Masaryk University
Faculty of Informatics

Quantitative Formal Methods for High-level Robot Path Planning

PhD Thesis

Jana Tůmová

Brno, Spring 2013

Declaration

Hereby I declare, that this thesis is my original authorial work, which I have worked out by my own. All sources, references and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Advisor: prof. RNDr. Ivana Černá, CSc.

Acknowledgement

There is a number of people without whom this thesis would not happen. I am deeply grateful to my advisor Ivana Černá for all of her guidance, support, valuable advice and the great amount of knowledge she has passed on me. Most of all, I appreciate her friendly and warm attitude. She has been not only an academic advisor, but also my mentor, always willing to listen about both my research and life struggles.

My many thanks go to Jiří Barnat who has been my consultant. He has been a constant source of novel ideas and I will never forget the passionate discussions we have had. I would also like to express my sincere gratitude to all of my collaborators for the opportunity to work with them and to learn from them. Especially, I would like to thank Calin Belta from Boston University, and Emilio Frazzoli and Daniela Rus from Massachusetts Institute of Technology, who have hosted me as a visitor in their groups and shared with me their remarkable insight into control theory and robotics. My special thanks go to Sertac Karaman, for his encouragement and many inspiring and entertaining discussions.

It has been my privilege to be a part of ParaDiSe lab as well as HyNeSs lab at Boston University, whose members have created a stimulating work environment. My friends Maja Svoreňová and Milan Češka made my experience especially pleasant. I thank them and also Dalibor Klusáček and Nikola Beneš for their moral support and their numerous suggestions that helped to improve this thesis.

Last, but not least, I would like to thank my whole family and friends for making my life fulfilled, my parents for their unconditional support and unlimited patience, and Jakub for always being on my side.

Abstract

This thesis addresses selected theoretical and applied topics in quantitative model checking and control strategy synthesis. First, we develop a theoretical framework for *quantitative model checking of systems with degradation,* which are systems that naturally incorporate a decaying quality, such as electronic devices with degrading electric charge, broadcasting networks with decreasing power or quality of a transmitted signal. We propose transition systems with degradation as a modeling formalism and linear temporal logic with degradation constraints as a suitable specification language. Two different approaches to model checking and control strategy synthesis are proposed. The first one is a limited, yet arbitrary precision method that exploits a resemblance between the systems with degradation and timed automata. The second one is a full precision method that directly builds on principles of automata-based approach. We also investigate the relationship between systems with degradation and Markov decision processes and report on the incomparable expressive power of the suggested temporal logic with respect to the commonly used probabilistic logics.

Second, in the latter part of the thesis we focus on *automatic optimal control strategy synthesis specifically tailored for mobile robotic systems.* In particular, we consider high-level planning, when a robot's motion capabilities are modeled as a discrete transition system and its desired task is specified as a linear temporal logic formula. We address cases when multiple plans meet the specification and we introduce an algorithm to find an optimal plan with respect to a carefully chosen cost function. In contrast, we also aim at instances when there is no control strategy ensuring the satisfaction of the desired task as a whole. For such cases, we propose methods to find a least-violating plan. We address separately tasks in the form of long-term missions to be accomplished and safety rules to be obeyed. We demonstrate our results in illustrative case studies.

Contents

Chapter 1

Introduction

1.1 Motivation

Computer-based systems have become an inseparable part of today's world and our everyday lives. They have become more and more complex and their utilization has become more and more common. Consider, for instance, autonomous mobile robots. Robotic vacuum cleaners have occurred as common helpers in many households. The employment of robotic units in warehouses, brought, *e.g.,* by KIVA Systems [kiv13], has provided an innovative alternative to traditional storage. Instead of a human operator traveling around a warehouse to pick up goods from static shelves, the operator stays in a work station and an autonomous robot brings the goods to her. Lately, significant attempts have been made towards enabling operation of driverless cars in real traffic and many of them have been successful. Namely, in DARPA 2007 Urban Challenge, teams of researchers, mostly from academia, competed in designing an entirely autonomous car. Six vehicles finished the final race, driving approximately 89 kilometers in traffic each [dar13]. Another example comes from a private company. As of August 2012, the Google driverless car has reported 480,000 kilometers driven without an accident in total [goo13].

As the level of the integration of hardware and software systems in our lives has been growing, our dependence on them and the degree of our trust into them has been growing, too. Failures of such systems might be insignificant, causing a minor discomfort, however, they might as well result in an enormous financial loss or even a loss of human lives. For instance, since March 2012, the use of autonomous cars is authorized in Nevada, USA. The consequences of letting an unreliable driverless vehicle freely navigate there might be catastrophic.

There is no doubt that it is desirable to prevent the presence of errors in computer-based systems and a number of methods have been developed to address this uneasy task. The very well-known simulation and testing aim to detect errors. Yet, these techniques typically cannot be used to show the absence of errors and they cannot provide any proofs of the system correctness. In the DARPA 2007 Urban Challenge, 47 out of 53 participants were not able to successfully

finish the race in spite of extensive testing. The systems were simply too complex and the designers could not think of all possible scenarios. This was a good example motivating the need of techniques that go beyond; techniques that can provide formal guarantees that a system behaves as expected; techniques that are based on mathematics and allow to rigorously model and analyze various kind of systems, commonly known as *formal methods*.

In particular, *formal verification methods* allow us to prove or disprove that a given system satisfies desired requirements. The well-known formal verification approaches include *model checking*. Loosely speaking, a model checking method involves three steps: First, a system (*e.g.,* a communication protocol) is formalized, usually as some kind of a finite state-transition model. Second, a property (*e.g.,* deadlock-freeness or eventual message delivery) is specified, often by a formula of a temporal logic. Finally, all possible behaviors of the given system are then systematically explored using an automated procedure and an answer, whether the system satisfies the specification or not, is provided.

A different point of view on guaranteeing faultlessly behaving systems is from perspective of their correct-by-construction design. Rather than checking whether a system meets desired requirements, the aim is to purposely *synthesize* the system (or its parts) in a way already ensuring their satisfaction. An interesting instance of the synthesis problem is a *control strategy synthesis* problem, derived for systems that need to be automatically controlled, such as autonomous mobile robots. Instead of a full system, we seek for a *control strategy*, (also called a controller, or a supervisor) restricting the behavior of a given system. Specifically, the problem can be described as a counterpart to model-checking problem: Given a modeled system (*e.g.,* an autonomous robot vehicle in an environment), and a desired property specification (*e.g.,* repeated pickup of a data package in a specific location and their upload in another location), the task is to automatically find a control strategy (*e.g.,* a sequence of robot motion primitives), that, when executed in the model, guarantees the satisfaction of the specification.

1.2 Focus of the Thesis

The development of new models, logics, algorithms and computer-aided techniques to deal with both verification and controller synthesis problems has been one of the research directions in the formal methods field experiencing continual advances. On one hand, the interest lies in the development of general frameworks, that would allow us to model a wide class of systems and cope with a wide class of their properties. This point of view shows and offers the possibility to use the formal methods at places where they have not been employed yet and possibly also where one has not even thought of their utilization. On

the other hand, there are very specific real-world problems where we would like to deliberately use formal methods, but none of the existing techniques fits the investigated settings. In such a case, new formal methods can be carefully tailored, possibly not matching any other scenarios than the single one. An analogy to these two approaches can be seen in the use of a general-purpose computer versus an embedded system; the first one is rather flexible to deal with a whole class of tasks, whereas the latter one is dedicated to a specific purpose.

In this thesis we follow both of these lines. We introduce a new, general-purpose formal methods framework to model and analyze robotic systems (and not only robotic ones) featuring certain quantitative aspects. In addition, we focus on the design of formal methods algorithms dealing with rather specific problems arising in autonomous mobile robot path planning.

1.2.1 Systems with Degradation

The first of our aims in this thesis is the development of a *theoretical framework for analysis of systems with degradation*, *i.e.,* systems that naturally incorporate a decaying quality, such as electronic devices with degrading electric charge, or broadcasting networks with decreasing power or quality of a transmitted signal. Such systems have one feature in common; along a run of a system, the quality degrades relatively, such as the number of remaining particles in a sample subject to radioactive decay, not absolutely, such as the remaining distance when driving a car from a start point to a target point.

When reasoning about these systems, a quite natural interest is to incorporate their characteristic feature, *i.e.,* to include constraints on the *quantity* of degradation along their runs. For instance, consider a broadcasting network with a sender and a receiver. A signal is being transmitted trough the network, and its strength degrades with the distance from the sender. As the sender and the receiver are too far away from each other, the signal cannot reach the receiver without being irreversibly damaged. Therefore, relays that fully restore the signal are built in between the sender and the receiver. A question to answer is, whether the signal reaches the receiver in a proper shape, *i.e.,* whether the signal strength is always above a certain threshold, and if so, whether all of the relays are necessary or whether some of them can be removed while preserving the signal strength within safe bounds.

For an example of control strategy synthesis problem, consider an autonomous robot that moves along an environment, collects data packages in data gather locations and brings them to data upload locations. The collected data are stored in a volatile memory and are subject to degradation. The task is to find a control strategy in the form of a sequence of robot motion primitives, such that both

the data gather and the data upload locations are periodically visited and at the same time a data upload region is always visited before the integrity of the collected data drops below a certain threshold. Another problem to address is to find an optimal control strategy that ensures a consecutive visit to a data gather location and a data upload location while minimizing the level of data degradation in between them.

Our goal in this thesis is to suggest a modeling and specification formalisms for systems with degradation and to design algorithms capable of solving the verification and control strategy synthesis problems, such as the ones described above.

1.2.2 Robotic Systems

Next to the development of a general-purpose formal methods framework for systems with degradation, the thesis aims at *control strategy synthesis problems specifically tailored for scenarios arising in robot path planning.* In many cases, there are multiple control strategies that comply with a given specification and a choice of the optimal one is desirable. Consider, for example, an autonomous car navigating in urban traffic. The car must reach its final destination while obeying the traffic rules. Naturally, the shortest path satisfying the requirements is the preferred one. In contrast, often, there is no path satisfying the given mission as a whole, *e.g.,* due to unrealizability of the specification and/or environmental constraints. In such a case, it might still be more interesting to ensure the most important pieces of the mission than to report a failure to find a control strategy. For instance, for an autonomous car and also for a human driver, the road rules include obstacle avoidance and staying in the right lane, where the first one is of a higher priority. It is quite common that, when driving towards a final destination, the least important rules are temporarily violated, such as the latter named rule is when taking over a parked car. Another example is from the popular literature. Isaac Asimov's "three laws of robotics" (see [Asi50]) define how robots shall interact with humans. According to these laws, a robot may violate any order given by a human operator, if another human life comes in danger.

This thesis aims at three different types of optimization criteria for control strategies. One is within the first mentioned direction, where we look for a strategy satisfying a given specification and optimizing a certain cost function and two are within the second direction, where we look for a least-violating strategy. The particular settings that we consider is an autonomous robot traversing regions of a partitioned environment modeled as a discrete, finite state-transition syste). For the specification of its mission, we consider *linear temporal logic* (LTL) that

has recently gained significant popularity among robotics community. There are two major reasons for that; LTL reasons about individual, infinite runs of systems and is expressive enough to cover many interesting scenarios including, *e.g.,* reachability, safety, periodicity, reactivity, sequencing, and therefore, it is suitable to capture complex robotic missions. Furthermore, it is perceived as quite a user-friendly formalism due to some resemblance to natural language.

First, motivated by problems in monitoring and data gathering, we consider a robotic vehicle moving in an environment, where regions of interest are connected with roads of different lengths. The robot is given an LTL mission and a set of so-called optimizing locations (*e.g.,* data delivery locations) to be periodically surveyed. Our aim is to minimize the maximal distance traveled in between two successive visits to the optimizing locations while satisfying the LTL mission.

Second, we focus on long-term robotic missions in the form of a set of LTL formulas, each of which is assigned a reward earned if the respective formula is satisfied. Our goal is to find a robot path maximizing the cumulative rewards earned for the accomplished. Third, we aim at robotic missions given as a "Go from A to B." task together with a set of safety rules that need to be obeyed. Unlike in the previous case, each rule is assigned a penalty that is collected for each step leading to violation of the rule. The goal is to drive the robot from A to B while collecting the least penalties possible.

1.3 Contributions Summary

1.3.1 Thesis Contribution

In Section 1.2, we have informally introduced the focus of this thesis and the investigated problems. Let us now briefly summarize our results. The full list of the author's contributions can be also found in Appendix B.

Formal Methods for Systems with Degradation

- We propose a modeling formalism called *Transition System with Degradation (TSD)* that involves a possibility to capture the level of degradation in between two system states.

- We propose an extension to linear temporal logic called *Linear Temporal Logic with Degradation Constraints (DLTL)* as a specification formalism to capture complex system properties, including requirements on the level of degradation.

- We show an analogy between the transition systems with degradation and timed automata used to model systems with timing aspects and an analogy between DLTL and metric interval time logic (MITL) used to specify timed properties of runs of timed automata. Furthermore, we show that the translation of DLTL model checking problem for systems with degradation into previously solved MITL model checking problem for timed systems is possible, and that the targeted DLTL model checking problem can be this way solved with limited, yet arbitrary, precision.

- We propose an alternative automata-like specification formalism for systems with degradation, called *Büchi Automata with Degradation Constraints* (*BADCs*) and we develop a model checking algorithm leveraging ideas from automata-based approach to model checking of "standard" transition systems with respect to "standard" Büchi automata.

- For a specific subclass of DLTL formulas called *half-bounded DLTL*, we present an algorithm for the direct translation of such DLTL formulas into Büchi automata with degradation constraints and thus, for this subclass we provide a full-precision model checking method.

- We show, that under the assumption that a given TSD is deterministic, the suggested framework can be used not only to verify systems with degradation, but also to automatically generate control strategies for them.

- We investigate the relationship between TSDs and Markov decision processes and we show that the expressive powers of DLTL and common probabilistic logics LTL, PCTL, and PCTL* are incomparable.

These results are based on the following publications:

- [BČT12b] Jiří Barnat, Ivana Černá, Jana Tůmová. Verification of Systems with Degradation. *Computing and Informatics*, 31(3):507-530, 2012.

- [BČT12a] Jiří Barnat, Ivana Černá, Jana Tůmová. Timed Automata Approach to Verification of Systems with Degradation. In *Annual Doctoral Workshop on Mathematical and Engineering Methods in Computer Science (MEMICS)*, volume LNCS 7719, pages 86-95, 2011.

- [BČT09] Jiří Barnat, Ivana Černá, Jana Tůmová. Quantitative Model Checking of Systems with Degradation. *International Conference on Quantitative Evaluation of SysTems (QEST)*, pages 21-30, 2009.

Optimal High-level Robot Path Planning

- We present an algorithm to automatically generate an optimal path for a mobile robot in an environment modeled as a weighted transition system. The path is desired to satisfy a given LTL formula containing as its part an optimizing proposition to be repeatedly reached. At the same time, the aim is to minimize the maximum time in between revisits to locations satisfying the optimizing proposition. We build on the automata-based approach to model-checking and translate the problem into looking for a certain structure of a minimum weight in a finite graph. We demonstrate our algorithm in a robotic testbed with a pickup-delivery case study.

- We suggest a technique to synthesize a control strategy for robotic systems modeled as a transition systems with a set of LTL specifications that are assigned a reward each. The strategy is optimal with respect to the sum of the rewards of the formulas that are satisfied. In our solution, we translate the LTL formulas into automata over infinite words and we capture their respective rewards through weights assigned to the transitions of the automata. Final steps of the solution again leverage ideas from the automata-based approach to model-checking. We illustrate the algorithm through a simulation of a complex rescue mission.

- We propose an algorithm to address a robot path planning problem with a given set of safety rules with assigned priorities. More precisely, we plan a path from a given initial state to a given goal state in a transition system that models the robot's motion capabilities in an environment. To identify a least-violating path, we define a *level of unsafety* of a single robot path as a number of steps that caused violation of a safety rule weighted by the respective rule priority. We propose an approach based on the construction of a weighted structure capturing the rule priorities to find the least-violating robot path leading from the initial to the goal state. We demonstrate the approach through a simulation case study of an autonomous vehicle navigating through a urban-like environment.

These results are based on the following publications:

- [TRCK+12] Jana Tůmová, Luis I. Reyes-Castro, Sertac Karaman, Emilio Frazzoli and Daniela Rus. Minimum-violating Planning with Conflicting Specifications. 2013. Submitted.

- [THK+12] Jana Tůmová, Gavin C. Hall, Sertac Karaman, Emilio Frazzoli and Daniela Rus. Least-violating Control Strategy Synthesis with Safety

Rules. In *Hybrid Systems: Computation and Control (HSCC)*, 2013. To appear.

- [STBR11] Stephen L. Smith, Jana Tůmová, Calin Belta, Daniela Rus. Optimal Path Planning for Surveillance with Temporal Logic Constraints. In *The International Journal of Robotics Research*, 30(14):1695-1708, 2011.

- [STBR10] Stephen L. Smith, Jana Tůmová, Calin Belta, Daniela Rus. Optimal Path Planning under Temporal Logic Constraints. In *IEEE/RSJ International Conference on Intelligent Robot and Systems (IROS)*, pages 3288-3293, 2010.

1.3.2 Author's Other Results

Other research results can be split into three different categories based on their topics and are listed below.

High-Level Robot Path Planning in Environments with Uncertain Elements

- [CTUB13] Yushan Chen, Jana Tůmová, Alphan Ulusoy, Calin Belta. Temporal Logic Robot Control based on Automata Learning of Environmental Dynamics. *The International Journal of Robotics Research*, 2013. In print.

- [CTB12] Yushan Chen, Jana Tůmová, Calin Belta. LTL Robot Motion Control based on Automata Learning of Environmental Dynamics. *In IEEE International Conference on Robotics and Automation (ICRA)*, pages 5177–5182, 2012.

- [STBČ12] Mária Svoreňová, Jana Tůmová, Jiří Barnat, and Ivana Černá. Attraction-Based Receding Horizon Path Planning with Temporal Logic Constraints. In *IEEE Conference on Decision and Control (CDC)*, pages 6749–6754, 2012.

LTL Control and Analysis of Piecewise-Affine Systems

- [YTČ+13] oyan Yordanov, Jana Tůmová, Ivana Černá, Jiří Barnat, Calin Belta. Formal Analysis of Piecewise Affine Systems through Formula-Guided Refinement. *Automatica*, 49(1):261 – 266, 2013.

