mathcal{X}_i, q) \in S_?^{\mathcal{P}}$. It follows that there exists a Player 1 action $\mathcal{U}_i^J$ and Player 2 action $J'$ such that after their application in $(\mathcal{X}_i, q)$, the game is not in a losing state with probability 1. We can graphically represent this property as follows:

$$(\mathcal{X}_i, q) \xrightarrow{\mathcal{U}_i^J} ((\mathcal{X}_i, \mathcal{U}_i^J), q') \xrightarrow{J'} \begin{cases} (\mathcal{X}_{j_1}, q') \\ \vdots \\ (\mathcal{X}_{j_n}, q') \end{cases} \tag{4.10}$$

where an arrow $a \xrightarrow{b}$ represents the uniform probability distribution $P_{\mathcal{P}}(a, b)$, and $\{j_1, \ldots, j_n\} = J' \subseteq J_{\mathsf{yes}}^{q'} \cup J_?^{q'}$. Note that from the construction of the product game $\mathcal{P}_{\{\mathcal{X}_i\}}$ it follows that $q'$ is given uniquely over all actions $\mathcal{U}_i^J$. The following design ensures that every polytope $\mathcal{X}_i$ is refined at least once for every its appearance $(\mathcal{X}_i, q) \in S_?^{\mathcal{P}}, q \in Q$.

Let $(\mathcal{X}_i, q) \in S_?^{\mathcal{P}}$. The positive refinement first refines $\mathcal{X}_i$ according to the robust predecessor

$$\mathrm{PreR}(\mathcal{X}_i, \mathcal{U}, \{\mathcal{X}_j\}_{j \in J_{\mathsf{yes}}^{q'}}). \tag{4.11}$$

That means, we find all states $x \in \mathcal{X}_i$ for which there exists any control input under which the system $\mathcal{L}$ evolves from $x$ to a state $x' \in \mathcal{X}_j, j \in J_{\mathsf{yes}}^{q'}$.

Next, the positive refinement considers three cases. First, assume that from $(\mathcal{X}_i, q)$, the two players can cooperatively reach a winning state of the product

game in two steps with probability 1, and let $\mathcal{U}_i^J$ and $J'$ be Player 1 and Player 2 actions, respectively, that accomplish that, i.e., in Equation 4.10, $\{j_1, \ldots, j_n\} = J' \subseteq J_{\mathsf{yes}}^{q'}$. For every such $\mathcal{U}_i^J, J'$, we find an (arbitrary) partition $\{\mathcal{U}_y\}_{y \in Y}$ of the polytope $\mathcal{U}_i^J$ and we partition $\mathcal{X}_i$ according to the robust attractors

$$\mathrm{AttrR}(\mathcal{X}_i, \mathcal{U}_y, \{\mathcal{X}_j\}_{j \in J_{\mathsf{yes}}^{q'}}). \tag{4.12}$$

Intuitively, the above set contains all $x \in \mathcal{X}_i$ such that under every control input $u \in \mathcal{U}_y$, $\mathcal{L}$ evolves from $x$ to a state $x' \in \mathcal{X}_j, j \in J_{\mathsf{yes}}^{q'}$. Note that the robust attractor sets partition the robust predecessor set from Equation 4.11, as every state $x$ that belongs to one of the robust attractor set must lie in the robust predecessor set as well. In the next iteration of the overall algorithm, the partition elements given by the robust attractor sets will belong to the set $S_{\mathsf{yes}}^{\mathcal{P}}$. In the second case, assume that the two players can reach a winning state of the product game cooperatively in two steps, but only with probability $0 < p < 1$, while the probability of reaching a losing state is 0. Let $\mathcal{U}_i^J, J'$ be Player 1 and Player 2 actions, respectively, that maximize $p$, i.e., in Equation 4.10, there exists $m < n$ such that $\{j_1, \ldots, j_m\} = J' \cap J_{\mathsf{yes}}^{q'}$, $\{j_{m+1}, \ldots, j_n\} = J' \cap J_?^{q'}$ and $p = \frac{m}{n}$ is maximal. Similarly as in the first case, we refine the polytope $\mathcal{X}_i$ according to the robust attractor sets as in Equation 4.12, but we compute the sets with respect to the set of indices $J_{\mathsf{yes}}^{q'} \cup \{j_{m+1}, \ldots, j_n\}$. Finally, assume that $(\mathcal{X}_i, q)$ does not belong to any of the above two categories. As argued at the beginning of this section, there still exist Player 1 and Player 2 actions $\mathcal{U}_i^J$ and $J'$, respectively, such that in Equation 4.10, $\{j_1, \ldots, j_n\} = J' \subseteq J_?^{q'}$. Again, we refine the polytope $\mathcal{X}_i$ according to the robust attractor sets as in Equation 4.12, where the sets are computed with respect to the set of indices $J_?^{q'}$.

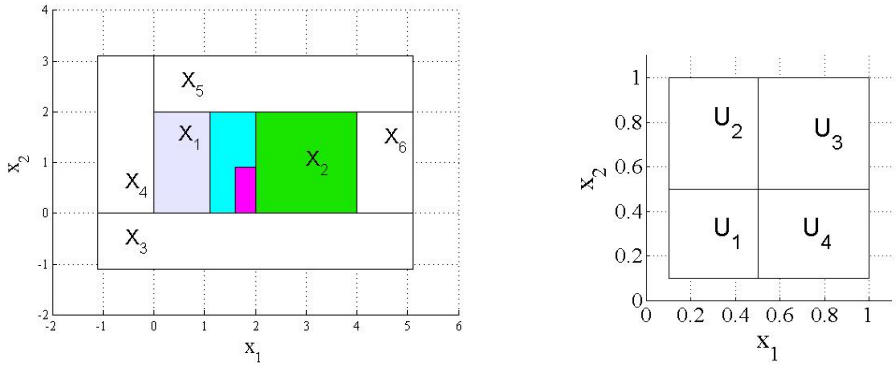
**Example 8.** We demonstrate a part of the the positive refinement for the game in Example 7. Consider polytope $\mathcal{X}_1 \in S_?$. It follows from the form of the $\omega$-automaton in Example 7 that $\mathcal{X}_1$ appears in $S_?^{\mathcal{P}}$ only in pair with $q_0$, i.e., $(\mathcal{X}_1, q_0) \in S_?^{\mathcal{P}}$. Note that for state $(\mathcal{X}_1, q_0)$, every successor state is of the form $((\mathcal{X}_1, \mathcal{U}_1^J), q_0)$, i.e., $q' = q_0$. First, polytope $\mathcal{X}_1$ is refined with respect to the robust predecessor

$$\mathrm{PreR}(\mathcal{X}_1, \mathcal{U}, \{\mathcal{X}_2\}),$$

since $J_{\mathsf{yes}}^{q_0} = \{\mathcal{X}_2\}$ because $(\mathcal{X}_2, q_0) \in S_{\mathsf{yes}}^{\mathcal{P}}$ is a winning state of the product game. The robust predecessor set is depicted in Figure 4.4a in cyan. Next, we decide which of the three cases described in the positive refinement procedure above applies to state $(\mathcal{X}_1, q_0)$. Consider for example Player 1 action $\mathcal{U}_1^{\{1,2,5\}}$ and Player 2 action $\{2\}$, as shown in Figure 4.2b. It holds that

$$(\mathcal{X}_1, q_0) \xrightarrow{\mathcal{U}_1^{\{1,2,5\}}} ((\mathcal{X}_1, \mathcal{U}_1^{\{1,2,5\}}), q_0) \xrightarrow{\{2\}} (\mathcal{X}_2, q_0),$$

and $(\mathcal{X}_2, q_0) \in S_{\mathsf{yes}}^{\mathcal{P}}$ is a winning state of the product game. Therefore, the state $(\mathcal{X}_1, q_0)$ is of the first type. To further refine polytope $\mathcal{X}_1$, we first partition the




(a) (b)

**Figure 4.4:** (a) Part of the positive refinement for the system in Example 8. Polytope $\mathcal{X}_1$ is first refined according to the robust predecessor as in Equation 4.11, the robust predecessor is shown in cyan. Next, we consider the polytope of control inputs $\mathcal{U}_1^{\{1,2,5\}}$ and its partition as depicted in (b). The robust predecessor of $\mathcal{U}_3$ is then shown in magenta.

polytope

$$\mathcal{U}_1^{\{1,2,5\}} = \{u \in \mathcal{U} \mid 0.1 \leq u(1), u(2) \leq 1\},$$

e.g., into 4 parts as shown in Figure 4.4b. The robust attractor

$$\mathrm{AttrR}(\mathcal{X}_1, \mathcal{U}_3, \{\mathcal{X}_2\})$$

for one of the polytopes $\mathcal{U}_3$ is depicted in magenta in Figure 4.4a. This polytope will be recognized as a satisfying initial polytope in the next iteration, since starting in any $x$ within the robust predecessor, system $\mathcal{L}$ as defined in Example 6 evolves from $x$ under every control input from $\mathcal{U}_3$ to polytope $\mathcal{X}_2$.

In the negative refinement, we consider all Player 1 states $(\mathcal{X}_i, q) \in S_?^{\mathcal{P}}$ such that if Player 2 does not cooperate, but rather plays against Player 1, the game is lost with non-zero probability. In other words, for every Player 1 $\mathcal{U}_i^J$, there exists a Player 2 action $J'$ such that in Equation 4.10, there exists an index $j \in J'$ such that $(\mathcal{X}_j, q') \in S_{\mathsf{no}}^{\mathcal{P}}$. In this case, we refine polytope $\mathcal{X}_i$ according to the attractor set

$$\mathrm{Attr}(\mathcal{X}_i, \mathcal{U}, \{\mathcal{X}_j\}_{j \in J_{\mathsf{no}}^{q'}}).$$

Intuitively, the attractor set contains all states $x \in \mathcal{X}_i$ such that by applying any control input $u \in \mathcal{U}$, system $\mathcal{L}$ evolves from $x$ to a state in $\mathcal{X}_j$ for some $j \in J_{\mathsf{no}}^{q'}$ with non-zero probability. In the next iteration of the algorithm, the partition elements given by the attractor set will belong to the set $S_{\mathsf{no}}^{\mathcal{P}}$.

**Remark 6.** We remark that both game theoretic aspects as well as the linear stochastic dynamics play an important role in the refinement step. The game theoretic results compute the undecided states, and thereby determine what parts of the state space need to be refined and which actions need to be considered in the refinement. The linear stochastic dynamics allow us to perform the refinement itself using polytopic operators.

### Properties of the solution

We prove that the algorithm presented in this section provides a partially correct solution to Problem 5.

For $n \in \mathbb{N}$, let $S^n_{\mathsf{yes}}, S^n_{\mathsf{no}}$ be the sets from Equations 4.4 and 4.6, respectively, computed in the $n$-th iteration of the algorithm presented above. We use $\mathcal{X}^n_{\mathsf{yes}}, \mathcal{X}^n_{\mathsf{no}} \subseteq \mathcal{X}$ to denote the union of polytopes from $S^n_{\mathsf{yes}}$ and $S^n_{\mathsf{no}}$, respectively.

**Theorem 5.** For every $n \in \mathbb{N}$, it holds that $\mathcal{X}^n_{\mathsf{yes}} \subseteq \mathcal{X}^{n+1}_{\mathsf{yes}}$ and $\mathcal{X}^n_{\mathsf{no}} \subseteq \mathcal{X}^{n+1}_{\mathsf{no}}$.

*Proof.* Follows from Propositions 7 and 9, and the fact that the partition of the state space $\mathcal{X}$ used in $n+1$-th iteration is a subpartition of the one used in $n$-th iteration. □

**Theorem 6.** For every $n \in \mathbb{N}$, it holds that $\mathcal{X}^n_{\mathsf{yes}} \subseteq \mathcal{X}_{\mathsf{init}}$ and $\mathcal{X}^n_{\mathsf{no}} \subseteq \mathcal{X} \backslash \mathcal{X}_{\mathsf{init}}$.

*Proof.* Follows directly from Propositions 8 and 10. □

**Theorem 7.** If the algorithm from Sec. 4.1.3 terminates, after $n$-th iteration, then $\mathcal{X}_{\mathsf{init}} = \mathcal{X}^n_{\mathsf{yes}}$ is the solution of Problem 5 and the corresponding winning strategies for every $x \in \mathcal{X}_{\mathsf{init}}$ are given by the winning strategies in the $2^1/_2$-player game from the last iteration.

*Proof.* Follows directly from the condition of the algorithm termination and from Theorems 5 and 6. □

It is important to note that if instead of a $2^1/_2$-player game a weaker abstraction model such as a 2 player game, i.e., the NTS $\mathcal{N}_{\{\mathcal{X}_i\}}$, was used, our approach would not be sound. Namely, some states of $\mathcal{X}$ might be wrongfully identified as non-satisfying initial states based on behavior that has zero probability in the original stochastic system. In such a case, even after termination, the resulting set would only be a subset of $\mathcal{X}_{\mathsf{init}}$. Therefore, the approach with 2-player games is not complete. The $2^1/_2$-player game is needed to account for both the non-determinism introduced by the abstraction and for the stochasticity of the system to be able to recognize (non-satisfying) behavior of zero probability.

Note that there exist linear stochastic systems for which our algorithm does not terminate, i.e., there does not exist a finite partition of the systems' state space over which Problem 5 can be solved for a given GR(1) formula.

**Example 9.** Let $\mathcal{L}$ be a linear stochastic system of the form given in Equation 4.1.2, where

$$A = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, B = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix},$$

state space $\mathcal{X} = \{x \in \mathbb{R}^2 \mid 0 \leq x(1), x(2) \leq 3\}$, control space $\mathcal{U} = \{u \in \mathbb{R}^2 \mid -1.5 \leq u(1), u(2) \leq 1.5\}$ and the random vector takes values in polytope $\mathcal{W} = \{w \in \mathbb{R}^2 \mid -0.5 \leq w(1), w(2) \leq 0.5\}$. Let $AP$ contain four linear predicates that partition the state space into a grid of three by three equally sized square polytopes. Assume that the aim is to eventually reach the polytope $\mathcal{X}_f$, where $1 \leq x(1), x(2) \leq 2$. In this case, the maximal set $\mathcal{X}_{\mathsf{init}}$ of states from which $\mathcal{X}_f$ can be reached with probability 1 is the whole state space $\mathcal{X}$, as for any $x \in \mathcal{X}$, there exists exactly one control input $u = (1.5, 1.5) - x \in \mathcal{U}$ that leads the system $\mathcal{L}$ from $x$ to a state in $\mathcal{X}_5$ with probability 1. Since the control input is different for every $x \in \mathcal{X}$, there does not exist any finite state space partition, which could be used to solve Problem 5.

#### Complexity analysis

Finally, let us analyze the computational complexity of the designed algorithm. In the abstraction part, the $2^1/_2$-player game $\mathcal{G}_{\{\mathcal{X}_i\}}$ requires to first compute the set of actions for every state $\mathcal{X}_i, i \in I$, in time in $\mathcal{O}(2^{|I|})$ using algorithm in Appendix C.1. For every action $\mathcal{U}_i^J$, the set of valid supports $J' \subseteq J$ is then computed in time in $\mathcal{O}(2^J)$, see Appendix C. Overall, the abstraction runs in time in $\mathcal{O}(2^{2\cdot|I|})$. The game is then analyzed using the algorithm described in Appendix B in time in $\mathcal{O}(|I|^3)$. Finally, the refinement procedure iteratively refines every polytope $\mathcal{X}_i$ at most $|Q| \times |\{\mathcal{U}_i^J\}|$ times, where $\{\mathcal{U}_i^J\}$ denotes the set of all actions of $\mathcal{X}_i$. For every $q \in Q$ such that $(\mathcal{X}_i, q) \in S_?^{\mathcal{P}}$, $\mathcal{X}_i$ is first refined using the robust predecessor operator in time exponential in $|J_{\mathsf{yes}}^{q'}|$. Then $\mathcal{X}_i$ is refined $|Y|$ times using the robust attractor operator in polynomial time. Negative refinement is performed again for every $q \in Q$ such that $(\mathcal{X}_i, q) \in S_?^{\mathcal{P}}$, using the attractor operator in polynomial time. Overall, the refinement runs in time in $\mathcal{O}(|Q| \cdot 2^{|I|})$.

As the game construction is the most expensive part of the overall algorithm, the refinement procedure is designed in a way that extends both sets $S_{\mathsf{yes}}, S_{\mathsf{no}}$ as much as possible and thus speed up convergence and minimize the number of iterations of the overall algorithm.

### 4.1.4 Case Study

We demonstrate the designed framework on a discrete-time double integrator dynamics with uncertainties. Let $\mathcal{L}$ be a linear stochastic system as defined in Definition 12, where

$$A = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}, \quad B = \begin{pmatrix} 0.5 \\ 1 \end{pmatrix}. \tag{4.13}$$

The state space is $\mathcal{X} = \{x \in \mathbb{R}^2 \mid -5 \leq x(1) \leq 5, -3 \leq x(2) \leq 3\}$ and the control space is $\mathcal{U} = \{u \in \mathbb{R} \mid -1 \leq u \leq 1\}$. The random vector, or uncertainty, takes values within polytope $\mathcal{W} = \{w \in \mathbb{R}^2 \mid -0.1 \leq w(1), w(2) \leq 0.1\}$. The set $AP$ consists of 4 linear predicates:

$$\begin{aligned} a_1 &: x(1) \leq -1, \\ a_2 &: x(1) \leq 1, \\ a_3 &: x(2) \leq -1, \\ a_4 &: x(2) \leq 1. \end{aligned}$$

We consider GR(1) formula

$$\mathbf{F}(\neg a_1 \wedge a_2 \wedge \neg a_3 \wedge a_4)$$

that requires the system to eventually reach a state, where both variables of the system have values in interval $(-1, 1)$.

As we consider a reachability property, we can compare our approach to the two following algorithms. First, the algorithm shown in Algorithm 5 is an extension of the reachability algorithm for Markov decision processes [BK08] to linear stochastic systems. Intuitively, the algorithm finds the set $\mathcal{X}_{\mathsf{init}}$ using two fixed-point computations. The first one computes the set of all states that can reach the given target polytopes with non-zero probability. As a result, the remaining states of the state space $\mathcal{X}$ have zero probability of reaching the target polytopes. The second fixed-point computation finds the attractor of this set, i.e., all states that have non-zero probability, under each control input from $\mathcal{U}$, of ever transiting to a state from which the target polytopes cannot be reached. Finally, the complement of the attractor is the desired set $\mathcal{X}_{\mathsf{init}}$.