- [YTČ+12] Boyan Yordanov, Jana Tůmová, Ivana Černá, Jiří Barnat, and Calin Belta. Temporal Logic Control of Discrete-Time Piecewise Affine

Systems. *IEEE Transactions on Automatic Control*, 57(6):1491 –1504, 2012.

- [TYB+10] Jana Tůmová, Boyan Yordanov, Calin Belta, Ivana Černá, and Jiří Barnat. A Symbolic Approach to Controlling Piecewise Affine Systems. In *IEEE Conference on Decision and Control (CDC)*, pages 4230–4235, 2010.

- [YTB+10] Boyan Yordanov, Jana Tůmová, Calin Belta, Ivana Černá, and Jiří Barnat. Formal Analysis of Piecewise Affine Systems through Formula-Guided Refinement. In *IEEE Conference on Decision and Control (CDC)*, pages 5899–5904, 2010.

LTL Verification of Markov Decision Processes

- [BBČ+09] Jiří Barnat, Luboš Brim, Ivana Černá, Milan Češka and Jana Tůmová. Local Quantitative LTL Model Checking. In *International Workshop on Formal Methods for Industrial Critical Systems (FMICS)*, pages 63–78, 2008.

- [BBČ+08] Jiří Barnat, Luboš Brim, Ivana Černá, Milan Češka and Jana Tůmová. ProbDiVinE-MC: Multi-core LTL Model Checker for Probabilistic Systems. In *International Conference on Quantitative Evaluation of Systems (QEST)*, pages 77–78, 2008. (Tool paper.)

- [BBČ+07] Jiří Barnat, Luboš Brim, Ivana Černá, Milan Češka and Jana Tůmová. ProbDiVinE: A Parallel Qualitative LTL Model Checker. In *International Conference on Quantitative Evaluation of Systems (QEST)*, pages 215–216, 2007. (Tool paper.)

1.4 Thesis Structure

The remainder of the thesis is organized as follows.

Chapter 2 briefly introduces principles that we build on and summarizes the related work, including advances in quantitative formal methods, control strategy synthesis and the application of formal methods in mobile robotics.

Chapter 3 presents preliminaries used throughout the thesis. Namely, transition systems, linear temporal logic, automata-based approach to model checking and also timed automata and metric interval temporal logic and Markov decision processes together with probabilistic LTL and CTL are introduced.

Chapter 4 is devoted to the framework for analysis and control of systems with degradation. We present a modeling and a specification formalism as well

as two different approaches to model checking, namely a method exploiting translation into verification problem for timed automata and a method directly leveraging automata-based approach to model checking.

Chapter 5 focuses on the problem of optimal path planning with surveillance mission specified as an LTL formula. We formalize the optimization problem and present an automata-based solution to this problem. We demonstrate our approach on an experiment in a robotic testbed.

Chapter 6 studies the problem of least-violating robot path planning with the set of conflicting specifications. We focus on two separate subproblems: the problem of conflicting long-term tasks and the problem of conflicting safety rules. For both of them we propose algorithms to find the best paths with respect to a suitably defined metrics. Illustrative case studies are included.

Chapter 7 concludes the thesis.

Chapter 2

Principles and Related Work

In this chapter we set our contributions in the context of related work. We start with an informal introduction to the basic principles of model checking and control strategy synthesis that we build on and we narrow our focus to the two research areas that we see as the most related ones to our work; the fields of quantitative model checking and formal methods in mobile robotics.

2.1 Model Checking and Control Strategy Synthesis

Loosely speaking, the goal of *model checking* is to answer whether all behaviors of a system meet a given property. Dually, *control strategy synthesis* aims to find a control strategy for a system that, when followed, ensures the satisfaction of a desired requirement. For both of them, the starting point is modeling a system and a specification of requirements. Numerous formalisms can be considered for that purpose and the choice of the appropriate ones depends on the structure of the system and the property of interest.

The very basic models to describe behavior of various systems are finite, discrete, *labeled transition systems*. They are essentially directed graphs, whose vertices, *i.e.,* states, map to the states of the system and edges, *i.e.,* transitions labeled with actions, correspond to the changes in the system states caused by an execution of an action. The desired properties can be split into two categories based on their nature. While *linear-time properties* are evaluated on individual (infinite) runs of a given system, *branching-time properties* specify permitted behaviors of infinite trees of runs obtained by considering all possible futures from the current system state. For capturing properties in the first category, *Linear Temporal Logic (LTL)*, first introduced by Pnueli in [Pnu77], is widely used. On the other hand, *Computation Tree Logic (CTL)*, first used in [CE81, QS82], is a typical representative for formalizing properties in the latter category. CTL* [CE81] combines both the linear time and the branching time aspects and even more powerful specification formalism is modal μ-calcul [Koz83].

In this thesis, we study linear-time properties for several arguable reasons. First, in terms of reasoning about robotic behavior, it is most natural to impose

requirements on a single execution of the robot. Second, for a human, reasoning about a single run is more comfortable than reasoning about a whole tree of runs. Third, linear temporal logic has a great expressive power to capture many interesting properties of infinite runs while being quite intuitive and having some resemblance to natural language. An example of properties that can be expressed via an LTL formula include reachability (ϕ holds true in future), global absence (keep avoiding ϕ), response (whenever ϕ holds true, ψ must happen in future), reactivity (whenever ϕ holds true in future for any time point, ψ has to happen in future for any time point as well), sequencing (ϕ happens before ψ) and many others.

Given a labeled transition system and an LTL formula, the model checking question can be answered exploiting the concept of the *automata-based approach* [VW86]. The basic idea is as follows. Instead of checking that all runs of the given system satisfy the formula, we ask whether there exists a single run violating the formula. The formula is thus first negated and the negation is translated into a *Büchi automaton* [Büc62]. Then, a product automaton of the transition system and the Büchi automaton is built that accepts only the runs of the system violating the formula. Using quite simple and efficient graph algorithms such as nested depth first search [CVWY92b] the language emptiness of the product automaton is examined. In case an accepting run of the product automaton is found, it projects onto a counterexample in the transition system. Well-established automata-based LTL model checkers include tools such as Spin [Hol97] or DiVinE [BBR09] that is dedicated for parallel and distributed environments.

The automata-based approach can be utilized in a straightforward way to address the control strategy synthesis problem, too. Namely, if the given transition system is deterministic, *i.e.,* if the execution of each action at each state results into a transition to a unique state, the sought control strategy is simply a single system run. Hence, it is enough not to negate the formula in the above scheme. An accepting run of the product automaton (if such exists) then projects onto a run of the transition system that meets given requirements. In this work, we focus on this particular type of problems considering deterministic models only. We call them commonly *trace synthesis* problems.

The situation gets significantly more complicated when the given model of the controlled system is nondeterministic, *i.e.,* when the execution of some of the actions may have different consequences, for instance, due to reactivity with the system's environment. In such a case, the automaton designed for the purpose of control strategy synthesis needs to be deterministic (Büchi, if such exists, or Rabin [Rab69], if it does not) and the product automaton requires deeper graph analysis than checking for its language emptiness. Particularly, the prod-

uct automaton can be viewed as a 2-player graph game with Büchi or Rabin winning objective, respectively (see *e.g.,* [AG11] for an overview). Informally, the choice of the actions is driven by the game protagonist, while the nondeterministic choices are resolved by the game adversary. The goal is then to find a strategy for the game protagonist that ensures her victory regardless of the choices of the game adversary. The emerging factor in this kind of approaches is the determinization of the automata that is double exponential with respect to the size of the formula and thus extremely expensive. Several recent works focus on different ways how to cope with this. For instance, the authors in [KV05] recently suggest an approach trough universal co-Büchi instead of Rabin games. A different point of view on reducing the complexity is to consider just a subset of linear temporal logic as a specification, as the authors do *e.g.,* in [BJP$^+$12].

2.2 Quantitative Model Checking

The above described transition systems and linear temporal logics provide a powerful formalism and are sufficient to model and analyze a huge set of systems and their properties. Yet, none of the mentioned formalisms covers *quantitative* properties of systems. For instance, timing constraints such as "Whenever a request is submitted, a response comes within 30 seconds." are not supported in either the labeled transition system or the above temporal logics. Hence, *timed automata* have been developed in [AD90] and quantitative versions of temporal logics, such as Metric Temporal Logic (MTL), or Timed Computation Tree Logic (TCTL) have been derived in [Koy90] and [ACD93], respectively. Analysis of timed automata is quite hard for linear time properties. For instance, model checking of MTL formulas is undecidable. When punctuality is sacrificed, and *Metric Interval Temporal Logic (MITL)*, introduced in [AFH96], is used instead, the problem becomes solvable in EXPSPACE exploiting the ideas of the automata-based approach. In contrast, TCTL model checking can be handled much more efficiently (see, *e.g.,* [BK08]) and has been implemented in tools Kronos [Yov97], or Uppaal [BDL04] that has been constantly improved and extended since mid-nineties. One of its extensions, Uppaal Tiga, addresses the strategy synthesis problem through the implementation of timed games, enabling to find a time-optimal winning strategy with respect to liveness and safety objectives.

To formalize randomized behavior as well as uncertainty or unreliability, *probabilistic* models can be used. Namely, *Markov chains* and *Markov decision processes* together with the a probabilistic interpretation of LTL, a probabilistic extension of CTL, called *PCTL* (introduced in [HJ89]) and Continuous Stochastic Logic (CSL) [ASSB00] allow us to consider requirements such as "The prob-

ability that a message is delivered is at least 90%." Enhancing the probabilistic models with costs or rewards (as in Markov reward models or probabilistic (priced) timed automata) then provides an option to study properties such as "The probability that a message is delivered within 30 seconds is at least 90%." In model checking, we can even ask what is the maximal or minimal probability that a given system satisfies certain LTL formula. Dually, in control strategy synthesis, we seek for a controller optimizing the probability of the LTL formula satisfaction. The solution to the first named problem can be reached using an alternation of the automata-based approach designed for nonprobabilistic models (see *e.g.,* [BK08] for an overview). The latter problem is particularly interesting; in Markov chains, there is always only one action enabled in each state and therefore there is no point in searching for a control strategy. On the other hand, the control strategy for Markov decision processes, often called also scheduler or policy, can be found by playing a $1\frac{1}{2}$-player graph game on the product automaton with a Rabin winning objective and subsequent solving of a linear programming problem [BGL+04]. The tools for probabilistic model checking include PRISM [HKNP06] whose extension called PRISM-games aims to solve also selected strategy synthesis problems, MRMC [KZH+11] addressing among others also Markov reward models, and LiQuor [CB06] and ProbDiVinE-MC [BBČ+08], that both focus on probabilistic LTL verification.

Time and probability are the classical and arguably the most extensively studied instances of quantitative characteristics. The above paragraphs are just a non-exhaustive list of fundamental formalisms, techniques and tools, and it is important to note that quantitative model checking and control strategy synthesis addresses is much wider topic that covers reasoning about other significant quantitative aspects and their combinations.

Along these lines, this thesis aims at the development of formal methods framework for systems with degradation, also called decay. Unlike time, that flows in additive fashion, in systems with degradation, the quality that is subject to decay changes relatively, based on its current value, *i.e.,* in multiplicative fashion. This well-known phenomenon is present both in nature and our everyday lives (*e.g.,* in radioactive decay, biodegradation, or money inflation), and in computer-based systems (*e.g.,* in bit rot, degradation of a transmitted signal in a broadcasting network, decaying route flapping in route flapping damping). However, to our best knowledge our works [BČT09, BČT12a, BČT12b] have been the first ones addressing this characteristics.

The model we suggest, transition systems with degradation (TSD) have some resemblance to Markov decision processes. In Section 4.5 we actually show, that probability can be viewed as a special case of degrading property and that MDPs are a special variant of TSDs. As a specification formalism, we propose linear

temporal logic with degradation constraints (DLTL), which involves operators with built-in quantitative constraints on the level of degradation. Although we were partially inspired by PCTL that also involves operators of such kind, PCTL is a branching-time type of logic and its expressive power cannot be compared to DLTL. On the other hand, probabilistic LTL does not allow for any quantitative bounds in its operators.

Although interpreted over a significantly different model, the most related specification formalism to our DLTL is metric interval temporal logic as it is both linear-time logic and with constrained operators. The relation between transition systems with degradation and timed automata and DLTL and MITL is in detail investigated in Section 4.4.2.

Other related works include various temporal logics and methods for reasoning about properties that involve quantitative constraints on systems behavior. In [MN04], the authors introduce signal temporal logic (STL), a variant of MITL enhanced with a mapping from continuous domains into atomic propositions that can capture quantitative properties of dense-time real-valued signals. Later extension of STL, called timed-frequency logic (TFL) [DMB$^+$12] allows for expressing time-frequency properties of signals and interestingly, it can be used to specify and recognize music melodies. Both STL and TFL can be efficiently monitored.

Counting LTL [LMP10] provides a possibility to put constraints on the number of satisfying instances of its subformulas, such as "The response to each request occurs before three other requests are made." Counting LTL can be model checked either through translation to LTL or with a use of specifically tailored automata-based algorithms. In frequency LTL [BDL12], the aim is to express relative frequencies of subformulas satisfaction, such as that a formula ϕ has to hold only at 95% of the position of a run. The satisfiability problem for frequency LTL is undecidable, however, it becomes decidable when restricting to some of its fragments.

A different point of view on quantitative formal methods that is related to our work comes from the fact that classical yes/no model checking is often fragile. Even small perturbation in the system model may cause the opposite answer to the verification question. Therefore, efforts have been made towards the robustness of logics and in particular towards expressing to what extend a property holds. Literature in this direction includes [dAFS04, FLS08], where the authors consider model checking of quantitative LTL properties. They interpret quantitative LTL formulas over structures whose atomic propositions are not evaluated as true or false. Rather than that, they have values in $[0, 1]$ to express the level of their satisfaction. In [dAFH$^+$05] discounted CTL is introduced that enables to discount the importance of events based on how late they occur. Discounted

CTL formulas are not interpreted as true or false, their value is a real number in interval $[0, 1]$. The authors have also designed a model checking algorithm. Work of Alur et al. ([AETP01]) focuses on introduction of parametric temporal logic for model measuring, which permits finding the valuations of parameters that limit the scope of temporal operators. In contrast, the authors in [ABK12] define an extension of LTL with several quality operators used to prioritize different satisfying instances of an LTL formula. They also present an automata-based model checking algorithm.

2.3 Formal Methods in Robot Motion Planning

The use of formal methods employing a discrete model of an investigated system has recently gained considerable attention in a research field of mobile robotics, where a robot (or a group of robots) is aimed to be controlled (or coordinated) to autonomously perform an assigned task (*e.g.,* in [AM95, LK04, QBIZ04, BIP05, FGKGP09, KGFP09, WTM10]). Although several works have focused on model-checking of a controlled robotic system [AM95, QBIZ04], the major portion of the interest lies in the correct-by-construction controller design. The specific problem that has been intensively addressed and that we focus on in this thesis, can be informally stated as follows. Given a robotic system (*e.g.,* an autonomous robot vehicle in an environment), and a desired high-level task (*e.g.,* periodical surveillance of a certain set of locations), automatically find a sequence of robot motion primitives that, when being followed, guarantee the accomplishment of the task. A variety of temporal logics, including the Computation Time Logic (CTL) [LWAB10], Linear Temporal Logic (LTL) and its fragments [KB08, WTM09, KGFP09, STBR11, BKV10], and μ-calculus [KF09, KF12a] have been successfully utilized to express complex missions that arise in robotics applications. Tools that address robot planning with LTL specifications include *e.g.,* TuLiP [WTO+11] and LTLMoP [FJKG11].

The control strategy synthesis problem has been traditionally perceived differently in computer science, and in control theory and robotics. The formal methods perspective discussed earlier in this chapter represents a high-level approach to this problem. There, we start with a model of the robotic system that is discrete and finite in the state space, and a complex, temporal logic specification of the task. The result of control strategy synthesis are actions to be executed in the model. However, the focus is not on whether or how the found abstract control strategy can be translated into a low-level description of the robot's real motion. In contrast, the control theory and the robot motion planning approaches view this problem from a rather low-level angle. The complex continuous, infinite-state model of a robot (usually in the form of a set of differ-

ential equation) reflecting its dynamics, sensing, motion constraints or even its size is considered together with its initial and goal state. The task seems to be rather simple; to navigate the robot through the given environment to the goal state while avoiding collision with obstacles.

Generally speaking, in formal methods, "simple" (discrete, finite) models have been traditionally controlled from "complex" (temporal logic) specifications, whereas in robot control, "complex" (differential equation) motion models have been controlled from "simple" (reachability, safety) specifications. Similar holds for the verification; while in formal methods, discrete models have usually been verified against temporal logic formulas, in control theory, infinite, continuous-space systems are analyzed to verify properties like reachability, stability or safety. Lately, a significant progress has been made towards bridging the gap between the two points of view.

In robot path planning, the high-level and the low-level points of view on control have been meeting through a three-layered hierarchical approach as commonly used in [FGKGP09, KGFP09, WTM10] and many other works. In the first layer, a discretization of the robotic system is built. In the second one, a high-level discrete controller is found for the model from a temporal logic specification and finally in the third one, the discrete controller is translated into a continuous plan. The discrete transition system can be built by using sampling-based approaches [BKV10, KF09, KF12a] or cell decomposition methods [LaV06], representing each cell as a system state and connecting adjacent cell states with transitions.

In this thesis, we do not address how exactly the discrete model is built or how the discrete control strategy that we design is applied in the robotic system itself. In other words, we only focus on the second layer, which we call *high-level robot path planning* in the sequel. Clearly, if the model is *e.g.,* a deterministic transition system, or a Markov decision process, and the specification is an LTL formula, the standard formal methods described in the previous section can be applied here. However, further research in this direction is motivated by the nature of the robotics context. Specific problem instances arise when considering multi-robot planning, uncertainty or reactivity of the environment, or the desire for optimal control.

In this work, we focus on selected problems in optimal robot path planning. Particularly, frequently there are multiple robot paths that satisfy a given specification. In this case, one would like to choose the *optimal* path according to a cost function. The current tools from model checking do not provide a method for doing this. Based on our papers [STBR11, STBR10], in this work we consider linear temporal logic specifications, and a particular form of cost function, and provide a method for computing optimal paths.