The second algorithm summarized in Algorithm 6 combines the simple approach from Algorithm 5 that uses only polytopic operations with the abstraction-refinement method. In every iteration, we build the non-deterministic transition system $\mathcal{N}_{\{\mathcal{X}_i\}}$, which is the first step of the abstraction in Section 4.1.3. The partition $\{\mathcal{X}_i\}_{i \in I}$ is then iteratively refined using the two fixed-point algorithms as in Algorithm 5.

Note that both Algorithms 5 and 6 operate directly on the linear stochastic system. Algorithm 5 performs polytopic operations only, and it involves neither refinement nor building a product with an automaton. In Algorithm 6, abstraction and its refinement are performed using only polytopic operators and no product is built. Therefore, both algorithms perform considerably faster than the abstraction-refinement algorithm from Section 4.1.3, as shown in Table 4.2. However, they have two serious drawbacks.

Firstly, Algorithm 5 computes the set of satisfying initial states of the system, but no satisfying strategy. In extreme cases, every state may use a different control input in order to reach polytopes computed during the fixed-point computations, as in Example 9. In order to extract a finite satisfying strategy (if there is one),

**Table 4.2:** Statistical comparison of the specialized algorithms for reachability and our approach.

| | |
|---|---|
| Algorithm 5 | |
| 1st fixed point: | in 7 iterations, in ¡1 sec. |
| 2nd fixed point: | in 1 iteration, in ¡1 sec. |
| Algorithm 6 | |
| 1st fixed point: | in 7 iterations, in 3 min. |
| | 1st NTS: 13 states, 27 actions |
| | 2nd NTS: 25 states 105 actions |
| | 3rd NTS: 45 states 289 actions |
| | 4th NTS: 63 states, 524 actions |
| | 5th NTS: 77 states, 745 actions |
| | 6th NTS: 88 states, 994 actions |
| | 7th NTS: 92 states, 1139 actions |
| 2nd fixed point: | in 1 iteration, in 2 sec. |
| Abstraction-refinement from Section 4.1.3 | |
| Initial partition: | in 3 sec. |
| | game: 13 states, 27 actions |
| 1st iteration: | in 7 min. |
| | game: 85 states, 712 actions |
| 2nd iteration: | in 19 min. |
| | game: 131 states, 1262 actions |
| 3rd iteration: | in 56 min. |
| | game: 250 states, 2724 actions |

**Algorithm 5** Simple algorithm for computing the set $\mathcal{X}_{\mathsf{init}} \subseteq \mathcal{X}$ of states from which a set of polytopes $\{\mathcal{X}_j\}_{j\in J}$ in $\mathcal{X}$ can be reached with probability 1.

Input: linear stochastic system $\mathcal{L}$, polytopes $\{\mathcal{X}_j\}_{j\in J}$
Output: $\mathcal{X}_{\mathsf{init}} \subseteq \mathcal{X}$
$\mathcal{X}_{>0} \leftarrow \{\mathcal{X}_j\}_{j\in J}$
**while** $\mathcal{X}_{>0}$ is not a fixed point **do**
  $\mathcal{X}_{>0} \leftarrow \mathrm{Pre}(\mathcal{X},\mathcal{U},\mathcal{X}_{>0})$
**end while**
$\mathcal{X}_{=0,\mathsf{attr}} \leftarrow \mathcal{X}_{\mathsf{out}} \cup \mathcal{X}\backslash\mathcal{X}_{>0}$
**while** $\mathcal{X}_{=0,\mathsf{attr}}$ is not a fixed point **do**
  $\mathcal{X}_{=0,\mathsf{attr}} \leftarrow \mathrm{Attr}(\mathcal{X},\mathcal{U},\mathcal{X}_{=0,\mathsf{attr}})$
**end while**
$\mathcal{X}_{=1} \leftarrow \mathcal{X}\backslash\mathcal{X}_{=0,\mathsf{attr}}$
return $\mathcal{X}_{=1}$

these polytopes have to be partitioned to smaller polytopes so that a fixed input can be used in all states of the new polytope. This partitioning is exactly the refinement procedure that our method performs when applied to reachability. Note that the simpler refinement method in Algorithm 6 that constructs only the NTS $\mathcal{N}_{\{\mathcal{X}_i\}}$ is also not sufficient. While it finds the set of satisfying initial states of the system, it does not provide the satisfying strategies either. However, in comparison with Algorithm 5, it can provide at least a partial information on the properties of satisfying strategies. Namely, it specifies for every polytope of the resulting partition $\{\mathcal{X}_i\}$ of the state space $\mathcal{X}$ which control inputs cannot be used in any satisfying strategy. As we are interested in reachability property, these are the control inputs for which the corresponding non-deterministic transition leads from $\mathcal{X}_i$ outside of $\mathcal{X}\mathcal{X}_{init}$. The whole $2^1/_2$-player game abstraction presented in Section 4.1.3 is necessary for ensuring the correctness of the constructed strategy.

Secondly, neither of the two algorithms can be used for more complex properties than reachability. For more complex formulas, the product of the game with the automaton for the formula needs to be considered, since a winning strategy may require memory and pure polytopic methods can only provide memoryless strategies. In contrast, our abstraction-refinement approach designed in Section 4.1.3 works for general GR(1) properties. Moreover, it could easily be extended to the whole LTL at the cost of a higher complexity.

We implemented all three algorithms in Matlab, on a dual-core Intel i7 processor with 8 GB of RAM. The results are summarized in Figure 4.5 and Table 4.2. In Figure 4.5, for Algorithm 5, we first depict the initial partition of $\mathcal{X}$ according to $AP$, with the polytope we aim to reach in green. The following columns show the two fixed point sets, in blue and red, respectively. The last column shows the resulting set $\mathcal{X}_{\mathsf{init}}$. For Algorithm 6, in the first column, we depict the initial partition of $\mathcal{X}$ according to $AP$ and polytopes from $\mathcal{X}_{\mathsf{out}}$, with the polytope we aim to reach in green. The following columns show the fixed point sets, in blue and red, respectively, together with the obtained partition. The last column shows

**Algorithm 6** Computing the set $\mathcal{X}_{\mathsf{init}} \subseteq \mathcal{X}$ of states from which a set of polytopes $\{\mathcal{X}_j\}_{j\in J}$ in $\mathcal{X}$ can be reached with probability 1, using abstraction to an NTS.

Input: linear stochastic system $\mathcal{L}$, partition $\{\mathcal{X}_i\}_{i\in I}$ of state space $\mathcal{X}$, subset $J \subseteq I$
Output: $\mathcal{X}_{\mathsf{init}} \subseteq \mathcal{X}$
$\mathcal{X}_{>0} \leftarrow \emptyset$
$\mathcal{X}'_{>0} \leftarrow \{\mathcal{X}_j\}_{j\in J}$
**while** $\mathcal{X}_{>0} \neq \mathcal{X}'_{>0}$ **do**
  $\mathcal{X}_{>0} \leftarrow \mathcal{X}'_{>0}$
  construct NTS $\mathcal{N}_{\{\mathcal{X}_i\}}$ for current partition
  **for** every state $\mathcal{X}_i \not\subseteq \mathcal{X}_{>0}$ **do**
    refine $\mathcal{X}_i$ according to $\mathrm{Pre}(\mathcal{X}_i, \mathcal{U}, \mathcal{X}_{>0})$
    $\mathcal{X}'_{>0} \leftarrow \mathcal{X}'_{>0} \cup \mathrm{Pre}(\mathcal{X}_i, \mathcal{U}, \mathcal{X}_{>0})$
  **end for**
**end while**
$\mathcal{X}_{=0,\mathsf{attr}} \leftarrow \mathcal{X}_{\mathsf{out}} \cup \mathcal{X}$
$\mathcal{X}'_{=0,\mathsf{attr}} \leftarrow \mathcal{X}_{\mathsf{out}} \cup \mathcal{X} \backslash \mathcal{X}_{>0}$
**while** $\mathcal{X}_{=0,\mathsf{attr}} \neq \mathcal{X}'_{=0,\mathsf{attr}}$ **do**
  $\mathcal{X}_{=0,\mathsf{attr}} \leftarrow \mathcal{X}'_{=0,\mathsf{attr}}$
  construct NTS $\mathcal{N}_{\{\mathcal{X}_i\}}$ for current partition
  **for** every state $\mathcal{X}_i$ s.t. all actions lead to $\mathcal{X}_{=0,\mathsf{attr}}$ **do**
    refine $\mathcal{X}_i$ according to $\mathrm{Attr}(\mathcal{X}_i, \mathcal{U}, \mathcal{X}_{=0,\mathsf{attr}})$
    $\mathcal{X}'_{=0,\mathsf{attr}} \leftarrow \mathcal{X}'_{=0,\mathsf{attr}} \cup \mathrm{Attr}(\mathcal{X}_i, \mathcal{U}, \mathcal{X}_{=0,\mathsf{attr}})$
  **end for**
**end while**
$\mathcal{X}_{=1} \leftarrow \mathcal{X} \backslash \mathcal{X}_{=0,\mathsf{attr}}$
return $\mathcal{X}_{=1}$

the resulting set $\mathcal{X}_{\mathsf{init}}$. Finally, for the abstraction-refinement from Section 4.1.3, we depict the results of for the initial partition and the next three iterations, where polytopes from sets $S_{\mathsf{yes}}, S_?, S_{\mathsf{no}}$ are shown in green, light blue and white, respectively.

As can be seen in Table 4.2, the set $\mathcal{X}_{\mathsf{init}}$ was computed fast using Algorithm 5 but it is a single polytope that does not provide any information about the satisfying strategies. Algorithm 6 also found fixed point sets for both fixed-point computation rather quickly, and in the same number of iterations as the polytopic algorithm, but it provides only partial information on the satisfying strategies, as discussed above. For the abstraction-refinement algorithm, we computed the initial game and the following three iterations. Unlike for Algorithms 5 and 6, in every iteration, a satisfying strategy for a state $x$ in the partial solution is constructed as described in the proof of Proposition 8.

### 4.1.5 Conclusion

In this section, we considered the problem of computing the set of initial states of a linear stochastic system such that there exists a control strategy to ensure a

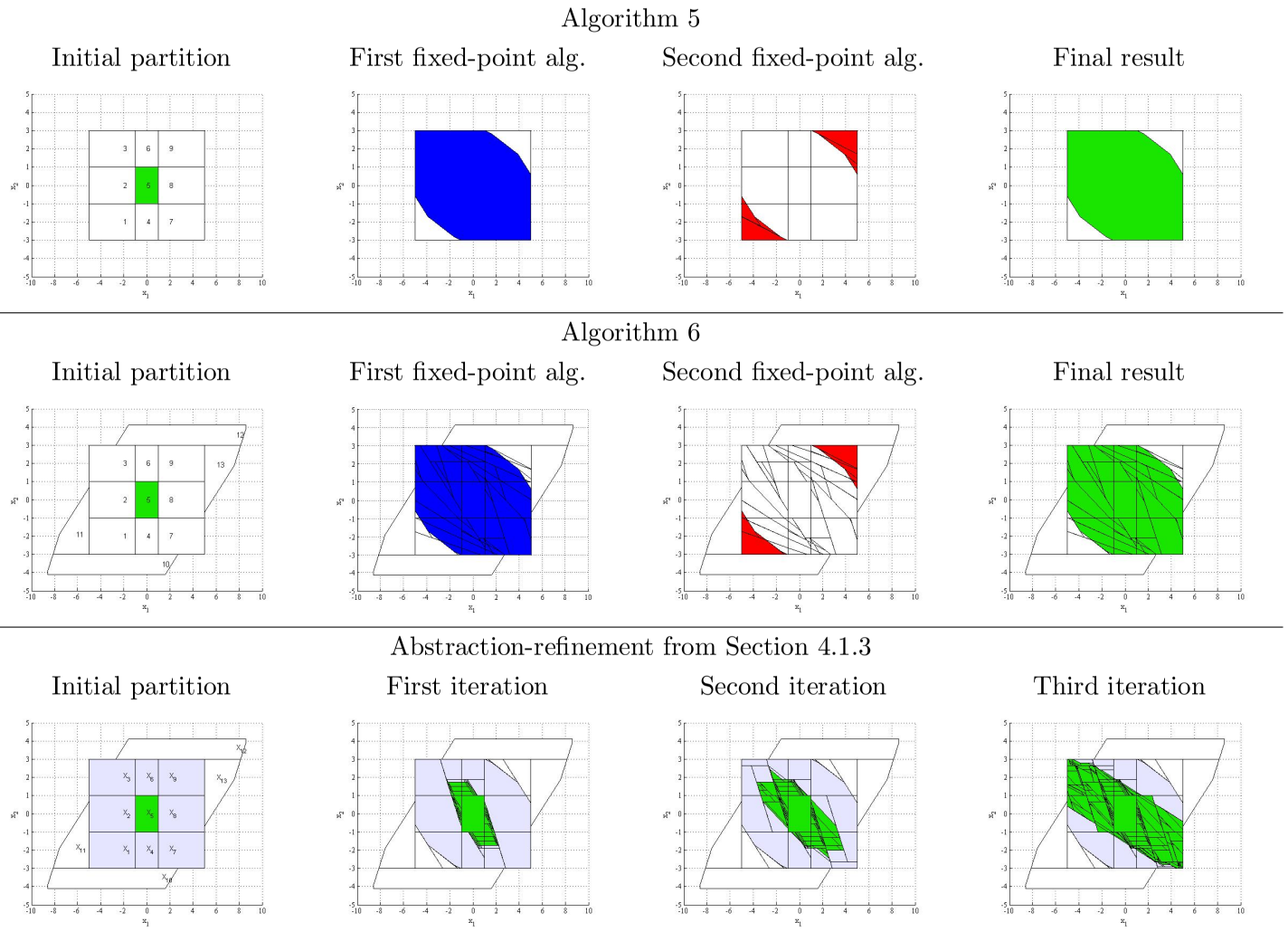



**Figure 4.5:** Simulation of Algorithms 5 and 6, and the abstraction-refinement algorithm.

GR(1) specification over states of the system. The solution is based on iterative abstraction-refinement using a $2^1/_2$-player game. Every iteration of the algorithm provides a partial solution given as a set of satisfying initial states with the satisfying strategies, and a set of non-satisfying initial states.

While the algorithm guarantees progress and soundness in every iteration, it's complexity calls for more efficient implementation. The analyzed case study with a reachability property indicates that the current design would be too complex to deal with more complex properties such as persistent surveillance. In our future work, we aim to design efficient heuristic refinements that minimize the overall computation time for both reachability and general GR(1).

# Chapter 5

# Summary and Future Work

In the last few decades, the rapid evolution of mobile and autonomous robotics has shown the need for provably correct and robust approaches to design and control of these systems. Formal methods from computer science offer a great tool to this end. The goal of this thesis was to design control synthesis algorithms for chosen problems motivated by path planning for mobile robots that leverage ideas and techniques from theoretical computer science such as formal verification and game theory. The problems of interest involve a complex model of the system as well as a complex temporal objective to be satisfied. We embraced the standard hierarchical approach to employing formal methods in path planning. The approach consists of first modeling the complex dynamic system using a discrete model, then synthesizing a provably correct control strategy for the model using formal methods and finally mapping the control rules to the original system.

In the first part of the thesis, we assumed that a discrete model of the system is already given and we focused on the second, synthesis step of the hierarchical approach. We designed algorithms to synthesize strategies that guarantee satisfaction of an LTL formula, while at the same time optimize a value function over (possibly) dynamic and partially observed values interpreted either as rewards or penalties. Such a combination of optimal and temporal logic control is an intriguing problem with a potentially high impact in applications such as control of a mobile robot on a complex mission under tight fuel and time constraints. The solution combined receding horizon control with techniques from game theory and automata-based model checking.

In the second part, we considered the more general problem of synthesizing control for a stochastic dynamic system described using a difference equation. The objective was to satisfy a complex temporal specification over the state space of the system. We designed an iterative abstraction-refinement algorithm that builds an abstraction of the system using a $2^1/_2$-player game, solves the game obtaining a partial solution and then builds a new, more precise abstraction using a deep analysis of the game. In the abstract phase, the probability 1 analysis allowed us to effectively construct the game using only polytopic operators and thus avoiding

expensive and often approximative computation of integrals. Similarly, polytopic operators are used to perform the refinement, while the deep analysis of the game specifies what parts of the state space need to be refined and which actions need to be considered in the process.

All designed algorithms build on the standard as well as recent results from both computer science and control theory and thus make a step towards interconnecting the two areas. In our future work, we primarily aim to follow the line of research considered in the second part of the thesis, i.e., control of dynamic systems with respect to temporal objectives. We believe that in order to maximize the benefit of the combination of formal and engineering methods, the integration of the techniques must take place on all three levels of the hierarchical approach such as in Chapter 4, not only in the synthesis step. From the experiments in Section 4.1.4 it appears that this comes with a great computational cost. In the future, we aim to investigate the causes of this complexity and design effective solutions that would allow us to demonstrate the algorithms on real-scale applications such as autonomous driving.

# Bibliography


[ADD00] R.B. Ash and C.A. Doléans-Dade. *Probability & Measure Theory.* Academic Press, 2000.

[ADDB11] A. Abate, A. D'Innocenzo, and M.D. Di Benedetto. Approximate Abstractions of Stochastic Hybrid Systems. *IEEE Transactions on Automatic Control*, 56(11):2688–2694, 2011.

[AG11] K.R. Apt and E. Grädel. *Lectures in Game Theory for Computer Scientists.* Cambridge University Press, 2011.

[AM08] K.J. Astrom and Richard M. Murray. *Feedback Systems: An Introduction for Scientists and Engineers.* Princeton University Press, 2008.

[Ber01] J.A. Bergstra. *Handbook of Process Algebra.* Elsevier Science Inc., 2001.

[Ber12] D.P. Bertsekas. *Dynamic Programming and Optimal Control Volume I & II.* Athena Scientific, 2005 & 2012.

[BK08] C. Baier and J.-P. Katoen. *Principles of Model Checking.* MIT Press, 2008.

[BKS13] F. Blahoudek, M. Křetínský, and J. Strejček. Comparison of LTL to Deterministic Rabin Automata Translators. In *Proc. of Conference on Logic for Programming, Artificial Intelligence and Reasoning LPAR*, pages 164–172, 2013.

[BKV10] A. Bhatia, L.E. Kavraki, and M.Y. Vardi. Sampling-based motion planning with temporal goals. In *Proc. of IEEE Conference on Robotics and Automation ICRA*, pages 2689–2696, 2010.

[CB12] I. Cizelj and C. Belta. Probabilistically safe control of noisy Dubins vehicles. In *Proc. of IEEE/RSJ Conference on Intelligent Robots and Systems IROS*, pages 2857–2862, 2012.