In terms of optimizing paths, the most closely related work has been on the vehicle routing problem (VRP) [TV01]. In vehicle routing, the problem is to plan routes for vehicles to optimally service customers. The VRP generalizes the well known traveling salesman problem (TSP) by considering aspects such as multiple vehicles, vehicles with capacity constraints, and vehicles that must depart and return to specified depot locations. Such aspects can be thought of as specific examples of logical, or temporal constraints. While the vehicle routing problem is generally NP-hard, many effective heuristics have been developed which provide good solutions to moderately sized problems [Lap09].

Recent results [KF08a, KF08b] present extensions of vehicle routing problems to more general classes of temporal constraints (see also [KRKF09]). The authors in [KF08b] consider vehicle routing with metric temporal logic specifications. The goal is to minimize a cost function of the vehicle paths (such as total distance traveled). The authors present a method for computing an optimal set of paths by converting the problem to a mixed integer linear program (MILP). While the approach is computationally intensive, it has been used to solve problems of real-world significance. However, their method cannot be applied to the persistent monitoring and data gathering applications that are subject to our focus due to the fact that their method applies only to a certain subset of specifications.

Our work on LTL optimal planning presented in [STBR11, STBR10] (and also in this thesis) has been extended later in [DSBR11] towards finding the optimal traces for robotic systems modeled as Markov decision processes with respect to expected value of a suitably defined cost function. This approach have been later enhanced in our works [CTB12, CTUB13], where we have considered unknown elements in the environment that the robot learns during its execution. Another known extension of our work is the optimal surveillance for multi-robot system [USD$^+$11, USDB12]. Other related work in optimal LTL robot control includes papers [DBC10, DLB12, STBČ12] aimed at the collection of maximal rewards in a partially unknown environment while guaranteeing the satisfaction of an LTL formula. The uncertainty in their case comes from the assumption that the robot has only limited sensing range.

The second direction in optimal robot path planning that we investigate in this thesis is finding a *least-violating* robot path in case that a given specification cannot be satisfied as a whole. In this sense, our work is related to [RKG11, RKG12], where the authors study the following problem: given an LTL specification and a model that does not satisfy this specification, decide whether or not the invalidity is limited to the provided model. Related literature includes among others also [CRST08], where the authors aim to pinpoint the (un)realizable fragments of the specification to reveal causes of the specification violation.

Other authors propose to construct control strategies with minimal changes

in the input. On one hand, in [Fai11, KFS12, KF12b], the authors aim to revise the given specification to (i) make it satisfiable by the given model, and (ii) make it as close as possible to the original specification according to a suitable metric. On the other hand, in [Hau12] the author focuses on finding the least set of constraints (in the model) violating which results in the satisfaction of the specification. Variants of this problem have been addressed also from the perspective of control theory and computer science. In [BEGL99, BGK$^+$11] the authors aim to repair a model; a transition system or a Markov chain in order to ensure the satisfaction a given CTL or PCTL formula, respectively.

Our work is probably closest to the papers aimed at finding a least-violating strategy without changing the model nor the specification. Namely, in [CY98], the authors consider a system modeled as a Markov decision process and a set of specifications given in the form of Büchi automata, each of which is assigned a reward. They aim to find a strategy maximizing the total reward gained for the specifications weighted by the respective probabilities with which they are satisfied. The solution builds on translating the problem into a linear programming problem. Unfortunately, the time complexity of their algorithm is exponential in the size of the automata. In [DF11], the authors consider a transition system with the variables partitioned into control inputs for the car, controllable environment variables and disturbances. The mission specifications are captured as a set of LTL formulas ordered according to their priorities. The goal is to find a strategy for the robot ensuring the satisfaction of the maximal subset high-priority formulas regardless of the environmental disturbances. Our approach is different as we do not consider the disturbances. Thus, we can solve the problem more efficiently.

Chapter 3

Preliminaries

In this chapter, we fix the notation and introduce necessary preliminaries that will be used throughout the thesis. We define several variants of a transition system as the basic modeling formalisms, Linear Temporal Logic (LTL) as the specification formalism and automata over finite and infinite words that can be used as an alternative means to capture the linear temporal properties of finite and infinite system traces. Later, we will use the infinite semantics in Chapters 4, 5, and Section 6.2. In Section 6.3, we will consider the finite semantics. Given the described model and specification, we formally state the model-checking and the control strategy synthesis problems and we present the details of the well-known automata-based approach to their solution. Further, we introduce some preliminaries that will be needed in Chapter 4, namely timed automata and Metric Interval Temporal Logic (MITL) and also Markov Decision Processes (MDP) and the probabilistic version of LTL as well as probabilistic computation tree logic (PCTL).

3.1 Notation

Given a set S, let $|\mathsf{S}|$, 2^{S}, S^*, S^+ and S^ω denote the cardinality of S, the set of all subsets of S, and set of all finite, finite nonempty, and infinite sequences of elements of S, respectively. A finite and infinite sequence of elements of S is called a finite and infinite word over S, respectively. Given a finite word w and a finite or an infinite word w' over S, we use $w \cdot w'$ and $w^\omega = w \cdot w \cdot w \ldots$ to denote the word obtained by the concatenation of w and w', and by infinitely many repetitions of w, respectively. Given an infinite word $w = w(0)w(1)\ldots$, let w^i denote the suffix of the word w starting at $w(i)$, *i.e.,* $w^i = w(i)w(i+1)\ldots$.

3.2 Transition Systems

Definition 3.2.1 (Transition System)**.** *A (non-deterministic) transition system (TS) is a tuple* $\mathcal{T} = (S, S_{\text{init}}, Act, T, AP, L)$*, where*

- S *is a finite set of states;*
- $S_{\text{init}} \subseteq S$ *is the set of initial states;*
- Act *is a finite set of actions;*
- $T \subseteq S \times Act \times S$ *is a non-deterministic transition relation;*
- AP *is a set of atomic propositions; and*
- $L : S \to 2^{AP}$ *is a labeling function.*

A transition $t = (s, \alpha, s') \in T$ represents that the system can make a transition from state s to state s' under action α. The labeling function determines the set of atomic propositions $L(s)$ that hold true in state s.

A *finite trace* of $\mathcal{T}$ is a finite sequence $r_{\text{fin}} = s_0\alpha_0 s_1\alpha_1 \dots s_{n-1}\alpha_{n-1}s_n$ of states and actions, such that $s_0 \in S_{\text{init}}$, and $(s_i, \alpha_i, s_{i+1}) \in T$, for all $0 \leq i < n$. The length of a finite trace $r_{\text{fin}} = s_0\alpha_0 s_1\alpha_1 \dots s_{n-1}\alpha_{n-1}s_n$ is equal to $n + 1$. A finite trace $r_{\text{fin}} = s_0\alpha_0 s_1\alpha_1 \dots s_{n-1}\alpha_{n-1}s_n$ produces a *finite word* $w(r_{\text{fin}}) = L(s_0)L(s_1)\dots L(s_n)$. The set of all finite words produced by all finite traces of $\mathcal{T}$ is called language of $\mathcal{T}$ and denoted by $\mathcal{L}_{\text{fin}}(\mathcal{T})$.

Analogously, an *infinite trace* of $\mathcal{T}$ is an infinite sequence $r = s_0\alpha_0 s_1\alpha_1 \dots$ of states and actions such that $s_0 \in S_{\text{init}}$ and $(s_i, \alpha_i, s_{i+1}) \in T$, for all $0 \leq i$. An infinite trace $r = s_0\alpha_0 s_1\alpha_1 \dots$ produces an infinite *word* $w(r) = L(s_0)L(s_1)\dots$. The language of all words produced by infinite traces of $\mathcal{T}$ is denoted by $\mathcal{L}(\mathcal{T})$.

A *control strategy* (or, simply, a *strategy*) for $\mathcal{T}$ is a function $\Omega : S^+ \to Act$, that maps each nonempty finite sequence of states to an action. A finite trace $r_{\text{fin}} = s_0\alpha_0 s_1\alpha_1 \dots s_{n-1}\alpha_{n-1}s_n$, and an infinite trace $r = s_0\alpha_0 s_1\alpha_1 \dots$ of $\mathcal{T}$ is a trace *under* control strategy Ω if $\alpha_i = \Omega(s_0 \dots s_i)$, for all $0 \leq i < n$, and for $0 \leq i$, respectively. A *history-independent control strategy* Ω has the property that $\Omega(s_0 \dots s_n) = \Omega(s'_0 \dots s'_n)$ whenever $s_n = s'_n$. Therefore, we view it as a function $\Omega : S \to Act$.

A *weighted transition system* is a tuple $\mathcal{T} = (S, S_{\text{init}}, Act, T, AP, L, W)$, such that the tuple $(S, S_{\text{init}}, Act, T, AP, L)$ is a transition system, and $W : T \to \mathbb{N}$ is a weight function. The weight function $W(t)$ determines the weight of the transition $t \in T$. We define the weight of a finite trace $r_{\text{fin}} = s_0\alpha_0 s_1\alpha_1 \dots s_{n-1}\alpha_{n-1}s_n$ as the sum of the weights of its transitions. With a slight abuse of notation, this quantity is denoted as follows:

$$W(s_0\alpha_0 s_1\alpha_1 \dots s_{n-1}\alpha_{n-1}s_n) = \sum_{i=0}^{n-1} W\big((s_i, \alpha_i, s_{i+1})\big).$$

A transition system is called *deterministic* (DTS) if S_{init} is a singleton and if there do not exist $s, s', s'' \in S$ and $\alpha \in Act$, such that $(s, \alpha, s') \in T$, $(s, \alpha, s'') \in T$ and $s' \neq s''$. In other words, in deterministic systems, an application of an enabled action always results into a transition to a single state. Note, that in a deterministic transition system there exists always at most one infinite trace and at most one finite trace of length n, for all $n \in \mathbb{N}$ under a given control strategy. In fact, the control strategy and the history-independent control strategy can then be viewed as functions $\Omega : S^+ \to S$ and $\Omega : S \to S$, respectively.

As the effect of each action choice at a state s of a DTS is uniquely determined, the action labels can be replaced with the tuple (s, s'), where the state s is the state from and the state s' is the state to which the particular transition leads. The action labels can thus omitted from the transition system without any harm. Namely, the deterministic transition system can be alternatively defined as a tuple $\mathcal{T} = (S, s_{\text{init}}, T, AP, L)$ with the transition relation $T \subseteq S \times S$. The finite and infinite traces are then defined as sequences of states only, and the words produced by the traces as well as the language of $\mathcal{T}$ are defined in the expected way as the sequence of state labels and the set of all words, respectively. The alternative definition of a weighted deterministic transition system is analogous; the system is a tuple $\mathcal{T} = (S, s_{\text{init}}, T, AP, L, W)$ and the definitions of a trace, a word and a language are straightforward.

In this work, we will use the alternative definitions of deterministic transition systems in Chapters 5 and 6 that focus on control of robotic systems. The use of the alternative definition will always be clear from the context.

3.3 Linear Temporal Logic

Definition 3.3.1 (Formulas of the LTL)**.** *LTL formulas over the set AP of atomic propositions are constructed inductively according to the following rules:*

$$\phi ::= \mathit{true} \mid \pi \mid \neg\phi \mid \phi \wedge \phi \mid \mathsf{X}\,\phi \mid \phi\,\mathsf{U}\,\phi,$$

where true is a predicate that is always true, $\pi \in AP$, $\neg$ (negation) and $\wedge$ (conjunction) are standard Boolean operators and X (next) and U (until) are temporal operators.

LTL formulas are interpreted over infinite words over 2^{AP}, such as those produced by the infinite traces of the transition system from Def. 3.2.1.

Definition 3.3.2 (LTL Semantics)**.** *Let ϕ be an LTL formula over AP, and*

let w be an infinite word. The satisfaction relation $\models$ is defined as follows:

$$
\begin{aligned}
w &\models \mathit{true} && \text{always} \\
w &\models \pi && \iff \pi \in w(0) \\
w &\models \neg\phi && \iff w \not\models \phi \\
w &\models \phi \wedge \psi && \iff w \models \phi \text{ and } w \models \psi \\
w &\models \mathsf{X}\,\phi && \iff w^1 \models \phi \\
w &\models \phi\,\mathsf{U}\,\psi && \iff \exists i \geq 0.\ w^i \models \psi \text{ and } \forall\, 0 \leq j < i.\ w^j \models \phi
\end{aligned}
$$

Informally speaking, the word $w = w(0)w(1)\ldots \in (2^{AP})^\omega$ *satisfies* the atomic proposition π if π is satisfied in the first position of the word w, *i.e.*, if $\pi \in w(0)$. The formula $\mathsf{X}\,\phi$ states that ϕ holds in the following state. The formula $\phi\,\mathsf{U}\,\psi$ states that ψ is true eventually, and ϕ is true at least until ψ is true. Besides that, we define temporal operators

$$
\begin{aligned}
\mathsf{F}\,\phi &\equiv \mathit{true}\,\mathsf{U}\,\phi && (\mathit{eventually}) \\
\mathsf{G}\,\phi &\equiv \neg\mathsf{F}\,\neg\phi && (\mathit{globally}) \\
\phi\,\mathsf{R}\,\psi &\equiv \neg(\neg\phi\,\mathsf{U}\,\neg\psi) && (\mathit{release})
\end{aligned}
$$

Intuitivelly, $\mathsf{F}\,\phi$ and $\mathsf{G}\,\phi$ state that ϕ holds *eventually* and *always*, respectively. The formula $\phi\,\mathsf{R}\,\psi$ states that ψ holds always or at least till the satisfaction ϕ "releases" the necessity for ψ to hold.

The language of all words that satisfy an LTL formula ϕ is denoted by $\mathcal{L}(\phi)$. With a slight abuse of notation, we extend the satisfaction relation to traces of $\mathcal{T}$, *i.e.,* a trace r *satisfies* ϕ (denoted by $r \models \phi$) if and only if the word $w(r)$ produced by r satisfies ϕ. Similarly, a word w and a trace r satisfies a set of formulas Φ ($w \models \Phi$ and $r \models \Phi$) if and only if $w \models \phi$ and $r \models \phi$, for all $\phi \in \Phi$, respectively.

Given a formula ϕ, we use $|\phi|$ to denote the *size* of the formula, *i.e.,* the number of operators present in ϕ, defined inductively as follows: $|\mathit{true}| = 0$, $|\pi| = 0$, for all $\pi \in AP$, $|\neg\phi| = |\mathsf{X}\,\phi| = |\phi| + 1$, $|\phi \wedge \psi| = |\phi\,\mathsf{U}\,\psi| = |\phi| + |\psi| + 1$, for all LTL formulas ϕ, ψ. Given a set of formulas Φ, we use $|\Phi|$ to denote $\sum_{\phi\in\Phi} |\phi|$.

LTL over Finite Words

Although LTL is primarily defined and used to reason about infinite traces and words, it also has a meaningful, expected semantics defined over finite traces and words, providing thus a compact means to represent interesting languages. In

this thesis, we take the syntax of FLTL [MP95], which is nicely summarized also in [GP02]. In short, FLTL formulas in addition to standard operators include weak next operator defined as $\overline{\mathsf{X}}\phi \equiv \neg\mathsf{X}\neg\phi$ to express that formula ϕ hods in the next state, if such a state is present on a trace.

We define syntactically safe FLTL fragment [KYV01] as the formulas in positive normal form, *i.e.,* those with negations applied to atomic propositions only, that use only the temporal operators X, and R.

3.4 Automata over Finite and Infinite Words

Definition 3.4.1 (Automaton)**.** *A (non-deterministic) automaton is a tuple* $\mathcal{A} = (Q, Q_{\text{init}}, \Sigma, \delta, Acc)$*, where*

- Q *is a finite set of states;*
- $Q_{\text{init}} \subseteq Q$ *is the initial state;*
- Σ *is an input alphabet;*
- $\delta \subseteq Q \times \Sigma \times Q$ *is a non-deterministic transition relation;*
- Acc *is the acceptance condition.*

Finite Automata

For *finite automata*, the acceptance condition Acc is given as a set of final states $F \subseteq Q$. The semantics of finite automata are defined over finite input words over Σ (such as those produced by finite traces of the transition system from Def. 3.2.1 if $\Sigma = 2^{AP}$). A *run* of the finite automaton $\mathcal{A}$ *over* a finite input word $w_{\text{fin}} = w(1) \dots w(n)$ is a finite sequence $\rho_{\text{fin}} = q_0 q_1 \dots q_n$, such that $q_0 \in Q_{\text{init}}$, and $(q_{i-1}, w(i), q_i) \in \delta$, for all $1 \leq i \leq n$. If $q_n \in F$, then ρ_{fin} is accepting and the word w_{fin} is accepted by the automaton. We say that a trace $r_{\text{fin}} = s_0 s_1 \dots s_n$ of $\mathcal{T}$ *satisfies* the property defined by $\mathcal{A}$ (denoted by $r_{\text{fin}} \models \mathcal{A}$) if and only if the word $w(r_{\text{fin}})$ produced by r_{fin} is accepted by $\mathcal{A}$. The language of all words accepted by an FA $\mathcal{A}$ is denoted by $\mathcal{L}_{\text{fin}}(\mathcal{A})$.

Languages accepted by finite automata are called regular languages and they can be captured conveniently as well through the expressivity-equivalent regular expressions (see, *e.g.,* [Sip12]). Furthermore, FLTL formulas can be algorithmically translated into finite automata and specifically a syntactically safe FLTL formula can be translated into a finite automaton with the set of accepting state equal to the set of all states.

ω-automata

The semantics of *ω-automata* are defined over infinite input words over Σ (such as those generated by infinite traces of the transition system from Def. 3.2.1 if $\Sigma = 2^{AP}$). A *run* of the ω-automaton $\mathcal{A}$ *over* an input word $w = w(0)w(1)\ldots$ is an infinite sequence of states $\rho = q_0q_1q_2\ldots$, such that $q_0 \in Q_{\text{init}}$, and $(q_i, w(i), q_{i+1}) \in \delta$, for all $i \geq 0$.

A run $\rho = q_0q_1\ldots$ is *accepting* if it satisfies the acceptance condition *Acc*. For *Büchi automata (BA)*, *Acc* is a set of states $F \subseteq Q$, and ρ is accepting if it intersects F infinitely many times. For *generalized Büchi automata (GBA)*, the acceptance condition is a set of sets of states $\mathcal{F} = \{F_1, \ldots, F_m\} \subseteq 2^Q$ and ρ is accepting if it intersects F_i infinitely many times for all $F_i \in \mathcal{F}$. A word w is *accepted* by $\mathcal{A}$ if there exists an accepting run over w. The *language* of all words accepted by $\mathcal{A}$ is denoted by $\mathcal{L}(\mathcal{A})$.

Let $\rho = q_0q_1\ldots$ be an accepting run over $w = w(0)w(1)\ldots$ of a Büchi automaton $\mathcal{B}$. We use

$$\mathit{Frag}(\rho) = \{q_i \ldots q_k \mid q_i, q_k \in F \text{ and } q_j \notin F \text{ for all } i < j < k\}$$

to denote the set of all finite fragments of ρ that begin and end in an accepting state and do not contain any other accepting state. Note that each accepting run ρ corresponds to a unique sequence of fragments.

Each generalized Büchi automaton $\mathcal{G} = (Q_\mathcal{G}, Q_\mathcal{G}^{\text{init}}, \Sigma, \delta_\mathcal{G}, \mathcal{F} = \{F_1, \ldots, F_m\})$ can be transformed into a Büchi automaton $\mathcal{B} = (Q, Q_{\text{init}}, \Sigma, \delta, F)$, such that $\mathcal{L}(\mathcal{B}) = \mathcal{L}(\mathcal{G})$ as follows.

- $Q = Q_\mathcal{G} \times \{1, \ldots, m\}$;
- $Q_{\text{init}} = \{(q, 1) \mid q \in Q_\mathcal{G}^{\text{init}}\}$;
- $\big((q, j), \sigma, (q', j')\big) \in \delta$ if and only if $(q, \sigma, q') \in \delta_\mathcal{G}$, and
  - $q \notin F_j$ and $j' = j$, or
  - $q \in F_j$ and $j' = (j \mod m) + 1$; and
- $F = F_1 \times \{1\}$.

Further, given n Büchi automata $\mathcal{B}_1, \ldots, \mathcal{B}_k$, where $\mathcal{B}_i = (Q_i, Q_i^{\text{init}}, \Sigma, \delta_i, F_i)$, for all $1 \leq i \leq n$, a Büchi automaton $\mathcal{B} = (Q, Q_{\text{init}}, \Sigma, \delta, F)$, such that $\mathcal{L}(\mathcal{B}) = \mathcal{L}(\mathcal{B}_1) \cap \ldots \cap \mathcal{L}(\mathcal{B}_n)$ can be built as follows.

- $Q = Q_1 \times \ldots \times Q_n \times \{1, \ldots, n\}$;
- $Q_{\text{init}} = \{(q_1, \ldots, q_n, 1) \mid q_i \in Q_i^{\text{init}}\}$;

- $\big((q_1,\ldots,q_n,j),\sigma,(q'_1,\ldots,q'_n,j')\big) \in \delta$ if and only if $(q_i,\sigma,q'_i) \in \delta_i$, for all $i \in \{1,\ldots,n\}$, and
  - $q_j \notin F_j$ and $j' = j$, or
  - $q_j \in F_j$ and $j' = (j \mod n) + 1$; and
- $F = F_1 \times Q_2 \times \ldots \times Q_n \times \{1\}$.

Intuitively, the set of states of $\mathcal{B}$ can be viewed as n copies (layers) of the Cartesian product of the sets of states $Q_1 \times \ldots \times Q_n$. Given a state $q = (q_1,\ldots,q_n,i)$ of $\mathcal{B}$, we denote by $layer(q)$ the last element of the tuple, *i.e.,* $layer((q_1,\ldots,q_n,i)) = i$.

Any LTL formula ϕ over AP can be translated into a Büchi automaton $\mathcal{B}_\phi$ with alphabet 2^{AP}, such that $\mathcal{L}(\phi) = \mathcal{L}(\mathcal{B}_\phi)$. A number of standard translation algorithms (see, *e.g.,* [GPVW96, GO01]) rely on a three-step procedure: First, the formula is normalized, second, it is translated into a generalized Büchi automaton and third, the obtained GBA is finally translated into a language-equivalent Büchi automaton.

Properties of Automata

For each non-deterministic finite automaton $\mathcal{A}$, each non-deterministic Büchi automaton $\mathcal{B}$ and each non-deterministic generalized Büchi automaton $\mathcal{G}$ there exist an FA $\mathcal{A}' = (Q_\mathcal{A}, Q_\mathcal{A}^{\text{init}}, \Sigma, \delta_\mathcal{A}, F_\mathcal{A})$, a BA $\mathcal{B}' = (Q_\mathcal{B}, Q_\mathcal{B}^{\text{init}}, \Sigma, \delta_\mathcal{B}, F_\mathcal{B})$, and a GBA $\mathcal{G}' = (Q_\mathcal{G}, Q_\mathcal{G}^{\text{init}}, \Sigma, \delta_\mathcal{G}, \mathcal{F}_\mathcal{G})$ with the property that $Q_\mathcal{A}^{\text{init}}$, $Q_\mathcal{B}^{\text{init}}$, and $Q_\mathcal{G}^{\text{init}}$ is a singleton, respectively. Therefore, we sometimes freely replace $Q_\mathcal{A}^{\text{init}} \subseteq Q_\mathcal{A}$, $Q_\mathcal{B}^{\text{init}} \subseteq Q_\mathcal{B}$, and $Q_\mathcal{G}^{\text{init}} \subseteq Q_\mathcal{G}$ with $q_\mathcal{A}^{\text{init}} \in Q_\mathcal{A}$, $q_\mathcal{B}^{\text{init}} \in Q_\mathcal{B}$, and $q_\mathcal{G}^{\text{init}} \in Q_\mathcal{G}$, respectively.

A finite, Büchi, or generalized Büchi automaton $\mathcal{A} = (Q, Q_{\text{init}}, \Sigma, \delta, Acc)$ is *non-blocking* if for all $q \in Q, \sigma \in \Sigma$ there exists $q' \in Q$, such that $(q,\sigma,q') \in \delta$. For each automaton $\mathcal{A} = (Q, q_{\text{init}}, \Sigma, \delta, Acc)$ a language equivalent non-blocking automaton can be constructed simply by adding a new state q_{new} to Q and introducing a transition $(q,\sigma,q_{\text{new}})$ for all $q \in Q \cup \{q_{\text{new}}\}, \sigma \in \Sigma$, satisfying the property that $(q,\sigma,q') \notin \delta$ for all $q' \in Q$.

An automaton $\mathcal{A}$ can be viewed as a directed graph, which is structure $G = (V_G, E_G)$, where the set of nodes V_G is the set of states Q and the set of edges E_G are given in the expected way by the transition function δ, respectively.

Hence, we sometimes refer to runs of automata as to *paths*[1]. An *infinite path* of an ω-automaton graph is an infinite suffix of an infinite run. A *finite*

1. A path in a graph is something else than a path of a robot and the concrete meaning will always follow from the context.

path in a finite automaton or an ω-automaton graph $G_{\mathcal{A}}$ starting in q_i and leading to q_j is a finite subsequence $q_i q_{i+1} \dots q_{j-1} q_j$ of an infinite or finite run $q_0 \dots q_i q_{i+1} \dots q_{j-1} q_j \dots$ or $q_0 \dots q_i q_{i+1} \dots q_{j-1} q_j \dots q_n$. A finite path is *simple* if and only if $q_i \neq q_j$, for all $0 \leq i < j \leq n$. A path $q_0 q_1 \dots q_{n-1} q_n$ is a *cycle* if and only if $q_0 \dots q_{n-1}$ is a simple path and furthermore, $q_0 = q_n$. A *lasso* is a path $q_0 \dots q_i q_{i+1} \dots q_n$, such that $q_0 \dots q_i$ is a simple path and $q_{i+1} \dots q_n$ is a cycle. A state q' is *reachable* from q if there exists a path starting at q and ending at q'.

3.5 Automata-based Approach to Model Checking

Given a transition system $\mathcal{T}$ and an LTL formula ϕ (interpreted over infinite words), the *model checking problem* is to prove or disprove that all infinite traces of $\mathcal{T}$ satisfy ϕ, whereas the *control strategy synthesis problem* is to find a control strategy for $\mathcal{T}$, such that all traces of $\mathcal{T}$ under the found strategy satisfy ϕ. In this work, we focus on the instance of control strategy synthesis problem that takes as an input a deterministic transition system $\mathcal{T}$ and thus, instead of a control strategy, it is enough to search for a single trace of $\mathcal{T}$ satisfying ϕ. Both the model checking problem and the deterministic control strategy synthesis problem can be addressed by solving the following problem, that we will commonly call the *satisfying trace synthesis problem* and that will, in some variant, be our key question throughout this whole work.

Problem Statement 3.5.1 (Satisfying trace synthesis problem)**.** *Find a trace of a transition system $\mathcal{T}$ that satisfies an LTL formula ϕ.*

The problem can be addressed with the use of so-called automata based approach, consisting ot three consecutive steps. First, we translate the formula ϕ into a corresponding Büchi automaton. Second, we construct a product automaton $\mathcal{P}$ that captures all the behaviors of $\mathcal{T}$ satisfying $\mathcal{B}$. Finally third, an accepting run of $\mathcal{P}$ projects onto a trace of $\mathcal{T}$, that solution to Problem 3.5.1.

Definition 3.5.2 (Product Automaton)**.** *A product automaton of a transition system $\mathcal{T} = (S, S_{\text{init}}, Act, T, AP, L)$ and a BA $\mathcal{B} = (Q, Q_{\text{init}}, \Sigma, \delta, F)$ is a Büchi automaton $\mathcal{P} = \mathcal{T} \otimes \mathcal{B} = (Q_{\mathcal{P}}, Q_{\mathcal{P}}^{\text{init}}, \delta_{\mathcal{P}}, F_{\mathcal{P}})$, where*

- $Q_{\mathcal{P}} = S \times Q$;
- $Q_{\mathcal{P}}^{\text{init}} = \{(s, q) \mid s \in S_{\text{init}} \text{ and } q \in Q_{\text{init}}\}$;
- $((s, q), (s', q')) \in \delta_{\mathcal{P}}$ *if there exists* $\alpha \in Act$, *such that* $(s, \alpha, s') \in T$ *and* $(q, L(s), q') \in \delta$; *and*
- $F_{\mathcal{P}} = S \times F$.

The product automaton has a trivial alphabet, which is therefore omitted. An accepting run $\rho_{\mathcal{P}}$ of the product automaton projects onto a trace r of $\mathcal{T}$ (denoted by $r = proj(\rho_{\mathcal{P}})$) that satisfies the property captured by the Büchi automaton $\mathcal{B}$. Vice versa, any trace of $\mathcal{T}$ satisfying the property corresponds to an accepting run of the product automaton. Furthermore, if there exists an accepting run $\rho_{\mathcal{P}}$ of $\mathcal{P}$, then there exists an accepting run $\rho'_{\mathcal{P}}$ of $\mathcal{P}$ in a *prefix-suffix structure*, *i.e.,* $\rho'_{\mathcal{P}} = \rho_{\text{pref}} \cdot (\rho_{\text{suf}})^{\omega}$ for some finite sequences ρ_{pref} and ρ_{suf} of states of $\mathcal{P}$, such that the first state of ρ_{suf} is an accepting state from $F_{\mathcal{P}}$. Intuitively, the prefix can be thought of as the transient, while the suffix is the steady-state periodic behavior.

The product automaton is a Büchi automaton and as such, it can be viewed as a graph. Efficient graph search algorithms can be used for finding a prefix ρ_{pref} (a simple path from the initial state to an accepting state in the product graph) followed by a periodically repeated suffix ρ_{suf} (a cycle in the product graph containing an accepting state) of an accepting run $\rho = \rho_{\text{pref}} \cdot (\rho_{\text{suf}})^{\omega}$ (a lasso-shaped path in the product graph). One of the standard algorithms to do so is *nested depth-first search (DFS)* [BK08], successfully implemented, for instance, in the pioneer model checker SPIN [Hol03]. The worst-case running time complexity of this algorithm is linear in time and space with respect to the size, *i.e.,* the number of states and transitions of the product automaton $\mathcal{P}$.

3.6 Timed Automata and Metric Interval Temporal Logic

A timed automaton is an ω-automaton equipped with a finite set of real-valued clock variables (*clocks*) that can be intuitivelly viewed as stopwatches allowing us to reason about timed properties of real-time systems.

A *clock constraint* γ over finite set of clocks X is a finite expression constructed according to the grammar

$$\gamma ::= x \bowtie c \mid \gamma \wedge \gamma,$$

where $\bowtie \in \{<, \leq, >, \geq\}$, $x \in X$, and $c \in \mathbb{R}_{\geq 0}$. Let $CC(X)$ denote the set of all clock constraints over X. A *clock valuation* ν is a function $\nu : X \to \mathbb{R}_{\geq 0}$ assigning to each clock $x \in X$ its current value $\nu(x)$. We use $\nu + c$ to denote valuation ν', where $\nu'(x) = \nu(x) + c$ for each $x \in X$.

Definition 3.6.1 (Timed Automaton)**.** *A timed automaton (TA) is a tuple* $\mathcal{A} = (Q, Q_{\text{init}}, \Sigma, X, \delta, Inv, AP, L)$*, where*

- Q *is a finite set of states;*
- $Q_{\text{init}} \subseteq Q$ *is a set of initial states;*

- Σ *is a finite set of actions;*
- X *is a finite set of clocks;*
- $\delta \subseteq Q \times CC(X) \times \Sigma \times 2^X \times Q$ *is a transition relation;*
- $Inv : Q \to CC(X)$ *is an invariant-assignment function;*
- AP *is a set of atomic propositions; and*
- $L : Q \to 2^{AP}$ *is a labeling function.*

A 5-tuple $\tau = (q_1, \gamma, \sigma, R, q_2) \in \delta$ corresponds to a transition from state q_1 to q_2 labeled with σ that is enabled if constraint γ is satisfied. R denotes the subset of clock variables that are reset to zero when the transition is executed. Time can progress (*i.e.* the value of clock can increase) in states, whereas transitions between states always take zero time. Function Inv assigns to each state an invariant that gives a limit on how much time can be spent in that state. There are two possible ways how a TA can evolve: via *discrete* transitions *i.e.*, those between states, and *delay* transition, *i.e.*, staying in a state with letting time pass.

A run of a timed automaton is a sequence

$$\rho = (q_0, \nu_0)c_0(q_0, \nu_0')\sigma_0(q_1, \nu_1)c_1(q_1, \nu_1')\sigma_1(q_2, \nu_2)\ldots,$$

such that $q_0 \in Q_{\text{init}}$, $\forall x \in X.\, \nu_0(x) = 0$, and $\forall i \in \mathbb{N}$ it holds

- $c_i \in \mathbb{R}_{\geq 0}$, $\nu_i'(x) = \nu_i(x) + c_i$, for all $x \in X$, and ν_i' satisfies $Inv(q_i)$, and
- there exists $(q_i, \gamma_i, \sigma_i, R_i, q_{i+1}) \in \delta$, such that ν_i' satisfies γ_i, $\nu_{i+1}(x) = \nu_i'(x)$, for all $x \in X \setminus R_i$, $\nu_{i+1}(x) = 0$ for all $x \in X \cap R_i$, and ν_{i+1} satisfies $Inv(q_{i+1})$.

A *position* on the run ρ is defined as any state that may appear during the run, *i.e.,* a tuple (q_i, ν), where $\nu = \nu_i + c$, and $c \leq c_i$. A *time duration* $\mathsf{T}_\rho(q_i, \nu)$ up to position (q_i, ν) is the sum of the delays up to this position $\mathsf{T}_\rho(q_i, \nu) = \sum_{j=0}^{i-1} c_i + c$. We denote by $\rho(\mathsf{t})$ and ρ^{t} the position (q, ν) on ρ, such that $\mathsf{T}_\rho(q, \nu) = \mathsf{t}$ and the suffix of run ρ initialized in $\rho(\mathsf{t})$, respectively. Run ρ produces a *word* $w = \big(L(q_0), \mathsf{T}_\rho(q_0, \nu_0')\big)\big(L(q_1), \mathsf{T}_\rho(q_1, \nu_1')\big)\ldots$. A *language* $\mathcal{L}(\mathcal{A})$ of a timed automaton $\mathcal{A}$ is a set of all words produced by all runs of $\mathcal{A}$.

Metric Interval Temporal Logic (MITL) is a specification logic for real-time systems, whose formulas are interpreted over runs of timed automata.

Definition 3.6.2 (MITL Syntax)**.** *The syntax of a MITL formula over the set of atomic propositions AP is given as follows:*

$$\phi ::= true \mid \pi \mid \neg\phi \mid \phi \wedge \phi \mid \phi \,\mathsf{U}_I\, \phi,$$

where $\pi \in AP$, and I is a non-singular[2] *interval (interval I may be also unbounded).*

Definition 3.6.3 (MITL Semantics)**.** *Given a MITL formula ϕ and a run ρ of a timed automaton $\mathcal{A}$, the satisfaction relation $\rho \models \phi$ is for formulas ϕ of form $tt \mid \pi \mid \neg\phi \mid \phi \wedge \phi$ given analogously as for LTL [AFH96]. Furthermore,*

$$\rho \models \phi \,\mathsf{U}_I\, \psi \Leftrightarrow \exists \mathsf{t} \in \mathbb{R}_{\geq 0}, \text{ such that } (\rho^{\mathsf{t}} \models \psi \wedge \mathsf{t} \in I \wedge \forall\, 0 \leq \mathsf{t}' < \mathsf{t}.(\rho^{\mathsf{t}'} \models \phi)).$$

Each MITL formula ϕ defines a language $\mathcal{L}(\phi)$ of all words produced by all runs satisfying ϕ. Note, that MITL formulas do not contain next operator, because the time domain is dense. Boolean operators $\vee$, and $\Rightarrow$ are defined in the usual way. Besides that, we define temporal operators $\mathsf{F}_I\, \phi \equiv true\, \mathsf{U}_I\, \phi$ (eventually), $\mathsf{G}_I\, \phi \equiv \neg \mathsf{F}_I\, \neg\phi$ (globally), and $\phi\, \mathsf{R}_I\, \psi \equiv \neg(\neg\phi\, \mathsf{U}_I\, \neg\psi)$ (release).