[CBRZ01] E. Clarke, A. Biere, R. Raimi, and Y. Zhu. Bounded Model Checking Using Satisfiability Solving. *Formal Methods in System Design*, 19(1):7–34, 2001.

[CCD14] K. Chatterjee, M. Chmelík, and P. Daca. CEGAR for qualitative analysis of probabilistic systems. In *Proc. of Conference on Computer-Aided Verification CAV*, pages 473–490, 2014.

[CCGK15] K. Chatterjee, M. Chmelík, R. Gupta, and A. Kanodia. Qualitative Analysis of POMDPs with Temporal Logic Specifications for Robotics Applications. In *Proc. of IEEE Conference on Robotics and Automation ICRA*, 2015. To appear.

[CD10] K. Chatterjee and L. Doyen. The Complexity of Partial-Observation Parity Games. In *Proc. of Conference on Logic for Programming, Artificial Intelligence and Reasoning LPAR*, volume 6397 of *Lecture Notes in Computer Science LNCS*, pages 1–14. 2010.

[CD12] K. Chatterjee and L. Doyen. Games and Markov Decision Processes with Mean-Payoff Parity and Energy Parity Objectives. In *Proc. of Workshop on Mathematical and Engineering Methods in Computer Science MEMICS*, volume 7119 of *Lecture Notes in Computer Science LNCS*, pages 37–46. 2012.

[CdAH11] K. Chatterjee, L. de Alfaro, and T.A. Henzinger. Qualitative concurrent parity games. *ACM Transactions on Computational Logic*, 12(4), 2011.

[CDH12] K. Chatterjee, L. Doyen, and T.A. Henzinger. A survey of partial-observation stochastic parity games. *Formal Methods in System Design*, pages 1–17, 2012.

[CE82] E.M. Clarke and E.A. Emerson. Design and Synthesis of Synchronization Skeletons Using Branching-Time Temporal Logic. In *Logic of Programs, Workshop*, pages 52–71, 1982.

[CGH97] E. Clarke, O. Grumberg, and K. Hamaguchi. Another Look at LTL Model Checking. *Formal Methods in System Design*, 10(1):47–71, 1997.

[CGJ+00] E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-Guided Abstraction Refinement. In *Proc. of Conference on Computer-Aided Verification CAV*, volume 1855 of *Lecture Notes in Computer Science LNCS*, pages 154–169. 2000.

[CH11] K. Chatterjee and M. Henzinger. Faster and dynamic algorithms for maximal end-component decomposition and related graph problems in probabilistic verification. In *Proc. of ACM-SIAM Symposium on Discrete Algorithms*, pages 1318–1336, 2011.

[CH12] K. Chatterjee and T.A. Henzinger. A survey of stochastic $\omega$-regular games. *Journal of Computer and System Sciences*, 78(2):394–413, 2012.

[Cha07] K. Chatterjee. *Stochastic $\omega$-Regular Games*. PhD thesis, UC Berkeley, 2007.

[CHJM05] K. Chatterjee, T.A. Henzinger, R. Jhala, and R. Majumdar. Counterexample-guided Planning. In *Proc. of Conference on Uncertainty in Artificial Intelligence UAI*, pages 104–111, 2005.

[CKSW13] T. Chen, M. Kwiatkowska, A. Simaitis, and C. Wiltsche. Synthesis for Multi-objective Stochastic Games: An Application to Autonomous Urban Driving. In *Proc. of Conference on the Quantitative Evaluation of Systems QEST*, volume 8054 of *Lecture Notes in Computer Science LNCS*, pages 322–337. 2013.

[CTB12] Y. Chen, J. Tůmová, and C. Belta. LTL Robot Motion Control based on Automata Learning of Environmental dynamics. In *Proc. of IEEE Conference on Robotics and Automation ICRA*, pages 5177–5182, 2012.

[CY95] C. Courcoubetis and M. Yannakakis. The complexity of probabilistic verification. *Journal of the ACM*, 42(4):857–907, 1995.

[dA97] L. de Alfaro. *Formal Verification of Probabilistic Systems*. PhD thesis, Stanford University, 1997.

[DBC10] X.C. Ding, C. Belta, and C.G. Cassandras. Receding Horizon Surveillance with Temporal Logic Specifications. In *Proc. of IEEE Conference on Decision and Control CDC*, pages 256–261, 2010.

[DLB12] X.C. Ding, M. Lazar, and C. Belta. Receding Horizon Temporal Logic Control for Finite Deterministic Systems. In *Proc. of American Control Conference ACC*, pages 715–720, 2012.

[DSBR11] X.C. Ding, S.L. Smith., C. Belta, and D. Rus. MDP Optimal Control under Temporal Logic Constraints. In *Proc. of IEEE Conference on Decision and Control and European Control Conference CDC/ECC*, pages 532 –538, 2011.

[EH86] E.A. Emerson and J.Y. Halpern. "Sometimes" and "Not Never" Revisited: On Branching Versus Linear Time Temporal Logic. *Journal of the ACM*, 33(1):151–178, 1986.

[FGKP09] G.E. Fainekos, A. Girard, H. Kress-Gazit, and G.J. Pappas. Temporal logic motion planning for dynamic robots. *Automatica*, 45(2):343–352, 2009.

[FJK10] C. Finucane, G. Jing, and H. Kress-Gazit. LTLMoP: Experimenting with language, Temporal Logic and robot control. In *Proc. of IEEE/RSJ Conference on Intelligent Robots and Systems IROS*, pages 1988–1993, 2010.

[GDLB12] E.A. Gol, X. Ding, M. Lazar, and C. Belta. Finite Bisimulations for Switched Linear Systems. In *Proc. of IEEE Conference on Decision and Control CDC*, pages 7632–7637, 2012.

[Gir12] A. Girard. Controller synthesis for safety and reachability via approximate bisimulation. *Automatica*, 48(5):947 – 953, 2012.

[GLB12] E.A. Gol, M. Lazar, and C. Belta. Language-guided controller synthesis for discrete-time linear systems. In *Proc. of Conference on Hybrid Systems: Computation and Control HSCC*, pages 95–104, 2012.

[GN00] E.R. Gansner and S.C. North. An open graph visualization system and its applications to software engineering. *Software - Practice and Experience*, 30(11):1203–1233, 2000.

[GO01a] P. Gastin and D. Oddoux. Fast LTL to Büchi Automata Translation. In *Proc. of Conference on Computer-Aided Verification CAV*, pages 53–65, 2001.

[GO01b] P. Gastin and D. Oddoux. LTL2BA : fast translation from LTL formulae to Büchi automata, 2001. `http://www.lsv.ens-cachan.fr/ gastin/ltl2ba/`.

[GTW02] E. Grädel, W. Thomas, and T. Wilke. *Automata, Logics, and Infinite Games: A Guide to Current Research*, volume 2500 of *Lecture Notes in Computer Science LNCS*. 2002.

[HJM03] T.A. Henzinger, R. Jhala, and R. Majumdar. Counterexample-guided control. In *Proc. of Colloquium on Automata, Languages, and Programming ICALP*, 2003.

[HNP+11] E.M. Hahn, G. Norman, D. Parker, B. Wachter, and L. Zhang. Game-based Abstraction and Controller Synthesis for Probabilistic Hybrid Systems. In *Proc. of Conference on the Quantitative Evaluation of Systems QEST*, pages 69–78, 2011.

[JGP06] A.A. Julius, A. Girard, and G.J. Pappas. Approximate bisimulation for a class of stochastic hybrid systems. In *Proc. of American Control Conference ACC*, pages 4724–4729, 2006.

[Kar78] R.M. Karp. A characterization of the minimum cycle mean in a digraph. *Discrete Mathematics*, 23(3):309–311, 1978.

[KB06] J. Klein and C. Baier. Experiments with deterministic $\omega$-automata for formulas of linear temporal logic. *Theoretical Computer Science*, 363(2):182 – 195, 2006.

[KB08] M. Kloetzer and C. Belta. A Fully Automated Framework for Control of Linear Systems from Temporal Logic Specifications. *IEEE Transactions on Automatic Control*, 53(1):287–297, 2008.

[KF11] S. Karaman and E. Frazzoli. Sampling-based algorithms for optimal motion planning. *International Journal of Robotics Research*, 30(7):846–894, June 2011.

[KGFP09] H. Kress-Gazit, G.E. Fainekos, and G.J. Pappas. Temporal-Logic-Based Reactive Mission and Motion Planning. *IEEE Transactions on Robotics*, 25(6):1370–1381, 2009.

[KKNP10] M. Kattenbelt, M. Kwiatkowska, G. Norman, and D. Parker. A game-based abstraction-refinement framework for Markov decision processes. *Formal Methods in System Design*, 36(3):246–280, 2010.

[Kle07] J. Klein. ltl2dstar: LTL to Deterministic Streett and Rabin Automata, 2007. `http://www.ltl2dstar.de/`.

[Koz83] D. Kozen. Results on the propositional $\mu$-calculus. *Theoretical Computer Science*, 27(3):333 – 354, 1983.

[LAB12] M. Lahijanian, S.B. Andersson, and C. Belta. Approximate Markovian abstractions for linear stochastic systems. In *Proc. of IEEE Conference on Decision and Control CDC*, pages 5966–5971, 2012.

[LaV06] S.M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006.

[LP85] O. Lichtenstein and A. Pnueli. Checking That Finite State Concurrent Programs Satisfy Their Linear Specification. In *Proc. of ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, pages 97–107, 1985.

[McM93] K.L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, 1993.