Given a timed automaton $\mathcal{A}$ and a MITL formula ϕ, the model checking question whether $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\phi)$ and the synthesis question of finding a run of $\mathcal{A}$ satisfying ϕ can be solved using automata-based approach, assuming that each end-point of each interval I that appears in the MITL formula is an integer number or unbounded. The scheme of the algorithm is as follows. First, the negation of the model-checked formula ϕ or directly the formula ϕ for control is translated into a timed automaton $\mathcal{A}_{\neg\phi}$. Then, a product timed automaton $\mathcal{A} \otimes \mathcal{A}_{\neg\phi}$ is built, such that $\mathcal{L}(\mathcal{A} \otimes \mathcal{A}_{\neg\phi}) = \mathcal{L}(A) \cap \mathcal{L}(\mathcal{A}_{\neg\phi})$. Finally, by checking emptiness of $\mathcal{L}(\mathcal{A} \otimes \mathcal{A}_{\neg\phi})$, the answer to model checking or the control synthesis problem is obtained. MITL model checking is EXPSPACE-complete [AFH96].

The translation process from a MITL formula $\neg\phi$ into timed automaton $\mathcal{A}_{\neg\phi}$ requires the intervals appearing in $\neg\phi$ to have integer bounds. Although this might seem restrictive, there is a simple way how to extend the results to deal with intervals with rational bounds. The "trick" is to pick a suitable constant $prec \in \mathbb{Q}_{>0}$ and multiply all the interval bounds appearing in $\neg\phi$ with *prec* in order to get integer interval bounds. All the constants that appear in the model checked timed automaton $\mathcal{A}$ have to be multiplied with *prec* as well.

3.7 Markov Decision Processes and Probabilistic Logics

Discrete Markov chains and Markov decision processes are both transition systems enhanced with probabilities. In Markov chains, the transition relation is,

2. singular intervals are those of form $[\mathsf{t}, \mathsf{t}]$

loosely speaking, given by a probability distribution. On the other hand, Markov decision processes combine both nondeterminism and probability. In each state of an MDP, an action is chosen nondeterministically and then the successor state is determined according to the probability distribution associated with the action.

Definition 3.7.1 (Markov Decision Process)**.** *A finite Markov Decision Process (MDP) is a tuple* $\mathcal{M} = (S, s_{\text{init}}, Act, P, AP, L)$*, where*

- S *is a finite, nonempty set of states;*
- $s_{\text{init}} \in S$ *is the initial state;*
- Act *is a finite, nonempty set of actions;*
- $P : S \times Act \times S \to [0, 1]$ *is a transition probability function, such that*

$$\forall \alpha \in Act \text{ it holds } \sum_{s' \in S} P(s, \alpha, s') \in \{0, 1\};$$

- AP *is a set of atomic propositions; and*
- $L : S \to 2^{AP}$ *is a labelling function.*

$Act(s)$ denotes the set of actions that are enabled in the state s, i.e. the set of actions $\alpha \in Act$ such that $P(s, a\ , s') > 0$ for some state $s' \in S$. For any state $s \in S$, we require that $Act(s) \neq \emptyset$.

The intuitive semantics of an MDP is as follows. If s is the current state then an action $\alpha \in Act(s)$ is chosen nondeterministically and is executed leading to a state s' with probability $P(s, \alpha, s')$. An infinite *trace* of an MDP *starting* at s_0 is a sequence $r = s_0\alpha_0 s_1\alpha_1 \ldots$ such that $P(s_i, \alpha_i, s_{i+1}) > 0$ for all $i \geq 0$. A *word* produced by r is the sequence $L(s_0)L(s_1)\ldots$ over the alphabet 2^{AP} obtained by the projection of r to the state labels.

State s is called *deterministic* if exactly one action is enabled in s. If all states of an MDP are deterministic, the MDP is called *Markov chain* and the alphabet is omitted. To resolve the nondeterminism of an MDP a *scheduler* function is used. We consider deterministic history dependent schedulers which are given by a function η assigning an action $\eta(r_{\text{pref}}) \in Act(s_n)$ to every finite trace prefix $r_{\text{pref}} = s_0\alpha_0 \ldots \alpha_{n-1}s_n$. Given a scheduler η, the behavior of $\mathcal{M}$ under η can be formalized as a Markov chain:

Definition 3.7.2. *A Markov chain of an MDP* $\mathcal{M} = (S, s_{\text{init}}, Act, P, , AP, L)$ *induced by a scheduler* η *is* $\mathcal{M}_\eta = (S^+, s_{\text{init}}, P_\eta, AP, L_\eta)$*, where*

$$P_\eta(s_0 s_1 \ldots s_n, s_0 s_1 \ldots s_n s_{n+1}) = P(s_n, \eta(s_0 s_1 \ldots s_n), s_{n+1})$$

and $L_\eta(s_0 s_1 \ldots s_n) = L(s_n)$.

Let $\mathcal{M}$ be a Markov chain, $s \in S$ be a state of $\mathcal{M}$, and $Traces$ be a set of traces of $\mathcal{M}$ starting at s. We define *the probability of the set* $Traces$ as a measure of the set $Traces$ in the set of all traces of $\mathcal{M}$ strating at s. A set $Traces$ of traces of a Markov Chain $\mathcal{M}$ is called *basic cylinder set* if there is a prefix $s_0\alpha_0 \dots \alpha_{n-1}s_n$ such that $Traces$ contains exactly all traces of $\mathcal{M}$ with that prefix. The probability measure of a basic cylinder set with prefix $s_0\alpha_0 \dots \alpha_{n-1}s_n$ is then $\prod_{i=0}^{n-1} P(s_i, \alpha_i, s_{i+1})$. If the set $Traces$ of traces of $\mathcal{M}$ is not a basic cylinder set, its measure is determined as a sum of measures of maximal (w.r.t. inclusion) basic cylinder sets fully contained in $Traces$.

Probabilistic Linear Temporal Logic

The syntax of probabilistic LTL remains exactly same as defined for the transition systems in Def. 3.3.1. The semantics of LTL are defined over infinite traces of an MDP $\mathcal{M} = (S, s_{\text{init}}, Act, P, AP, L)$ in the same way as for the transition systems, however, it is the model-checking and the control strategy synthesis questions that are posed differently. Further details are beyond the scope of this thesis and can be found *e.g.,* in [BK08].

Probabilistic Computation Tree Logic

LTL formulae themselves do not contain any constraints on probabilities. In Probabilistic Computation Tree Logic (PCTL), the probabilistic requirements are built directly in its formulae. Whereas LTL captures properties of individual paths originating at a certain state, PCTL is a branching-time logic. Similarly, as in nonprobabilistic setting, LTL and PCTL are incomparable.

PCTL *state formulae* over the set of atomic propositions AP are defined inductivelly:

$$\phi ::= true \mid \pi \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \mathsf{P}_{\bowtie p}(\varphi)$$

where $\pi \in AP$, $true$ is a predicate true in each state of the system, φ is a PCTL path formula, $\bowtie \in \{<, \leq, >, \geq\}$ and $p \in [0, 1]$ is a rational bound. PCTL *path formulae* are formed according to the grammar:

$$\varphi ::= \mathsf{X}\phi \mid \phi_1 \mathsf{U} \phi_2 \mid \phi_1 \mathsf{U}^{\leq n} \phi_2$$

where ϕ, ϕ_1 and ϕ_2 are PCTL state formulae and $n \in \mathbb{N}$.

Intuitivelly, state formulae express properties of states and path formulae specify properties of paths. For both of them, we define the satisfaction relation $\models$. Let us consider an MDP $\mathcal{M} = (S, s_{\text{init}}, Act, P, AP, L)$, a state $s \in S$, a trace

$r = s_0\alpha_0 s_1\alpha_1 s_2\alpha_2 \ldots$ in $\mathcal{M}$, state formulae ϕ, ϕ_1, ϕ_2 and a path formula φ.

$$
\begin{array}{ll}
s \models true & \text{always} \\
s \models \pi & \iff \pi \in L(s) \\
s \models \neg\phi & \iff s \not\models \phi \\
s \models \phi_1 \wedge \phi_2 & \iff s \models \phi_1 \wedge s \models \phi_2 \\
s \models \mathsf{P}_{\bowtie p}(\varphi) & \iff \text{for all schedulers } \eta \text{ for } \mathcal{M}.\, Pr_\eta(s \models \varphi) \bowtie p
\end{array}
$$

where $Pr_\eta(s \models \varphi)$ denotes probability of the set of paths originating at s and satisfying a path formula φ under the scheduler η.

$$
\begin{array}{ll}
r \models \varphi & \iff r(0) \models \varphi \\
r \models \neg\phi & \iff r \not\models \phi \\
r \models \phi_1 \wedge \phi_2 & \iff r \models \phi_1 \wedge \pi \models \phi_2 \\
r \models \mathsf{X}\phi & \iff r(1) \models \phi \\
r \models \phi_1 \mathsf{U} \phi_2 & \iff \exists k.\, r^k \models \phi_2 \text{ and } \forall 0 \leq j < k.\, r^j \models \phi_1 \\
r \models \phi_1 \mathsf{U}^{\leq n} \phi_2 & \iff \exists 0 \leq k \leq n.\, r^k \models \phi_2 \text{ and } \forall 0 \leq j < k.\, r^j \models \phi_1
\end{array}
$$

Similarly as in LTL, we can define derived PCTL path operators F and G. We can observe that in a formula with nested operators, the state operator $\mathsf{P}_{\bowtie p}$ must alternate with the path operators. Thus, for example $\mathsf{FG}\phi$ is not a correctly defined PCTL formula. On the other hand $\mathsf{P}_{\leq 0.5}\ \mathsf{F}(\mathsf{P}_{\geq 0.1}\ \mathsf{G}\phi)$ is.

A notable extension to PCTL is PCTL*. In PCTL* a path formula does not need to be immediatelly preceded by the state operator $\mathsf{P}_{\bowtie p}$. This causes that any LTL formula φ is a PCTL* formula as well.

Chapter 4

Formal Methods for Systems with Degradation

This chapter is devoted to modeling, specification, model checking, and control strategy synthesis methods for systems with degradation. After a brief motivation and an informal introduction of systems that inherently incorporate a degrading quality in Section 4.1, we define Transition Systems with Degradation (TSD) as their modeling formalism and a temporal logic for specification of their quantitative properties, called LTL with Degradation Constraints (DLTL) in Section 4.2 and Section 4.3, respectively.

In Section 4.4 we focus on model checking and control strategy synthesis for systems with degradation. First, the syntax of LTL with degradation constraints resembles the syntax of Metric Interval Temporal Logic (MITL) whose formulas are interpreted over timed automata. Therefore, in Section 4.4.2 we investigate similarities between transition systems with degradation and DLTL, and timed automata and MITL, respectively. We show how satisfying trace synthesis problem for the earlier can be solved with the use of existing techniques for automata-based model checking of the latter, with limited, yet arbitrary precision. Second, in Section 4.4.3 we present a full-precision satisfying trace synthesis method for systems with degradation that leverages ideas from the very well known automata-based approach to model checking. In particular, we introduce an extension of Büchi automata, called Büchi Automata with Degradation Constraints (BADC) and an algorithm to translate a BADC into its normal form. A finite product automaton of a TSD and a normalized BADC can then be built. As the product automaton is a standard Büchi automaton, the answer to the model checking problem as well as the answer to the control strategy synthesis problem can be obtained via application of the well-known graph algorithms for checking the product automaton language non-emptiness. Third, Section 4.4.4 focuses on solving the satisfying trace synthesis problem with DLTL specifications through the translation of DLTL formulas into BADCs. We suggest a translating algorithm for a subclass of DLTL, which we call half-bounded DLTL formulas. Thus, for this half-bounded fragment, we complete a full-precision automata-based satisfying trace synthesis method.

Section 4.5 shows that probability can be interpreted as a degrading quality

and that an MDP can be interpreted as a transition system with degradation. We show that DLTL can distinguish two MDPs that are not distinguishable by any LTL, PCTL or even PCTL* formula. Hence, we prove that the expressive power of DLTL is incomparable with the expressive power of the common logics used to reason about discrete Markov decision processes.

Finally, we conclude and outline directions for future research in Section 4.6.

4.1 Motivation

Degradation, sometimes also called decay, is a natural phenomenon present in many systems that we encounter in our everyday lives. For instance, the value of money degrades in time due to inflation. Degradation occurs in nature, too; the examples include the process of radioctive decay as well as biodegradation. Finally, the degradation phenomenon is inherently present in computer-based systems. For example, data stored in memory are subject to bit rot, signal strength degrades with the distance from the transmitter, capacity of a recharging battery pack degrades with every charging cycle, etc.

A number of computer-based systems developed must take the degradation phenomenon into account. Verification of the worst-case degradation scenarios that a software system under development must survive or designing a strategy to avoid a quality decaying below a given threshold are examples of problems that can be addressed with the model checking and the control strategy synthesis approaches.

Example 4.1.1 (Broadcasting network)**.** *Let us consider a broadcasting network with a start point* `s` *(sender) and an end point* `e` *(receiver). Unfortunately, the points are too far from each other, so the signal that is being broadcast from the sender cannot reach the receiver without being unrepairably damaged. A possible solution to the problem is to build relays in between* `s` *and* `e` *that can fully restore the original quality of the signal. Let us assume that we have a map of possible places where a relay may be built including pairwise signal degradation values as illustrated in Figure 4.1 and that several relays have been built, placed in the points marked with* `a`*.*

The properties we are interested in include: "Does the signal reach the target point in a proper shape if the relays are built at the `a`*-points?" or "If yes, can we place fewer* `a`*-points than the two while ensuring the signal strength always being within admissible bounds?".*

Example 4.1.2 (Data Gathering Robot I)**.** *For an example of a robotic system with degradation, consider a data gathering robot in an environment as illustrated in Figure 4.2. A mobile robot is initially placed in the region labeled with* g_1 *and*

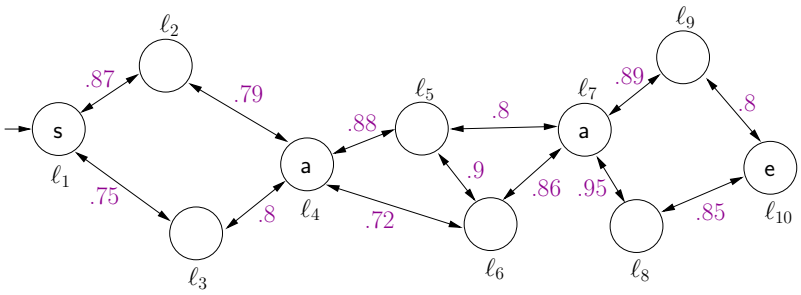


Figure 4.1: An example of signal coverage map for a broadcasting network. Each circle in the map represents a geographical location. The one labeled with S is the sender, the one labeled with E corresponds to the end point – the receiver. The labels of the bidirectional arrows in between the states represent the level of the signal strength degradation between the respective adjacent states. For instance, the signal degrades to 87% of its original strength in between ℓ_1 and ℓ_2 and to 79% between ℓ_2 and ℓ_4.

it can traverse the environment, while gathering data in the data gather states g_1 *and* g_2 *and uploading the data in the data upload states* u_1 *and* u_2*. The robot is equipped with a fast, cheap, volatile memory that stores the collected data in capacitors within an integrated circuit. Since the capacitors leak charge, the data gathered in the data gather locations gradually decay.*

An example of a task we wish to solve is to find a path of the robot that periodically visits both data gather and data upload locations and ensures that after the data are gathered, they are uploaded before decaying below certain threshold.

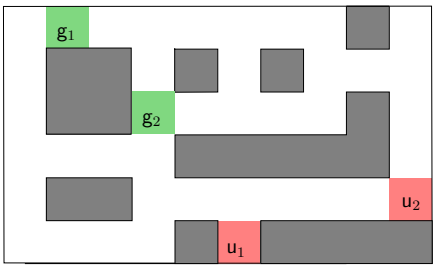


Figure 4.2: An example of a data gathering robotic system. The figure illustrates an environment with two data gather locations (g_1, and g_2) and two data upload locations (u_1, and u_2). Initially, the robot is placed in g_1. In intersections, one of the actions n (go north), s (go south), w (go west), or e (go east) can be applied. When an action is chosen, the robot turns the corresponding direction and further simply follows the path until another intersection, a data gather location, or a data upload location is reached.

4.2 Modeling Systems with Degradation

In this section we introduce a *Transition Systems with Degradation* (TSD) as a modeling formalism for systems with degradation. Informally, a TSD is a labeled transition system that is enhanced with a degradation constant associated with every transition. A degradation constant is a rational number from interval $(0, 1]$ and it determines the effect of the transition execution on the system attribute that is subject to degradation. To reflect possibly different effects of different transitions, the degradation constants may attain different values.

Definition 4.2.1 (Transition System with Degradation)**.** *A transition system with degradation (TSD) is a tuple* $\mathcal{T} = (S, S_{\text{init}}, Act, T, AP, L, D)$*, where*

- S *is a finite set of states;*
- $S_{\text{init}} \subseteq S$ *is the set of initial states;*
- Act *is a finite set of actions;*
- $T \subseteq S \times Act \times S$ *is a transition relation;*
- AP *is a set of atomic propositions;*
- $L : S \to 2^{AP}$ *is a labeling function; and*
- $D : T \to (0, 1]$ *is a degradation relation.*

A transition $t = (s, \alpha, s') \in T$ represents that the system can make a transition from the state s to the state s' under the action $\alpha \in Act$. The degradation constant $D(t)$ associated with the transition t determines the effect of the transition on the current level of degradation, *i.e.,* it represents the fraction to which the degrading attributes drops when the transition t is executed. If $D(t) = 1$ the level of quality is unchanged, if $D(t) = 0.75$ the level of quality is decreased to 75% of the level of quality at the moment before the transition was executed. In other words, if the level of degradation is d at state s, then after the execution of $t = (s, \alpha, s')$, the level of degradation at state s' is $d \cdot D(t)$.

A *trace* of a TSD $\mathcal{T} = (S, S_{\text{init}}, Act, T, AP, L, D)$ is an infinite sequence $r = s_0\alpha_0 s_1\alpha_1 \ldots$, where $s_0 \in S_{\text{init}}$, $s_i \in S$ and $t_i = (s_i, \alpha_i, s_{i+1}) \in T$, for all $i \geq 0$. As expected, we denote by $r(i)$ and r^i the (i+1)-st state of the trace r (*i.e.*, s_i) and the suffix beginning in $r(i)$, respectively. A *word* produced by a trace $r = s_0\alpha_0 s_1\alpha_1 \ldots$ is a sequence $w(r) = (L(s_0), d_0)(L(s_1), d_1) \ldots$ over $(2^{AP} \times (0, 1])^\omega$, where $d_i = D\big((s_i, \alpha_i, s_{i+1})\big)$.