[NO14] P. Nilsson and N. Ozay. Incremental synthesis of switching protocols via abstraction refinement. In *Proc. of IEEE Conference on Decision and Control CDC*, pages 6246–6253, 2014.

[Nor98] J.R. Norris. *Markov chains*. Cambridge University Press, 1998.

[Pel93] D. Peled. All from One, One for All: On Model Checking Using Representatives. In *Proc. of Conference on Computer-Aided Verification CAV*, pages 409–423, 1993.

[PGT08] G. Pola, A. Girard, and P. Tabuada. Approximately bisimilar symbolic models for nonlinear control systems. *Automatica*, 44(10):2508–2516, 2008.

[Pnu77] A. Pnueli. The Temporal Logic of Programs. In *Proc. of IEEE Symposium on Foundations of Computer Science FOCS*, pages 46–57, 1977.

[PPS06] N. Piterman, A. Pnueli, and Y. Sa'ar. Synthesis of Reactive(1) Designs. In *Proc. of Verification, Model Checking, and Abstract Interpretation VMCAI*, volume 3855 of *Lecture Notes in Computer Science LNCS*, pages 364–380. 2006.

[PR89] A. Pnueli and R. Rosner. On the Synthesis of a Reactive Module. pages 179–190, 1989.

[QS82] J.P. Queille and J. Sifakis. Specification and verification of concurrent systems in CESAR. In *Proc. of International Symposium on Programming*, volume 137 of *Lecture Notes in Computer Science LNCS*, pages 337–351. 1982.

[Rei11] G. Reissig. Computing Abstractions of Nonlinear Systems. *IEEE Transactions on Automatic Control*, 56(11):2583–2598, 2011.

[RKG13] V. Raman and H. Kress-Gazit. Explaining Impossible High-Level Robot Behaviors. *IEEE Transactions on Robotics*, 29(1):94–104, 2013.

[RM09] J.B. Rawlings and D.Q. Mayne. *Model Predictive Control: Theory and Design*. Nob Hill Pub., 2009.

[RV01] A. Robinson and A. Voronkov, editors. *Handbook of Automated Reasoning Volume I & II*. Elsevier Science Inc. and MIT Press, 2001.

[Saf88] S. Safra. On the Complexity of Omega-automata. In *Proc. of Annual Symposium on Foundations of Computer Science SFCS*, pages 319–327, 1988.

[SB00] F. Somenzi and R. Bloem. Efficient Büchi Automata from LTL Formulae. In *Proc. of Conference on Computer-Aided Verification CAV*, pages 248–263, 2000.

[SCL+15] M. Svoreňová, M. Chmelík, K. Leahy, H.F. Eniser, K. Chatterjee, I. Černá, and C. Belta. Temporal Logic Motion Planning using POMDPs with Parity Objectives. In *Proc. of Conference on Hybrid Systems: Computation and Control HSCC*, pages 233–238, 2015.

[SKC+15] M. Svoreňová, J. Křetínský, M. Chmelík, K. Chatterjee, I. Černá, and C. Belta. Temporal Logic Control for Stochastic Linear Systems using Abstraction Refinement of Probabilistic Games. In *Proc. of Conference on Hybrid Systems: Computation and Control HSCC*, pages 259–268, 2015.

[SMv13] M. Svoreňová, Lukáš Mařica, and I. Černá. ConTool: Tool for Simulation of Control Algorithms, 2013. `http://www.fi.muni.cz/∼x175388/contool.html`.

[STBR11] S.L. Smith, J. Tůmová, C. Belta, and D. Rus. Optimal path planning for surveillance with temporal-logic constraints. *International Journal of Robotics Research*, 30(14):1695–1708, 2011.

[STBv12] M. Svoreňová, J. Tůmová, J. Barnat, and I. Černá. Attraction-based Receding Horizon Path Planning with Temporal Logic Constraints. In *Proc. of IEEE Conference on Decision and Control CDC*, pages 6749–6754, 2012.

[SvB13a] M. Svoreňová, I. Černá, and C. Belta. Optimal Control of MDPs with Temporal Logic Constraints. In *Proc. of IEEE Conference on Decision and Control CDC*, pages 3938–3943, 2013.

[SvB13b] M. Svoreňová, I. Černá, and C. Belta. Optimal Receding Horizon Control for Finite Deterministic Systems with Temporal Logic Constraints. In *Proc. of American Control Conference ACC*, pages 4399–4404, 2013.

[SvB15] M. Svoreňová, I. Černá, and C. Belta. Optimal Temporal Logic Control for Deterministic Transition Systems with Probabilistic Penalties. *IEEE Transactions on Automatic Control*, 60(6):1–14, 2015.

[Sze10] C. Szepesvari. *Algorithms for Reinforcement Learning.* Morgan & Claypool, 2010.

[Tar72] R. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1972.

[TCK+13] J. Tůmová, L.I.R. Castro, S. Karaman, E. Frazzoli, and D. Rus. Minimum-violation LTL planning with conflicting specifications. In *Proc. of American Control Conference ACC*, pages 200–205, 2013.

[TI12] Y. Tazaki and J. Imura. Discrete Abstractions of Nonlinear Systems Based on Error Propagation Analysis. *IEEE Transactions on Automatic Control*, 57(3):550–564, 2012.

[TMDK14] J. Tůmová, A. Marzinotto, D.V. Dimarogonas, and D. Kragic. Maximally satisfying LTL action planning. In *Proc. of IEEE/RSJ Conference on Intelligent Robots and Systems IROS*, pages 1503–1510, 2014.

[TP06] P. Tabuada and G.J. Pappas. Linear Time Logic Control of Discrete-Time Linear Systems. *IEEE Transactions on Automatic Control*, 51(12):1862–1877, 2006.

[VW94] M.Y. Vardi and P. Wolper. Reasoning about Infinite Computations. *Information and Computation*, 115(1):1–37, 1994.

[Won10] T. Wongpiromsarn. *Formal Methods for Design and Verification of Embedded Control Systems: Application to an Autonomous Vehicle*. PhD thesis, California Institute of Technology, 2010.

[WTM12a] E.M. Wolff, U. Topcu, and R.M. Murray. Optimal Control with Weighted Average Costs and Temporal Logic Specifications. In *Proc. of Robotics: Science and Systems Conference*, 2012.

[WTM12b] E.M. Wolff, U. Topcu, and R.M. Murray. Robust control of uncertain markov decision processes with temporal logic specifications. In *Proc. of IEEE Conference on Decision and Control CDC*, pages 3372–3379, 2012.

[WTM12c] T. Wongpiromsarn, U. Topcu, and R.M. Murray. Receding Horizon Temporal Logic Planning. *IEEE Transactions on Automatic Control*, 57(11):2817–2830, 2012.

[WTO+11] T. Wongpiromsarn, U. Topcu, N. Ozay, H. Xu, and R. M. Murray. TuLiP: a Software Toolbox for Receding Horizon Temporal Logic Planning. In *Proc. of Conference on Hybrid Systems: Computation and Control HSCC*, pages 313–314, 2011.

[YTv+12] B. Yordanov, J. Tůmová, I. Černá, J. Barnat, and C. Belta. Temporal Logic Control of Discrete-Time Piecewise Affine Systems. *IEEE Transactions on Automatic Control*, 57:1491–1504, 2012.

[ZPMT12] M. Zamani, G. Pola, M. Mazo, and P. Tabuada. Symbolic Models for Nonlinear Control Systems Without Stability Assumptions. *IEEE Transactions on Automatic Control*, 57(7):1804–1809, 2012.

# Appendix A

# Author's Contributions

## Journal Papers

[SvB15] M. Svoreňová, I. Černá, and C. Belta. Optimal Temporal Logic Control for Deterministic Transition Systems with Probabilistic Penalties. *IEEE Transactions on Automatic Control*, 60(6):1–14, 2015.

*Leading author. Major contribution to:* problem formulation, problem solution, experimental evaluation, writing (Sec. 1-6). *Minor contribution to:* implementation.

## Conference and Workshop Proceedings

[SKC+15] M. Svoreňová, J. Křetínský, M. Chmelík, K. Chatterjee, I. Černá, and C. Belta. Temporal Logic Control for Stochastic Linear Systems using Abstraction Refinement of Probabilistic Games. In *Proc. of Conference on Hybrid Systems: Computation and Control HSCC*, pages 259–268, 2015. **Best student paper finalist.**

*Leading author. Major contribution to:* problem formulation, problem solution, implementation, experimental evaluation, writing (Sec. 2-6). *Minor contribution to:* writing (Sec. 1).

[SCL+15] M. Svoreňová, M. Chmelík, K. Leahy, H.F. Eniser, K. Chatterjee, I. Černá, and C. Belta. Temporal Logic Motion Planning using POMDPs with Parity Objectives. In *Proc. of Conference on Hybrid Systems: Computation and Control HSCC*, pages 233–238, 2015.

*Leading author. Major contribution to:* problem formulation, problem solution, experimental evaluation, writing (Sec. 2, 4, 5). *Minor contribution to:* implementation, writing (Sec. 1, 3).

[SvB13a] M. Svoreňová, I. Černá, and C. Belta. Optimal Control of MDPs with Temporal Logic Constraints. In *Proc. of IEEE Conference on Decision and Control CDC*, pages 3938–3943, 2013.

*Leading author. Major contribution to:* problem formulation, problem solution, implementation, experimental evaluation, writing (Sec. 2-6). *Minor contribution to:* writing (Sec. 1).

[SvB13b] M. Svoreňová, I. Černá, and C. Belta. Optimal Receding Horizon Control for Finite Deterministic Systems with Temporal Logic Constraints. In *Proc. of American Control Conference ACC*, pages 4399–4404, 2013.

*Leading author. Major contribution to:* problem formulation, problem solution, implementation, experimental evaluation, writing (Sec. 2-6). *Minor contribution to:* writing (Sec. 1).

[STBv12] M. Svoreňová, J. Tůmová, J. Barnat, and I. Černá. Attraction-based Receding Horizon Path Planning with Temporal Logic Constraints. In *Proc. of IEEE Conference on Decision and Control CDC*, pages 6749–6754, 2012.

*Major contribution to:* problem solution, implementation, experimental evaluation, writing (Sec. 2, 4-6). *Minor contribution to:* problem formulation, writing (Sec. 1, 3).

# Appendix B

# Solving a $2^1/_2$-player Game

Here we present an algorithm to solve the almost-sure winning problem for a $2^1/_2$-player game $\mathcal{G} = (S_1 \cup S_2, Act, P, AP, L)$ with a Büchi implication condition $(E, F)$, where $E, F \subseteq S$. The optimal solution is a rather involved, quadratic time algorithm that can be found in [Cha07]. In this work, we use a more intuitive, cubic time algorithm presented in Algorithm 7, whose correctness follows from [CdAH11]. The algorithm is a simple iterative fixed-point algorithm that uses three types of local predecessor operator over the set of states of the game.

Consider sets $X, Y, Z$ such that $Y \subseteq Z \subseteq X \subseteq S$. Given a state $s \in S$ and an action $\alpha \in Act$, we denote by $\mathrm{Succ}(s, \alpha) = \mathrm{Supp}(\delta(s, \alpha))$ the set of possible successors of the state and the action. We define conditions on state action pairs as follows:

$$
\begin{aligned}
\mathsf{C}_1(X) &= \{(s,\alpha) \mid \mathrm{Succ}(s,\alpha) \subseteq X\},\\
\mathsf{C}_2(X,Y) &= \{(s,\alpha) \mid \mathrm{Succ}(s,\alpha) \subseteq X \text{ and } \mathrm{Succ}(s,\alpha) \cap Y \neq \emptyset\},\\
\mathsf{C}_3(Z,X,Y) &= \{(s,\alpha) \mid (\mathrm{Succ}(s,\alpha) \subseteq Z) \text{ or }\\
&\qquad\qquad (\mathrm{Succ}(s,\alpha) \subseteq X \text{ and } \mathrm{Succ}(s,\alpha) \cap Y \neq \emptyset)\}.
\end{aligned}
$$

The first condition ensures that given the state and action the next state is in $U$ with probability 1, the second condition ensures that the next state is in $X$ with probability 1 and in $Y$ with positive probability. The third condition is the disjunction of the first two. The three predecessor operators are defined as the set of Player 1, or Player 2 states, where there exists, or for all, respectively, actions, the condition for the predecessor operator is satisfied. The three respective predecessor operators, namely, $\mathsf{Pre}_1, \mathsf{Pre}_2$, and $\mathsf{Pre}_3$ are defined as follows:

**Algorithm 7** Solving $2^1/_2$-player game with a Büchi implication condition

Input: game $\mathcal{G}$, Büchi implication condition $(E, F)$
Output: $\mathsf{Almost}^{\mathcal{G}}((E, F))$
$D = S \setminus (E \cup F)$
$\overline{X} \leftarrow S; \quad \overline{Z} \leftarrow S; \quad \overline{Y} \leftarrow \emptyset;$
**do**
  $X \leftarrow \overline{X}$
  **do**
    $Y \leftarrow \overline{Y}$
    **do**
      $Z \leftarrow \overline{Z}$
      $\overline{Z} \leftarrow (F \cap \mathsf{Pre}_1(X)) \cup (E \cap \mathsf{Pre}_2(X, Y)) \cup (D \cap \mathsf{Pre}_3(Z, X, Y)$
    **while** $Z \neq \overline{Z}$
    $\overline{Y} \leftarrow Z$
    $\overline{Z} \leftarrow S$
  **while** $Y \neq \overline{Y}$
  $\overline{X} \leftarrow Y$
  $\overline{Y} \leftarrow \emptyset$
**while** $X \neq \overline{X}$
return $X$

$$
\begin{aligned}
\mathsf{Pre}_1(X) =& \{s \in S_1 \mid \exists \alpha \in Act.\ (s, \alpha) \in \mathsf{C}_1(X)\} \cup \\
& \{s \in S_2 \mid \forall \alpha \in Act.\ (s, \alpha) \in \mathsf{C}_1(X)\}, \\
\mathsf{Pre}_2(X, Y) =& \{s \in S_1 \mid \exists \alpha \in Act.\ (s, \alpha) \in \mathsf{C}_2(X, Y)\} \cup \\
& \{s \in S_2 \mid \forall \alpha \in Act.\ (s, \alpha) \in \mathsf{C}_2(X, Y)\}, \\
\mathsf{Pre}_3(Z, X, Y) =& \{s \in S_1 \mid \exists \alpha \in Act.\ (s, \alpha) \in \mathsf{C}_3(Z, X, Y)\} \cup \\
& \{s \in S_2 \mid \forall \alpha \in Act.\ (s, \alpha) \in \mathsf{C}_3(Z, X, Y)\}.
\end{aligned}
$$

# Appendix C

# Polytopic Operators

In this section, we describe in detail the computation of all polytopic operators introduced in Section 4.1.3.

## C.1 Action Polytopes

First, we describe how to compute the action polytopes $\mathcal{U}_i^J$ for every polytope $\mathcal{X}_i \in \{\mathcal{X}_i\}_{i\in I}$, formally defined in Equation 4.1.

For a polytope $\mathcal{X}' \subset \mathbb{R}^N$, we use $\mathcal{U}^{\mathcal{X}_i \to \mathcal{X}'}$ to denote the set of all control inputs from $\mathcal{U}$ under which the system $\mathcal{L}$ can evolve from a state in $\mathcal{X}_i$ to a state in $\mathcal{X}'$ with non-zero probability, i.e.,

$$\mathcal{U}^{\mathcal{X}_i \to \mathcal{X}'} = \{u \in \mathcal{U} \mid \text{Post}(\mathcal{X}_i, u) \cap \mathcal{X}' \text{ is non-empty}\}. \tag{C.1}$$

The following proposition states that $\mathcal{U}^{\mathcal{X}_i \to \mathcal{X}'}$ can be computed from the V-representations of $\mathcal{X}_i, \mathcal{X}'$ and $\mathcal{W}$.

**Proposition 12.** Let $H, K$ be the matrices from the H-representation of the following polytope:

$$\{y \in \mathbb{R}^N \mid \exists x \in \mathcal{X}_i, \exists w \in \mathcal{W} :\ Ax + y + w \in \mathcal{X}'\}, \tag{C.2}$$

which can be computed as the convex hull

$$\text{hull}(\{v_{\mathcal{X}'} - (Av_{\mathcal{X}_i} + v_{\mathcal{W}}) \mid v_{\mathcal{X}'} \in V(\mathcal{X}'), v_{\mathcal{X}_i} \in V(\mathcal{X}_i), v_{\mathcal{W}} \in V(\mathcal{W})\}). \tag{C.3}$$

Then the set $\mathcal{U}^{\mathcal{X}_i \to \mathcal{X}'}$ defined in Equation C.1 is the polytope with the following H-representation:

$$\mathcal{U}^{\mathcal{X}_i \to \mathcal{X}'} = \{u \in \mathcal{U} \mid HBu \leq K\}. \tag{C.4}$$

*Proof.* To fact that the set in Equation C.2 is a polytope with the V-representation given in Equation C.3 can be easily shown as follows. Let $y \in \mathbb{R}^N$ be such that there exist $x \in \mathcal{X}_i, w \in \mathcal{W}, x' \in \mathcal{X}'$ for which $Ax+y+w = x'$, i.e., $y = x'-(Ax+w)$.

By representing $x', x$ and $w$ as an affine combination of the respective vertices in $V(\mathcal{X}'), V(\mathcal{X}_i)$ and $V(\mathcal{W})$, we obtain the V-representation in Equation C.3. Next, let $H, K$ be the matrices from the H-representation of the set in Equation C.2. Then the definition of set $\mathcal{U}^{\mathcal{X}_i \to \mathcal{X}'}$ in Equation C.1 can be written as

$$\mathcal{U}^{\mathcal{X}_i \to \mathcal{X}'} = \{u \in \mathcal{U} \mid \exists x \in \mathcal{X}_i, \exists w \in \mathcal{W} : Ax + Bu + w \in \mathcal{X}'\},$$

that leads to H-representation in Equation C.4. □

**Corollary 2.** Let $J \subseteq I \cup I_{\mathsf{out}}$. The set $\mathcal{U}_i^J$ from Equation 4.1 can be computed as follows:

$$\mathcal{U}_i^J = \bigcap_{j \in J} \mathcal{U}^{\mathcal{X}_i \to \mathcal{X}_j} \setminus \bigcup_{j' \notin J} \mathcal{U}^{\mathcal{X}_i \to \mathcal{X}_{j'}}. \tag{C.5}$$

*Proof.* Follows directly from Equations 4.1 and C.1. □

Note that $\mathcal{U}_i^J$ is generally not a polytope but can be represented as a finite union of polytopes.