Definition 4.2.2 (Level of Degradation)**.** *A level of degradation* $\mathbb{D}_r(i)$ *on the trace* $r = s_0\alpha_0 s_1\alpha_1 \ldots$ *up to the state* s_i *is defined as a product of all degradation*

constants associated with the transitions along this trace

$$\mathbb{D}_r(i) = \prod_{j=0}^{i-1} D\big((s_j, \alpha_j, s_{j+1})\big) \cdot 1.$$

Example 4.2.3 (Broadcasting Network)**.** *The signal coverage map as depicted in Figure 4.1 is a simplified illustration of a transition system with degradation. The set of states is* $\{\ell_1, \ldots, \ell_{10}\}$*, the set of initial states is* $\{\ell_1\}$ *and the set of actions is* $Act = \{\alpha\}$*. The transitions are illustrated as bidirectional arrows where the action labels are for the simplicity of presentation omitted from the figure. The set of atomic propositions is* $AP = \{\mathsf{s}, \mathsf{e}, \mathsf{a}\}$ *and the labeling function is defined as* $L(\ell_1) = \{\mathsf{s}\}, L(\ell_{10}) = \{\mathsf{e}\}, L(\ell_4) = L(\ell_7) = \{\mathsf{a}\}$*, and* $L(\ell_i) = \emptyset$*, for all* $i \notin \{1, 4, 7, 10\}$*. An example of a trace of the TSD is* $r = \ell_1 \alpha \ell_2 \alpha \ell_4 \alpha \ell_6 \alpha \ell_7 \ldots$*. The level of degradation on this trace up to the state* $r(3) = \ell_6$ *is* $\mathbb{D}_r(3) = 0.87 \cdot 0.79 \cdot 0.72$*.*

Example 4.2.4 (Data Gathering Robot I)**.** *The transition system with degradation for the data gathering robot is illustrated in Figure 4.3. An example of trace of the TSD is* $r = s_1 e s_2 s s_9 s s_8 w s_7 \ldots$*. The level of degradation on this trace up to the state* $r(2) = s_9$ *is* $\mathbb{D}_r(2) = 0.98 \cdot 0.98 = 0.9604$*.*

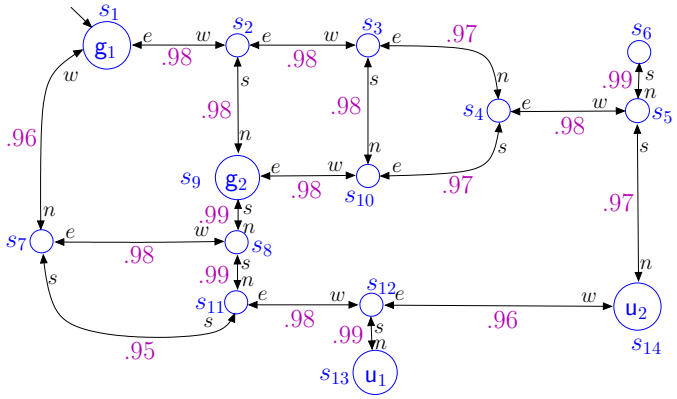


Figure 4.3: The figure illustrates the transition system with degradation for the data gathering robot from example 4.1.2. For simplicity, we illustrate two transitions representing connection of two regions as a bidirectional arrow. Each bidirectional arrow is thus labeled with two actions, one for each direction. In each state, the closer one of the two labels represents the action that can be applied in that state. For instance, $(s_1, e, s_2) \in T$, and $(s_2, w, s_1) \in T$. Each transition is also labeled with a degradation constant. States are labeled with subsets of atomic propositions of the set $AP = \{\mathsf{g}_1, \mathsf{g}_2, \mathsf{u}_1, \mathsf{u}_2\}$. Namely, $L(s_1) = \{\mathsf{g}_1\}$, $L(s_9) = \{\mathsf{g}_2\}$, $L(s_2) = \emptyset$, etc.

4.3 Temporal Logic for Systems with Degradation

In this section, we present a specification formalism for systems with degradation. Namely, we propose *Linear Temporal Logic with Degradation Constraints* (DLTL) that allows for specification of quantitative properties of systems with degradation involving constraints on the level of degradation between certain points on the system trace. The main idea is to augment LTL with a quantitative operator, similarly as it is done in PCTL for Markov chains and Markov decision process (see Section 3.7) or in MITL for timed automata (see Section 3.6).

Definition 4.3.1 (DLTL Syntax)**.** *Let $\pi \in AP$, and I be an interval within $(0, 1]$. The syntax of a DLTL formula over the set of atomic propositions AP is given according to the following rules:*

$$\phi ::= \mathit{true} \mid \pi \mid \neg\phi \mid \phi \wedge \phi \mid \mathsf{X}_I\, \phi \mid \phi\, \mathsf{U}_I\, \phi,$$

where true is a predicate that is always true, $\pi \in AP$, $\neg$ (negation) and $\wedge$ (conjunction) are Boolean operators and X_I (constrained next) and U_I (constrained until) are temporal operators with degradation constraints.

Although the syntax of DLTL strongly resembles the syntax of MITL for timed automata, the two logics differ in their semantics as they are interpreted over significantly different models. Their resemblance will be investigated shortly in Section 4.4.2. Similarly as LTL formulas are interpreted over traces of a transition system and MITL formulas are interpreted over runs of timed automata, DLTL formulas are interpreted over traces of transition systems with degradation.

Definition 4.3.2 (DLTL Semantics)**.** *Let $r = s_0\alpha_0 s_1\alpha_1 \ldots$ be a trace of a TSD $\mathcal{T}$. DLTL semantics is defined through the satisfaction relation $\models$.*

$$\begin{aligned}
&r \models \mathit{true} && \text{always}\\
&r \models \pi && \iff \pi \in L(s_0)\\
&r \models \neg\phi && \iff r \not\models \phi\\
&r \models \phi \wedge \psi && \iff r \models \phi \text{ and } r \models \psi\\
&r \models \mathsf{X}_I\, \phi && \iff r^1 \models \phi \wedge \mathbb{D}_r(1) \in I\\
&r \models \phi\, \mathsf{U}_I\, \psi && \iff \exists i \geq 0.\, r^i \models \psi \text{ and } \mathbb{D}_r(i) \in I, \text{ and } \forall\, 0 \leq j < i.\, r^j \models \phi
\end{aligned}$$

Intuitively, $\mathsf{X}_I\, \phi$ states that the formula ϕ has to hold in the next state of the trace and that the level of degradation after executing the first transition of the trace falls within interval I. The formula $\phi\, \mathsf{U}_I\, \psi$ states that ϕ holds true

until the moment when ψ holds true and the level of degradation at this moment is within I.

The standard LTL operators X, and U are included in DLTL as $\mathsf{X}_{(0,1]}$, and $\mathsf{U}_{(0,1]}$, respectively. Other boolean operators such as $\vee$ (disjunction), and $\Rightarrow$ (implication) are defined in the expected way. In addition to that, we also define three useful operators:

$$\begin{aligned} \mathsf{F}_I\,\phi &\equiv \mathit{true}\,\mathsf{U}_I\,\phi && (\textit{constrained eventually}) \\ \mathsf{G}_I\,\phi &\equiv \neg\mathsf{F}_I\,\neg\phi && (\textit{constrained globally}) \\ \phi\,\mathsf{R}_I\,\psi &\equiv \neg(\neg\phi\,\mathsf{U}_I\,\neg\psi) && (\textit{constrained release}) \end{aligned}$$

and their non-constrained versions $\mathsf{F}\,\phi \equiv \mathit{true}\,\mathsf{U}\,\phi$ *(eventually)*, $\mathsf{G}\,\phi \equiv \neg\mathsf{F}\,\neg\phi$ *(globally)*, and $\phi\,\mathsf{R}\,\psi \equiv \neg(\neg\phi\,\mathsf{U}\,\neg\psi)$ *(release)*. Similarly as LTL formulas, DLTL formulas can be *normalized*, *i.e.,* based on the above equations, it can be transformed into a form, where all negations are applied only directly to atomic propositions.

Example 4.3.3 (Broadcasting Network)**.** *An example of a quantitative linear temporal property for our broadcasting network system from Example 4.1.1 is "Assuming that the signal is fully restored in* a*-points, the signal eventually reaches the receiver* e *while its strength does not decrease below* 90% *of its original quality." expressed as a DLTL formula*

$$\mathsf{F}\,\mathsf{e} \wedge \mathsf{G}\big((\mathsf{s} \vee \mathsf{a}) \Rightarrow \mathsf{F}_{[0.9,1]}(\mathsf{a} \vee \mathsf{e})\big).$$

Example 4.3.4 (Data Gathering Robot I)**.** *In our robot data gathering system (Example 4.1.2), a property "Data are gathered in state* g_1*. The data are moved to an upload state before their integrity drops bellow 90% and meanwhile, no data are gathered in state* g_2*." can be expressed as a DLTL formula*

$$\mathsf{g}_1 \wedge \neg\mathsf{g}_2\ \mathsf{U}_{[0.9,1]}\ (\mathsf{u}_1 \vee \mathsf{u}_2).$$

A bit more complicated property for our robot data gathering system is for instance: "Periodically visit both data gathering states. Always upload the data in one of the upload states before their integrity drops bellow 90%." A DLTL formula expressing this behavior is

$$\mathsf{GF}\,\mathsf{g}_1 \wedge \mathsf{GF}\,\mathsf{g}_2 \wedge \mathsf{G}\big(\mathsf{g}_1 \vee \mathsf{g}_2 \rightarrow \mathsf{F}_{[0.9,1]}(\mathsf{u}_1 \vee \mathsf{u}_2)\big).$$

4.4 Model Checking and Control Strategy Synthesis

4.4.1 Problem Statement

Given a transition system with degradation and a DLTL formula, our goal is to address two problems: the model checking problem, and if the TSD is deter-

ministic then also the control strategy synthesis problem. In both cases, the key question to answer is the variant of satisfying trace synthesis problem (Problem 3.5.1).

Problem Statement 4.4.1 (Satisfying Trace Synthesis)**.** *Given a TSD $\mathcal{T}$ and a property specified by a DLTL formula ϕ, find a trace of $\mathcal{T}$ that satisfies ϕ, if such a trace exist.*

4.4.2 Problem Solution: Timed Automata Approach

As the resemblance between DLTL and MITL is noticable, in this section, we investigate whether the satisfying trace synthesis problem (Problem 4.4.1) can be addressed exploiting the already developed model checking techniques for timed automata. Particularly, the problem we are to solve is as follows.

Problem Statement 4.4.2 (DLTL to MITL Translation)**.** *Given a TSD $\mathcal{T}$ and an DLTL formula φ, find a timed automaton $\mathcal{A}$ and a MITL formula ϑ, such that $\mathcal{T} \models \varphi \iff \mathcal{A} \models \vartheta$.*

In other words, we approach the DLTL satisfying trace synthesis problem for systems with degradation via its conversion into the model checking problem for timed automata and MITL formulas. During the conversion process, two major differences have to be overcome:

(1) in systems with degradation, the degradation decreases along the *transitions*, whereas in timed systems, the time passes in the *states*, and

(2) the degradation constants are meant to be *multiplied*, whereas time passes in *additive* fashion.

Loosely speaking, we address the first one by modeling transitions of a TSD as states of a timed automaton and the second one by applying logarithm to the degradation constants appearing in both the TSD and the DLTL formula while building on the fact that $\log a \cdot b = \log a + \log b$, for each $a, b \in \mathbb{R}_{\geq 0}$.

To simplify the presentation of the translation process, we impose the following assumption on the DLTL formulas. We discuss how to deal with full DLTL later in Remark 4.4.11.

Assumption 4.4.3. *Assume that the given DLTL formula ϕ sastisfies two additional assumptions: (1) the intervals that appear in ϕ are non-singular, and (2) ϕ does not contain any next operator X or X_I.*

Furthermore, let us assume the following.

Assumption 4.4.4. *Assume that there exist a logarithm base $k \in (0,1)$, such that $\log_k c$ is a rational number for each constant c that appears in formula ϕ.*

Assumption 4.4.4 is quite restrictive as such constant k often does not exist. We will explain the means and the cost of relaxing this assumption shortly.

Let us now present the translation process in details. First, we preprocess the given transition system with degradation $\mathcal{T} = (S, S_{\text{init}}, Act, T, AP, L, D)$ into a TSD $\mathcal{T}' = (S', S'_{\text{init}}, Act', T', AP', L', D')$ this way:

- $S' = S \cup (S \times T) \cup (T \times S)$;
- $S'_{\text{init}} = S_{\text{init}}$;
- $Act' = Act \cup \{\epsilon\}$, where $\epsilon \notin Act$;
- $T' = \{\big(s_1, \alpha, (s_1, t)\big), \big((s_1, t), \epsilon, (t, s_2)\big), \big((t, s_2), \epsilon, s_2\big) | \, t = (s_1, \alpha, s_2) \in T\}$;
- $AP' = AP \cup \{\pi_\epsilon\}$, where $\pi_\epsilon \notin AP$;
- $L'(s) = L(s)$ for all $s \in S$, and $L'(s) = \{\pi_\epsilon\}$ for all $s \in S' \setminus S$; and
- $D'(t) = 1$ for all transitions leading from and to some $s \in S$, and $D'\big((s_1, t), \epsilon, (t, s_2)\big) = D(t)$ for the rest of the transitions.

Second, we convert the given normalized DLTL formula ϕ into ϕ' by replacing each *non-negated* occurrence of atomic proposition π with $\pi_\epsilon \mathsf{U} \pi$ and each *negated* occurrence of π with $\pi_\epsilon \mathsf{U} \neg\pi$. This way, we manage to "ignore" the newly added states corresponding to the transitions of $\mathcal{T}$.

Lemma 4.4.5. *There exists a trace r of $\mathcal{T}$, such that $r \models \phi$ if and only if there exists a trace r' of $\mathcal{T}'$, such that $r' \models \phi'$.*

Proof. Each trace $r = s_0\alpha_0 s_1 \alpha_1 s_2 \ldots$ of the TSD $\mathcal{T}$ that produces a word $w(r) = (L(s_0), d_0)(L(s_1), d_1)(L(s_2), d_2)\ldots$ maps to a single run r' of $\mathcal{T}'$ producing word $w(r') = (L(s_0), d_0)(\pi_\epsilon, 1)(\pi_\epsilon, 1)(L(s_1), d_1)(\pi_\epsilon, 1)(\pi_\epsilon, 1)(L(s_2), d_2)\ldots$. It is easy to show by induction, that for all $i \geq 2$ it holds that $r^i \models \phi$ if and only if $r'^{3i-4} \models \phi' \wedge r'^{3i-3} \models \phi' \wedge r'^{3i-2} \models \phi'$. Finally, we get $r \models \phi \iff r' \models \phi'$. □

Given a TSD $\mathcal{T}$ and the corresponding TSD $\mathcal{T}'$, we build a timed automaton $\mathcal{A} = (S \cup T, S'_{\text{init}}, Act', \{x\}, \delta, Inv, AP', L_{\mathcal{A}})$, where

- $\delta = \{\big(s_1, x = 0, \alpha, \emptyset, t\big), \big(t, x = \log D(t), \epsilon, \{x\}, s_2\big) \mid t = (s_1, \alpha, s_2) \in T\}$,
- $Inv(s) = x \leq 0$ for all $s \in S$, and $Inv(t) = x \leq \log_k D(t)$ for all $t \in T$,
- $L_{\mathcal{A}}(s) = L(s)$ for all $s \in S$, and $L_{\mathcal{A}}(t) = \{\pi_\epsilon\}$ for all $t \in T$,

such that k is a logarithm base complying with Assumption 4.4.4, *i.e.,* such that $\log_k c$ is a rational number for each degradation constant c that appears in ϕ. Note that $\log_k x > 0$ for all $x \in (0, 1]$. For simplicity, let log denote in further text $\log_k$.

A DLTL formula ϕ' is transformed into a MITL formula ϑ and the rest of the solution to satisfying trace synthesis is as follows. Each occurrence of interval (a, b) is replaced with $(\log b, \log a)$, and analogously, each occurrence of $(a, b]$, $[a, b]$, and $[a, b)$ is replaced with $[\log b, \log a)$, $[\log b, \log a]$, and $(\log b, \log a]$, respectively. In case $a = 0$, we use ∞ instead of $\log a$. The rest of the formula remains the same. We pick a suitable constant $prec \in \mathbb{Q}_{>0}$ and multiply all the constants both in $\mathcal{A}$ and ϑ with *prec* in order to make all the interval bounds appearing in formula ϑ integer. Formula ϑ can now be translated into a timed automaton $\mathcal{A}_\vartheta$ [AFH96]. The rest is the well-known checking of language emptiness for product timed automaton $\mathcal{A} \otimes \mathcal{A}_\vartheta$ of the timed automata $\mathcal{A}$ and $\mathcal{A}_\vartheta$ [AD94]. If there exists an accepting run of $\mathcal{A} \otimes \mathcal{A}_\vartheta$, its projection to $\mathcal{A}$ and then to $\mathcal{T}$ gives a counter-example for the case model-checking was our aim, or a desired trace for the case control strategy synthesis was our interest.

Lemma 4.4.6. *There exists a trace r' of $\mathcal{T}'$, such that $r' \models \phi'$ if and only if there exists a run ρ of $\mathcal{A}$, such that $\rho \models \vartheta$.*

Proof. Follows directly from the structure of $\mathcal{T}'$, construction of $\mathcal{A}$, and the fact, that $\log(a \cdot b) = \log a + \log b$, and $0 < a \leq b \leq 1 \Rightarrow 0 \leq \log b \leq \log a < \infty$. □

Corollary 4.4.7. *There exists a trace r of $\mathcal{T}$, such that $r \models \phi$ if and only if there exists a run ρ of $\mathcal{A}$, such that $\rho \models \vartheta$.*

Example 4.4.8 (TSD to TA Translation)**.** *An example of a transformation of a TSD $\mathcal{T}$ into a TSD $\mathcal{T}'$ and into a timed automaton $\mathcal{A}$ is illustrated in Figure 4.4.*

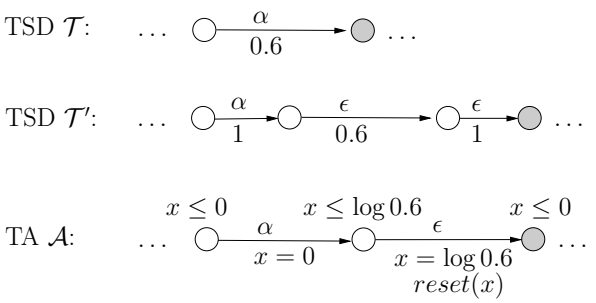


Figure 4.4: An example of edge transformation of a TSD $\mathcal{T}$.