## C.2 Posterior

The posterior operator $\mathrm{Post}(\mathcal{X}', \mathcal{U}')$, formally defined in Table 4.1, can be easily computed using Minkowski sum as

$$\begin{aligned}\mathrm{Post}(\mathcal{X}'\mathcal{U}') &= A\mathcal{X}' + \mathcal{U}' + \mathcal{W} \\ &= \mathrm{hull}(\{Av_{\mathcal{X}'} + Bv_{\mathcal{U}'} + v_{\mathcal{W}} \mid v_{\mathcal{X}'} \in V(\mathcal{X}'), v_{\mathcal{U}'} \in V(\mathcal{U}'), v_{\mathcal{W}} \in V(\mathcal{W})\}).\end{aligned}$$

## C.3 Predecessor

The predecessor operator $\mathrm{Pre}(\mathcal{X}', \mathcal{U}', \{\mathcal{X}_j\}_{j \in J})$, formally defined in Table 4.1, can be computed as follows. First, note that

$$\mathrm{Pre}(\mathcal{X}', \mathcal{U}', \{\mathcal{X}_j\}_{j \in J}) = \bigcup_{j \in J} \mathrm{Pre}(\mathcal{X}', \mathcal{U}', \mathcal{X}_j).$$

**Proposition 13.** Let $H, K$ be the matrices from the H-representation of the following polytope:

$$\{y \in \mathbb{R}^N \mid \exists u \in \mathcal{U}', \exists w \in \mathcal{W} : \; y + Bu + w \in \mathcal{X}_j\},$$

which can be computed as the convex hull

$$\mathrm{hull}(\{v_{\mathcal{X}_j} - (Bv_{\mathcal{U}'} + v_{\mathcal{W}}) \mid v_{\mathcal{X}_j} \in V(\mathcal{X}_j), v_{\mathcal{U}'} \in V(\mathcal{U}'), v_{\mathcal{W}} \in V(\mathcal{W})\}).$$

Then the set $\mathrm{Pre}(\mathcal{X}', \mathcal{U}', \mathcal{X}_j)$ is the polytope with the following H-representation:

$$\mathrm{Pre}(\mathcal{X}', \mathcal{U}', \mathcal{X}_j) = \{x \in \mathcal{X}' \mid HAx \leq K\}.$$

*Proof.* The proof is analogous to the one of Proposition 12. □

## C.4 Robust and Precise Predecessor

From definitions of the robust and precise predecessor operators in Table 4.1 it follows that

$$\mathrm{PreR}(\mathcal{X}',\mathcal{U}',\{\mathcal{X}_j\}_{j\in J}) = \bigcup_{J'\subseteq J, J'\neq\emptyset} \mathrm{PreP}(\mathcal{X}',\mathcal{U}',\{\mathcal{X}_j\}_{j\in J'}).$$

Below we describe computation of the precise predecessor $\mathrm{PreP}(\mathcal{X}',\mathcal{U}',\{\mathcal{X}_j\}_{j\in J'})$ for any $J' \subseteq J$.

Let $\mathcal{Z}$ denote the polytope, or finite union of polytopes, $\mathcal{Z} = A\mathcal{X}' + B\mathcal{U}'$, where $+$ denotes the Minkowski sum. For a polytope $\mathcal{P} \subset \mathbb{R}^N$, we define set

$$\mathcal{Z}(\mathcal{P}) = \{z \in \mathcal{Z} \mid (z + \mathcal{W}) \cap \mathcal{P} \text{ is non-empty}\}. \tag{C.6}$$

For a set of polytopes $\{\mathcal{P}\}$, $\mathcal{Z}(\{\mathcal{P}\})$ can be computed as the union of all $\mathcal{Z}(\mathcal{P})$ for every polytope $\mathcal{P}$ in the set $\{\mathcal{P}\}$.

**Proposition 14.** The set from Equation (C.6) is the following polytope, or finite union of polytopes:

$$\mathcal{Z}(\mathcal{P}) = hull(\{v_\mathcal{P} - v_\mathcal{W} \mid v_\mathcal{P} \in V(\mathcal{P}), v_\mathcal{W} \in V(\mathcal{W})\}) \cap \mathcal{Z}. \tag{C.7}$$

*Proof.* The proof is carried out in a similar way as the first part of proof of Proposition 12. □

For $J' \subseteq J$, we use $\mathcal{Z}(J')$ to denote the set

$$\mathcal{Z}(J') = \bigcap_{j\in J'} \mathcal{Z}(\mathcal{X}_j) \backslash \Big( \bigcup_{j\in J\backslash J'} \mathcal{Z}(\mathcal{X}_j) \cup \mathcal{Z}(\mathcal{X}_{\neg J})\Big), \tag{C.8}$$

where $\mathcal{Z}(\mathcal{X}_{\neg J}) = \mathcal{Z}((\mathcal{X} \cup \mathcal{X}_{\mathsf{out}}) \backslash \bigcup_{j\in J} \mathcal{X}_j)$.

**Proposition 15.** Let $\mathcal{U}' = \{\mathcal{U}_{l_1}\}_{l_1\in L_1}$, $J \subseteq J'$ and let $\mathcal{Z}(J') = \{\mathcal{Z}_{l_2}\}_{l_2\in L_2}$. Then the precise predecessor can be written as

$$\mathrm{PreP}(\mathcal{X}',\mathcal{U}',\{\mathcal{X}_j\}_{j\in J'}) = \bigcup_{l_1\in L_1} \bigcup_{l_2\in L_2} \{x \in \mathcal{X}' \mid \exists u \in \mathcal{U}_{l_1} : Ax + Bu \in \mathcal{Z}_{l_2}\}. \tag{C.9}$$

Let $l_1 \in L_1, l_2 \in L_2$ and let $H, K$ be the matrices from the H-representation of the following polytope:

$$\{y \in \mathbb{R}^N \mid \exists u \in \mathcal{U}_{l_1} : y + Bu \in \mathcal{Z}_{l_2}\}, \tag{C.10}$$

which can be computed as the convex hull

$$\mathrm{hull}(\{v_{\mathcal{Z}_{l_2}} - Bv_{\mathcal{U}_{l_1}} \mid v_{\mathcal{Z}_{l_2}} \in V(\mathcal{Z}_{l_2}), v_{\mathcal{U}_{l_1}} \in V(\mathcal{U}_{l_1})\}). \tag{C.11}$$

Then the set on the right-hand site of Equation C.9, for $l_1, l_2$, is a polytope with the following H-representation:

$$\{x \in \mathcal{X}' \mid HAx \leq K\}. \tag{C.12}$$

*Proof.* From the definition of the set $\mathcal{Z}(J')$ in Equation C.8, $z \in \mathcal{Z}(J')$ iff $z + \mathcal{W}$ intersects all $\mathcal{X}_j$ for $j \in J'$ and $z + \mathcal{W} \subseteq \bigcup_{j \in J'} \mathcal{X}_j$. Moreover, every $z \in \mathcal{Z}$ can be written as $z = Ax + Bu$ and therefore $z + \mathcal{W} = \text{Post}(x, u)$. This proves Equation C.9. The rest of the proof is carried out in a way similar to the proof of Proposition 12. □

## C.5 Attractor

The attractor operator $\text{Attr}(\mathcal{X}', \mathcal{U}', \{\mathcal{X}_j\}_{j \in J})$ from Table 4.1 can be computed using the robust predecessor operator, since it holds that

$$\begin{aligned}
\text{Attr}(\mathcal{X}', \mathcal{U}', \{\mathcal{X}_j\}_{j \in J}) &= \{x \in \mathcal{X}' \mid \forall u \in \mathcal{U}' : \text{Post}(x, u) \cap \bigcup_{j \in J} \mathcal{X}_j \text{ is non-empty}\} \\
&= \mathcal{X}' \backslash \{x \in \mathcal{X}' \mid \exists u \in \mathcal{U}' : \text{Post}(x, u) \subseteq (\mathcal{X} \cup \mathcal{X}_{\mathsf{out}}) \backslash \bigcup_{j \in J} \mathcal{X}_j\} \\
&= \mathcal{X}' \backslash \text{PreR}(\mathcal{X}', \mathcal{U}', (\mathcal{X} \cup \mathcal{X}_{\mathsf{out}}) \backslash \bigcup_{j \in J} \mathcal{X}_j).
\end{aligned}$$

## C.6 Robust Attractor

The robust attractor operator $\text{AttrR}(\mathcal{X}', \mathcal{U}', \{\mathcal{X}_j\}_{j \in J})$ from Table 4.1 can be computed using the predecessor operator, since it holds that

$$\begin{aligned}
&\text{AttrR}(\mathcal{X}', \mathcal{U}', \{\mathcal{X}_j\}_{j \in J}) \\
&\quad = \{x \in \mathcal{X}' \mid \forall u \in \mathcal{U}' : \text{Post}(x, u) \subseteq \bigcup_{j \in J} \mathcal{X}_j\} \\
&\quad = \mathcal{X}' \backslash \{x \in \mathcal{X}' \mid \exists u \in \mathcal{U}' : \text{Post}(x, u) \cap (\mathcal{X} \cup \mathcal{X}_{\mathsf{out}}) \backslash \bigcup_{j \in J} \mathcal{X}_j \text{ is non-empty}\} \\
&\quad = \mathcal{X}' \backslash \text{Pre}(\mathcal{X}', \mathcal{U}', (\mathcal{X} \cup \mathcal{X}_{\mathsf{out}}) \backslash \bigcup_{j \in J} \mathcal{X}_j).
\end{aligned}$$