So far, we have solved Problem 4.4.1 under Assumptions 4.4.3 and 4.4.4. Let us now focus on relaxing the latter assumption. In many cases it is not possible to find k, such that $\log_k c$ is a rational number for each constant c that appears in ϕ. Therefore, necessarily, some kind of approximation is needed.

Lemma 4.4.9. *Consider intervals I and I', such that interval I' is within I. For any run ρ, it holds that $\rho \models \phi \,\mathsf{U}_{I'}\, \psi \Rightarrow \phi \,\mathsf{U}_I\, \psi$, and dually, $\rho \models \phi \,\mathsf{R}_I\, \psi \Rightarrow \phi \,\mathsf{R}_{I'}\, \psi$.*

Proof. Directly from expanding definition of U_I and R_I, respectively. □

Based on Corollary 4.4.7 and Lemma 4.4.9, the satisfying trace synthesis procedure can be summarized as follows. Consider a TSD $\mathcal{T}$ and let ϕ be the negation of a desired property for the case of model checking or directly the desired property if control strategy synthesis is of our interest.

(i) Transform the TSD $\mathcal{T}$ into a timed automaton $\mathcal{A}$ and the DLTL formula ϕ into a formula ϑ as described above. Transform ϑ into its normal form.

(ii) Pick a precision constant $prec \in \mathbb{Q}_{>0}$, and multiply all constants in $\mathcal{A}$ and in ϑ with $prec$.

(iii) In each U_I operator in ϑ replace left bound a and right bound b of interval I with $\lceil a \rceil$ and $\lfloor b \rfloor$, respectively. Dually for each R_I operator.

(iv) Translate ϑ into $\mathcal{A}_\vartheta$ and check, whether $\mathcal{L}(\mathcal{A}) \cap \mathcal{L}(\mathcal{A}_\vartheta) = \emptyset$. If no, then there exists a trace of $\mathcal{T}$ satisfying ϕ and the answer is found, otherwise continue on line (v).

(v) In each U_I operator in of ϑ replace left bound a and right bound b of interval I with $\lfloor a \rfloor$ and $\lceil b \rceil$, respectively. Dually for each R_I operator.

(vi) Translate ϑ into $\mathcal{A}_\vartheta$ and check, whether $\mathcal{L}(\mathcal{A}) \cap \mathcal{L}(\mathcal{A}_\vartheta) = \emptyset$. If yes, then there does not exist a trace of $\mathcal{T}$ satisfying ϕ and the procedure terminates. Qtherwise pick a precision constant $prec' > prec$ and repeat the procedure from line (iii) with $prec = prec'$.

Correctness and Complexity

Theorem 4.4.10. *If the suggested algorithm terminates, then the answer is a solution to the satisfying trace synthesis problem (Problem 4.4.1).*

Proof. The proof follows directly from Corollary 4.4.7 and Lemma 4.4.9. □

Although the algorithm is sound, its termination is not guaranteed and thus we can only answer the satisfying trace synthesis question with limited, yet arbitrary precision. The price paid for increasing precision is rapidly increasing computational demands. The size of timed automaton $\mathcal{A}_\vartheta$ is $\mathcal{O}(2^{N \cdot K})$ with $N \cdot K$

clocks, where N is the number of atomic propositions, boolean, and temporal operators in ϑ and $K-1$ is the largest integer constant in ϑ [AFH96]. Higher precision causes increase of the constant K, and hence also significant increase of the size of $\mathcal{A}_\vartheta$.

Remark 4.4.11. *For the sake of presentation simplicity, we assumed continuous semantics of timed automata, although the dynamics of a TSD is purely discrete. Therefore we had to restrict DLTL formulas not to contain next operators and singular intervals. In order to solve the model checking problem for full DLTL, one can consider discrete semantics of timed automata and Metric Temporal Logic (which is MITL including singular intervals). The approach is analogous, but approximation is needed not only in the formulas, but in the timed automaton as well. The complexity of emptiness checking of $\mathcal{L}(\mathcal{A}) \cap \mathcal{L}(\vartheta)$ remains EXPSPACE-complete and dependent on the size of constants appearing in ϑ [AFH96].*

4.4.3 Problem Solution: Automata-like Specification

In the previous subsection, we have introduced a limited-precision and quite expensive method for model checking of DLTL formulas and deterministic control strategy synthesis. Here, we focus on the full-precision method to address the satisfying trace synthesis problem as defined in Problem 4.4.1. Particularly, we start with defining Büchi automata with degradation constraints, an automata-like structure that is able to capture quantitative linear temporal properties of systems with degradation and at the same time provides a starting point for the direct automata-based approach to model checking of systems with degradation.

Büchi Automata with Degradation Constraints

Büchi Automata with Degradation Constraints are the standard Büchi automata that are enriched with a finite set of rational-domain variables (*degradation variables*), used to capture the level of degradation during the system execution. Furthermore, transitions of BADCs are enriched with the so-called *resets* of degradation variables and they are guarded with *degradation constraints*. Informally, the role of a degradation variable is to measure the level of degradation since the last time the variable was reset. Hence, note that the range of every degradation variable is $(0, 1]$. Likewise the standard Büchi automata, BADCs recognize sets of words over a given set of atomic propositions. They can be thus used to express desired or invalid properties of traces of TSDs.

In order to define BADCs formally, we first give definition of degradation constraints.

Definition 4.4.12 (Degradation Constraint)**.** *A degradation constraint γ over the set of degradation variables X is constructed according to the following rules:*

$$\gamma ::= x \bowtie d \mid d \bowtie d \mid \gamma \wedge \gamma,$$

where $\bowtie \in \{<, \leq, >, \geq\}$, $x \in X$, and $d \in (0, 1]$ is a rational number.

Degradation valuation ν is a function that assigns every degradation variable a value from $(0, 1]$, *i.e.* $\nu : X \to (0, 1]$. The set of all possible degradation valuations and the set of all degradation constraints is denoted by $Eval(X)$ and $DC(X)$, respectively. We also use $\gg$ to denote $>$ or $\geq$, and $\ll$ to denote $<$ or $\leq$. We define negation of a degradation constraint $\neg(x \bowtie d)$ as follows: $\neg(x \leq d) = x > d$, $\neg(x < d) = x \geq d$, $\neg(x \geq d) = x < d$, and $\neg(x > d) = x \leq d$.

Definition 4.4.13 (BADC)**.** *A Büchi Automaton with Degradation Constraints (BADC) is a tuple $\mathcal{B} = (Q, Q_{\text{init}}, \Sigma, X, \delta, F)$, where*

- *Q is a finite, nonempty set of states,*
- *$Q_{\text{init}} \subseteq Q$ is an initial state,*
- *Σ is a finite alphabet,*
- *X is a finite set of degradation variables,*
- *$\delta \subseteq Q \times \Sigma \times DC(X) \times 2^X \times Q$ is a transition relation,*
- *$F \subseteq Q$ is a finite set of states (so-called Büchi accepting condition).*

A 5-tuple $\tau = (q, \sigma, \gamma, R, q') \in \delta$ corresponds to a transition from state q to q' labeled with σ that is enabled if constraint $constraint(\tau) = \gamma$ is satisfied. $reset(\tau) = R$ then denotes the subset of degradation variables that are reset to 1 when the transition is executed.

The semantics of Büchi automata with degradation constraints are defined over infinite input words from $(\Sigma \times (0, 1])^\omega$, *i.e.,* when $\Sigma = 2^{AP}$ then the semantics are defined over traces of a TSD.

Definition 4.4.14 (Transition System Semantics of BADC)**.** *The semantics of a BADC $\mathcal{B} = (Q, Q_{\text{init}}, \Sigma, \delta, X, F)$ is given by a (possibly infinite) labeled transition system $\mathcal{S}(\mathcal{B}) = (S, S_{\text{init}}, Act, T)$, where*

- *$S = Q \times Eval(X)$;*
- *$S_{\text{init}} = \{(q, \nu) \mid q \in Q_{\text{init}}, \nu(x) = 1$ for all $x \in X\}$;*
- *$Act = \Sigma \times (0, 1]$;*

- $T \subseteq S \times Act \times S$. $\big((q,\nu),(\sigma,d),(q',\nu')\big) \in T$ *if and only if there is a transition* $(q,\sigma,\gamma,R,q') \in \delta$, *such that*
  - ν *satisfies constraint* γ
  - $\nu'(x) = \begin{cases} d & \text{if } x \in R \\ \nu(x) \cdot d & \text{otherwise.} \end{cases}$

A *run* of a BADC over a word $\sigma = (\sigma_0,d_0)(\sigma_1,d_1)\ldots \in (\Sigma \times (0,1])^\omega$ is an infinite sequence $\rho = (q_0,\nu_0)(q_1,\nu_1)(q_2,\nu_2)\ldots$ such that $(q_0,\nu_0) \in S_{\text{init}}$ and $\big((q_i,\nu_i),(\sigma_i,d_i),(q_{i+1},\nu_{i+1})\big) \in T$ for all $i \geq 0$. A run $\rho = (q_0,\nu_0)(q_1,\nu_1)\ldots$ is accepting if $q_i \in F$ for infinitely many indices i. Language of all words accepted by BADC $\mathcal{B}$ is

$$\mathcal{L}(\mathcal{B}) = \{w \in (\Sigma \times (0,1])^\omega \mid \text{ there exists an accepting run for } w \text{ in } \mathcal{B}\}.$$

Similarly as a Büchi automaton, a BADC can also be viewed as an oriented graph. An *infinite path* in a BADC $\mathcal{B} = (Q, Q_{\text{init}}, \Sigma, \delta, X, F)$ starting at state q_0 is an infinite sequence of states $q_0q_1\ldots$, where $q_i \in Q$ and there exist $\tau_i = (q_i,\sigma,\gamma,R,q_{i+1}) \in T$ for all $i \geq 0$. A finite *path* from q_0 to q_n is a finite subsequence $q_kq_{k+1}\ldots q_{l-1}q_l$ of a run, where $0 \leq k \leq l \leq n$. Further, we define a *simple path* and a *cycle* as a finite path $q_0q_1\ldots q_{n-1}q_n$ with the property that $q_i \neq q_j$, $\forall\, 0 \leq i,j \leq n-1, i \neq j$, and with the property that $q_0 = q_n$ and $q_0q_1\ldots q_{n-1}$ is a simple path, respectively.

Example 4.4.15 (Broadcasting Network)**.** *For our broadcasting network example (see Example 4.1.1) a BADC capturing the "redundant* a*-point" property is depicted in Figure 4.5.*

Example 4.4.16 (Data Gathering Robot I)**.** *For our robot data gathering system from Example 4.1.2, Figure 4.6 illustrates a BADC that captures the following desired property: "Data are periodically gathered in state* g_1 *and uploaded in an upload state before their integrity drops below 90%."*

Satisfying Trace Synthesis with BADC Specification

Before we focus on satisfying trace synthesis with the specification given in the form of a DLTL formula, we solve the following problem.

Problem Statement 4.4.17 (Satisfying Trace Synthesis with BADC Spec.)**.** *Given a TSD model* $\mathcal{T}$ *of a system with degradation and a specification in the form of a BADC* $\mathcal{B}$*, find a trace of* $\mathcal{T}$ *producing a word from* $\mathcal{L}(\mathcal{B})$*, if such a trace exist.*

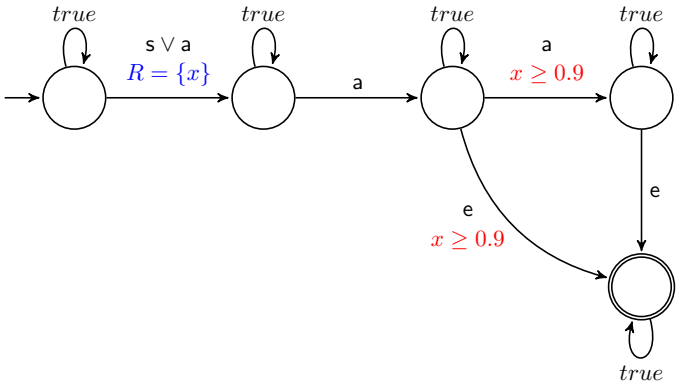


Figure 4.5: A BADC representing a property of the broadcasting system saying that there exists a redundant a-point in between the sender or another a and the end point. The BADC captures that if the signal was not be amplified in the redundant a, the signal would still reach the end point while keeping its quality above 90%. For simplicity of presentation, the edge labels are boolean combinations of atomic propositions, denoting all subsets of atomic propositions that comply with the label. For instance, the edge labeled with *true* in fact represents $2^{|AP|}$ transitions labeled with all possible atomic propositions subsets and the edge labeled with a represents $2^{|AP|-1}$ transitions labeled with all subset of atomic propositions that contain a.

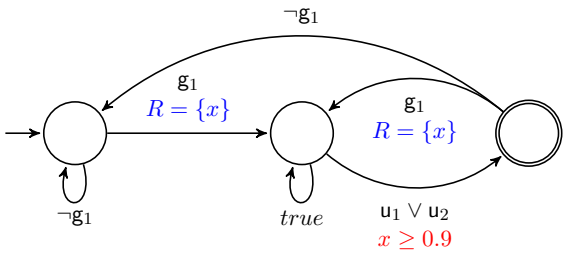


Figure 4.6: A BADC for a robot data gathering system representing that data are periodically gathered in g_1 and uploaded in an upload location before their integrity drops below 90%.

Our solution to Problem 4.4.17 follows the automata-based approach to LTL model checking [VW83]. In particular, we first define a product automaton and prove one-to-one correspondence between the accepting runs of this product automaton and the traces of TSD producing words that belong to the language of the BADC. Next, we demonstrate that searching for an accepting run of the product automaton is equivalent to finding a reachable accepting cycle in

the product automaton graph and can be performed efficiently by a number of known techniques like the nested DFS [CVWY92a] or OWCTY [FFK+01].

Definition 4.4.18 (Product of a TSD and a BADC)**.** *Product automaton of a TSD* $\mathcal{T} = (S, S_{\text{init}}, Act, T, AP, L, D)$ *and a BADC* $\mathcal{B} = (Q, Q_{\text{init}}, \Sigma, X, \delta, F)$ *is an automaton* $\mathcal{P} = \mathcal{M} \otimes \mathcal{B} = (Q_{\mathcal{P}}, Q_{\mathcal{P}}^{\text{init}}, Act, \delta_{\mathcal{P}}, F_{\mathcal{P}})$*, where*

- $Q_{\mathcal{P}} = S \times Q \times Eval(X)$*;*
- $Q_{\mathcal{P}}^{\text{init}} = \{(s, q, \nu) \mid s \in S_{\text{init}},\ q \in Q_{\text{init}},$ *and* $\nu(x) = 1$ *for all* $x \in X\}$*;*
- $\delta_{\mathcal{P}} : Q_{\mathcal{P}} \times Act \to 2^{Q_{\mathcal{P}}}$*, where*
  $(s', q', \nu') \in \delta_{\mathcal{P}}((s, q, \nu), \alpha)$ *if and only if*
  - $\exists\, (s, \alpha, s') \in\ T$
  - $\exists\, (q, L(s), \gamma, R, q') \in \delta$*, such that* $\nu \models \gamma$ *and* $\forall\ x \in X$:

$$\nu'(x) = \begin{cases} d & \text{if } x \in R \\ \nu(x) \cdot d & \text{otherwise;} \end{cases}$$

- $F_{\mathcal{P}} = \{(s, q, \nu) \mid q \in F\}$.

Note, that the product automaton $\mathcal{P}$ is in fact a Büchi automaton and as such it can be checked for language non-emptiness. The main obstacle in the verification process is that the product automaton graph may be infinite, as the number of states in $Q_{\mathcal{P}}$ might be infinite. An example of such a situation is depicted in Figure 4.7.

The key observation allowing for model checking of BADC properties of systems with degradation is that for a special type of BADC automata, the so-called *normalized* BADC, it is guaranteed that the product graph is finite. In the following, we give the definition of a normalized BADC and prove that the product automaton of a TSD and a normalized BADC is indeed finite. Further, we provide an algorithm which transforms any BADC to an equivalent normalized BADC.

Definition 4.4.19 (BADC Normal Form)**.**
Let us consider a BADC $\mathcal{B} = (Q, Q_{\text{init}}, \Sigma, X, \delta, F)$*. A degradation variable* $x \in X$ *is in normal form (or normalized, for short) in* $\mathcal{B}$ *if for each cycle* $q_0 q_1 \ldots q_{n-1} q_n$*, where* $q_0 = q_n$ *and for each sequence of transitions* $\tau_0, \ldots \tau_{n-1}$*, such that* $\forall 0 \leq i \leq n-1.\, \tau_i = (q_i, \sigma_i, \gamma_i, R_i, q_{i+1}) \in \delta$*, for some* $\sigma_i \in \Sigma, \gamma_i \in DC(X), R_i \subseteq X$*, the following statement holds true. There exist a transition* $\tau_j, 0 \leq j \leq n-1$ *with at least one of the following two properties:*

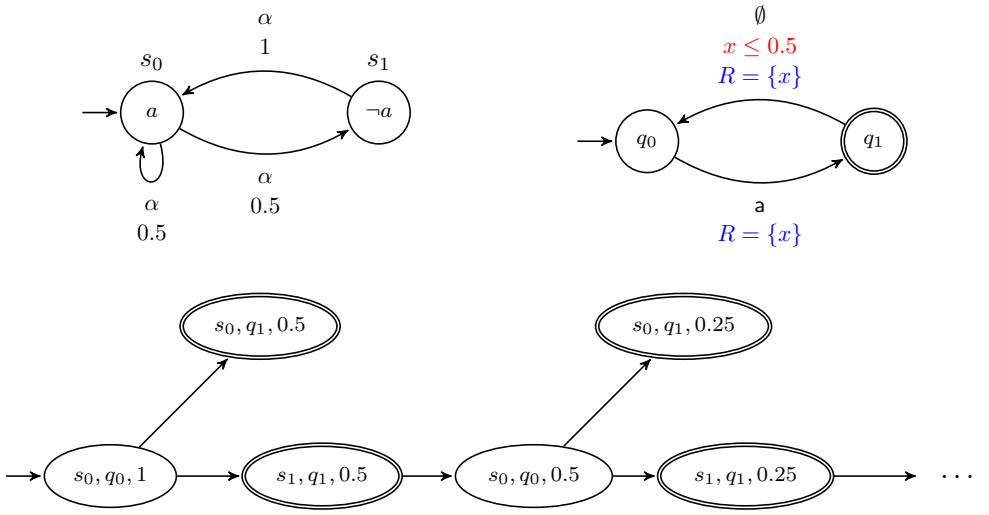


Figure 4.7: Example of an infinite product automaton (bottom) of a TSD (top left) and a BADC (top right). Here, the infinity is caused by decreasing value of the variable x always meeting the constraint $x \leq 0.5$.

- *constraint*$(\tau_i) = x \bowtie d$ *or* $x \bowtie d \wedge \gamma$, *where* $d \in (0, 1]$, $\bowtie \in \{\geq, >\}$, *and* $\gamma \in DC(X)$
- $x \in$ *reset*(τ_i)

$\mathcal{B}$ is in normal form (or normalized, for short) if each degradation variable $x \in X$ is in normal form in $\mathcal{B}$.

Informally speaking, in a normalized BADC, each sequence of states and transitions $q_0(q_0, \sigma_0, \gamma_0, R_0, q_1)q_1 \dots q_n(q_i, \sigma_i, \gamma_i, R_i, q_0)q_0$, each degradation variable of the BADC is reset or upper-bounded in at least one of the transitions of this sequence.

Lemma 4.4.20. *The product $\mathcal{P} = \mathcal{T} \otimes \mathcal{B}$ of a TSD $\mathcal{T} = (S, S_{\text{init}}, Act, T, AP, L, D)$ and a normalized BADC $\mathcal{B} = (Q, Q_{\text{init}}, \Sigma, X, \delta, F)$ is finite.*

Proof. To prove the finiteness of the product automaton $\mathcal{P}$ we have to demonstrate the finiteness of its set of states $Q_{\mathcal{P}} \subseteq S \times Q \times Eval(X)$. As S and Q are both finite (from the definition of TSD and BADC) it is enough to prove that every constraint variable $x \in X$ attains only finitely many different values in $\mathcal{P}$.

Let $\rho_{\mathcal{P}}^{\text{fin}} = (s_0, q_0, \nu_0)(s_1, q_1, \nu_1) \dots (s_k, q_k, \nu_k)$ be a finite run of $\mathcal{P}$ such that the degradation variable x is reset only in states (s_0, q_0, ν_0) and (s_k, q_k, ν_k). Formally, every transition from (s_i, q_i, ν_i) to $(s_{i+1}, q_{i+1}, \nu_{i+1})$ can be projected

to the corresponding transition $t_i = (s_i, \alpha_i, s_{i+1})$ of $\mathcal{T}$ and the transition $\tau_i = (q_i, L(s_i), \gamma_i, R_i, q_{i+1})$ of $\mathcal{B}$. The variable x is reset in a state (s_i, q_i, ν_i) of $\mathcal{P}$ iff $x \in R_i$. The initial value of x on $\rho_{\mathcal{P}}^{\text{fin}}$ is d_0 and along the run is changed to $d_1 = \prod_{i=0}^{1} d_i$, $d_2 = \prod_{i=0}^{2} d_i$, ..., $d_k = \prod_{i=0}^{k-1} d_i$. This sequence of x-values is non-increasing (with the possible exception of the last value d_k). We now prove that there is a bound $b \in \mathbb{N}$ (depending only on $\mathcal{T}$ and $\mathcal{B}$) such that the value of x can be decreased on any $\rho_{\mathcal{P}}^{\text{fin}}$ at most b times. The existence of the bound b, together with the fact that there are only finitely many different degradation constants d in transitions of $\mathcal{T}$, assure that x attains only a finite number of different values along a path in $\mathcal{P}$.

We define constants d_{min}, n_{min} and l_{max} distinguishing extremal values in $\mathcal{T}$ and $\mathcal{B}$. For the BADC $\mathcal{B}$ we define d_{min} as the minimal value, such that there is a transition τ with $constraint(\tau) = x \bowtie d_{min}$ or $x \bowtie d_{min} \wedge \gamma$, where $\bowtie \in \{\geq, >\}, d_{min} \in (0, 1]$ and $\gamma \in DC(X)$. For the TSD $\mathcal{T}$ we define n_{min} as the minimal number such that the product of any n_{min} degradation constants d from transitions of $\mathcal{T}$ is less than d_{min}. The constant l_{max} is the length (the number of transitions) of the longest elementary cycle in $\mathcal{B}$. Let us suppose the value of x is decreased on $\rho_{\mathcal{P}}^{\text{fin}}$ more than $n_{min} + l_{max}$ times. After the first n_{min} decreases the value of x is less than d_{min}. The length of the path $\rho_{\mathcal{P}}^{\text{suf}}$, which is a suffix of $\rho_{\mathcal{P}}^{\text{fin}}$, starting in the state where the value of x decreased below d_{min} for the first time, is greater than l_{max}. The variable x is normalized in $\mathcal{B}$ and thus there is a transition $\tau_i = \big((s_i, q_i, \nu_i), \sigma_i, \gamma_i, R_i, (s_{i+1}, q_{i+1}, \nu_{i+1})\big)$ on $\rho_{\mathcal{P}}^{\text{suf}}$ such that $\gamma_i = x \bowtie d$ or $x \bowtie d \wedge \gamma$, where $\bowtie \in \{\geq, >\}, d \in (0, 1]$ and $\gamma \in DC(X)$. However, this constraint cannot be satisfied as the value $\nu_i(x) < d_{min} \leq d$. This contradicts the assumption about $\rho_{\mathcal{P}}^{\text{fin}}$. □

Let $\mathcal{B}$ be a normalized BADC, $\mathcal{T}$ a TSD, and $\mathcal{P}$ their finite product. The following lemma gives us guidance how to solve the satisfying trace synthesis problem (Problem 4.4.17).

Lemma 4.4.21. *There exist a trace r of $\mathcal{T}$ producing a word $w(r)$ that is accepted by $\mathcal{B}$ if and only if there exist an accepting run of $\mathcal{P}$.*

Proof. First, let $\mathcal{P}$ contain an accepting cycle. Then there is an infinite run $\rho_{\mathcal{P}} = (s_0, q_0, \nu_0)(s_1, q_1, \nu_1)\ldots$ of $\mathcal{P}$ with infinitely many accepting states, such that $(s_{i+1}, q_{i+1}, \nu_{i+1}) \in \delta_P\big((s_i, q_i, \nu_i), \alpha_i\big)$, for all $0 \leq i$. The projection of $\rho_{\mathcal{P}}$ onto the corresponding TSD $\mathcal{T}$ gives the trace $r = s_0\alpha_0 s_1\alpha_1 s_2\ldots$ producing the word $w(r) = (L(s_0), d_0))\ (L(s_1), d_1)), \ldots$. The projection of $\rho_{\mathcal{P}}$ onto the states of $\mathcal{B}$ $\rho = (q_0, \nu_0)(q_1, \nu_1)(q_2, \nu_2)\ldots$ forms an accepting run of $\mathcal{B}$ over the word $w(r)$. Thus $\mathcal{L}(\mathcal{B}) \cap \mathcal{L}(\mathcal{T}) \neq \emptyset$.

On contrary, let $w = (\sigma_0, d_0)(\sigma_1, d_1)\ldots \in \mathcal{L}(\mathcal{B}) \cap \mathcal{L}(\mathcal{T})$. Then there is a trace $r = s_0\alpha_0 s_1\alpha_1 s_2\ldots$ in $\mathcal{T}$ with $w(r) = (L(s_0), D(t_0))\ (L(s_1), D(t_1))\ldots$ where $t_i = (s_i, \alpha_i, s_{i+1}) \in T$ for all $i \geq 0$. Let $\rho = (q_0, \nu_0)(q_1, \nu_1)(q_2, \nu_2)\ldots$ be an accepting run over $w(r)$ of $\mathcal{B}$. Then $q_0 \in Q_{\text{init}}, \nu_0(x) = 1$, for all $x \in X$, and $\big((q_i, \nu_i), (L(s_i), D(t_i)), (q_{i+1}, \nu_{i+1})\big) \in T_{\mathcal{S}}$ for all $i \geq 0$, such that the semantics of $\mathcal{B}$ is given by the transition system $\mathcal{S} = (S_{\mathcal{S}}, S^{\text{init}}_{\mathcal{S}}, Act_{\mathcal{S}}, T_{\mathcal{S}})$. Furthermore, there are infinitely many indices i with $q_i \in F$. By synchronization of the trace $r = s_0\alpha_0 s_1\alpha_1 s_2\alpha_2\ldots$ and the run $\rho = (q_0, \nu_0)(q_1, \nu_1)(q_2, \nu_2)\ldots$ we obtain a run $\rho_{\mathcal{P}} = (s_0, q_0, \nu_0)(s_1, q_1, \nu_1)(s_2, q_2, \nu_2)\ldots$ in the product $\mathcal{P}$ with infinitely many indices i, such that $(s_i, q_i, \nu_i) \in F_{\mathcal{P}}$, i.e. $\mathcal{P}$ contains an accepting cycle. □

Normalization

The remaining open question is how to normalize a BADC, formally stated as follows.

Problem Statement 4.4.22 (BADC Normalization)**.** *Given a BADC $\mathcal{B}$, find a normalized BADC $\mathcal{B}'$, such that $\mathcal{L}(\mathcal{B}) = \mathcal{L}(\mathcal{B}')$.*

To solve Problem 4.4.3, we first introduce some necessary definitions and then the normalization algorithm itself.

Definition 4.4.23 (Bounded Degradation Variable)**.** *Let us say that a degradation variable x is bounded in the degradation constraint $\gamma \in DC(X)$ if*

- $\gamma = x \bowtie d$, *or*
- $\gamma = x \bowtie d \wedge \gamma'$, $\gamma \in DC(X)$.

More precisely, x is n-bounded in constraint $\gamma \in DC(X)$ if

- $\gamma = x \bowtie_1 d_1 \wedge \ldots \wedge x \bowtie_n d_n$, *or*
- $\gamma = x \bowtie_1 d_1 \wedge \ldots \wedge x \bowtie_n d_n \wedge \gamma'$, *where x is not bounded in γ'.*

Variable x is (n-)bounded in transition τ if x is (n-)bounded in constraint(τ).

Assume that there is only one degradation constraint $x \bowtie d_x$ which bounds x in a BADC $\mathcal{B}$. We define a lower bound

$$lb(x) = \begin{cases} x \bowtie d_x & \text{if } \bowtie \in \{<, \leq\} \\ \neg(x \bowtie d_x) & \text{otherwise} \end{cases}$$

The transformation algorithm (see Algorithm 1) works in several stages. In the initial stage (see one_constraint_on_variable in Algorithm 2), the given BADC is transformed into a BADC in which every degradation variable x is

Algorithm 1 normalize($\mathcal{B}$)

Input: BADC $\mathcal{B}^{in} = (Q^{in}, Q^{in}_{\text{init}}, \Sigma, X^{in}, \delta^{in}, F^{in})$
Output: BADC $\mathcal{B} = (Q, Q_{\text{init}}, \Sigma, X, \delta, F)$ in normal form

$\mathcal{B} := \mathcal{B}^{in}$
one_constraint_on_variable
$Done := \emptyset$
while $Done \neq X$ **do**
  pick $x \in X \setminus Done$
  $(Q_s, Q_o, Q_x) :=$ reset_where_possible(x)
  split_into_layers(x)
  $Done := Done \cup \{x\}$
  Assert: Each $x \in Done$ is normalized in $\mathcal{B}$
end while
Assert: $\mathcal{B}$ is in normal form
Assert: $\mathcal{L}(\mathcal{B}^{in}) = \mathcal{L}(\mathcal{B})$

Algorithm 2 one_constraint_on_variable

for all $x \in X^{in}$ **do**
  $\mathcal{B}' := \mathcal{B}$
  $\delta_x := \{\tau_i \in \delta \mid x \text{ is bounded in } \tau_i\}$
  for all $\tau_i \in \delta_x$ **do**
    $m_i := n$ such that x is n-bounded in τ_i
    $X := (X \setminus \{x\}) \cup \{x_{i1} \ldots, x_{im_i} \mid x_{i1}, \ldots, x_{im_i}$ are new, unique degradation variables$\}$
    replace $constraint(\tau_i) = x \bowtie_1 d_1 \wedge \ldots \wedge x \bowtie_{m_i} d_{m_i} \wedge \gamma$
    with $x_{i1} \bowtie_1 d_1 \wedge \ldots \wedge x_{im_i} \bowtie_{m_i} d_{m_i} \wedge \gamma$
  end for
  for all $\tau \in \delta$ **do**
    if $x \in reset(\tau)$ **then**
      $reset(\tau) := (reset(\tau) \setminus \{x\}) \cup \{x_{i1}, \ldots, x_{im_i} \mid \tau_i \in \delta_x\}$
    end if
  end for
  Assert: $\mathcal{L}(\mathcal{B}') = \mathcal{L}(\mathcal{B})$
end for
Assert: $\forall x \in X : \exists! \tau \in \delta$ with bounded x
Assert: $\forall x \in X : \exists! \tau \in \delta$ with 1-bounded x
Assert: $\mathcal{L}(\mathcal{B}^{in}) = \mathcal{L}(\mathcal{B})$

bounded by at most one inequality $x \bowtie d_x$. This is accomplished by introducing new degradation variables into the BADC.

In the next stage, we iteratively pick a degradation variable $x \in X$ and transform the BADC so that x becomes normalized while preserving the normal form of the already processed degradation variables. Normalization of the degradation variable x involves two procedures. The first procedure reset_where_possible (see Algorithm 3) identifies those transitions where the variable x can be safely reset.

Algorithm 3 reset_where_possible(x)

$\delta_x := \{\tau_x \in \delta \mid x \text{ is bounded in } \tau_x\}$
$Q_x := \{q_x \in Q \mid \exists \text{ transition } \tau_x \in \delta_x \text{ from state } q_x\}$
$Paths := \{q_0 q_1 \dots q_n q_x \mid q_0 q_1 \dots q_n q_x$ is a simple or a lasso path in $\mathcal{B}$, $q_x \in Q_x$, and
  $q_0 = q_{\text{init}}$ or $\exists$ transition $\tau \in \delta$ leading to q_0 such that $x \in reset(\tau)$, and
  $\forall\, 0 \le i \le n.\, x \notin reset((q_i, \sigma_i, \gamma_i, R_i, q_{i+1}))$, for any $(q_i, \sigma_i, \gamma_i, R_i, q_{i+1}) \in \delta$ $\}$
$Q_s := \{q_0 \mid \exists\ \rho_{\text{path}} \in Paths$ starting at $q_0\}$
$Q_o := \{q_i, q_x \mid \exists\ \rho_{\text{path}} = q_0 \dots q_n q_x \in Paths,$ where $0 \le i \le n\}$
$\delta_o := \{\tau_i \mid \exists\ \rho = q_0 \dots q_n q_x \in Paths,$ where $\tau_i = (q_i, \sigma_i, \gamma_i, R_i, q_{i+1})$ for some $0 \le i \le n\}$
$\delta_n := \delta \setminus \delta_o$
for all $\tau \in \delta_n$ **do**
  $reset(\tau) := reset(\tau) \cup \{x\}$
end for
return (Q_s, Q_o, Q_x)
Assert: $\mathcal{L}(\mathcal{B}^{in}) = \mathcal{L}(\mathcal{B})$

Algorithm 4 split_into_layers(x)

Let $x \bowtie d$ be the constraint on x
$lb(x) := \bowtie \in \{<, \le\}\ ?\ x \bowtie d : \neg(x \bowtie d)$
$Q := Q \cup \{\widehat{q}, \breve{q} \mid q \in Q_o, \widehat{q}, \breve{q}$ are new, unique states$\} \cup \{\ddot{q} \mid q \in Q_s, \ddot{q}$ is a new, unique state$\}$
$Q_{init} := \{q_{\text{init}} \mid q_{\text{init}} \in Q_{\text{init}} \setminus Q_s\} \cup \{\ddot{q}_{init} \mid q_{init} \in Q_{\text{init}} \cap Q_s\}$
$F := (F \setminus Q_o) \cup \{\widehat{q}, \breve{q} \mid q \in Q_o \cap F\} \cup \{\ddot{q} \mid q \in Q_s \cap F\}$
for all $\tau = (q_1, \alpha, \gamma, R, q_2) \notin \delta_x$ **do**
  case $q_1 \notin Q_o$, $x \in R$, and $q_2 \in Q_s$: replace τ with $(q_1, \alpha, \gamma, R, \ddot{q}_2)$
  case $q_1 \in Q_o$, $x \in R$, and $q_2 \in Q_s$: replace τ with $(\widehat{q}_1, \alpha, \gamma, R, \ddot{q}_2)$, and $(\breve{q}_1, \alpha, \gamma, R, \ddot{q}_2)$
  case $q_1 \in Q_s$, $x \in R$, and $q_2 \in Q_s$: add transition $(\ddot{q}_1, \alpha, \gamma, R, \ddot{q}_2)$
  case $q_1 \in Q_o$, $x \notin R$, and $q_2 \in Q_o$: replace τ with $(\widehat{q}_1, \alpha, \gamma \wedge \neg lb(x), R, \widehat{q}_2)$, $(\widehat{q}_1, \alpha, \gamma \wedge lb(x), R \cup \{x\}, \breve{q}_2)$, and $(\breve{q}_1, \alpha, \gamma, R \cup \{x\}, \breve{q}_2)$
  case $q_1 \in Q_s$, $x \notin R$, and $q_2 \in Q_o$: add transitions $(\ddot{q}_1, \alpha, \gamma \wedge \neg lb(x), R, \widehat{q}_2)$, and $(\ddot{q}_1, a, \gamma \wedge lb(x), R \cup \{x\}, \breve{q}_2)$
  case $q_1 \in Q_o$, and $q_2 \notin Q_o$: replace τ with $(\widehat{q}_1, \alpha, \gamma \wedge, R \cup \{x\}, q_2)$, and $(\breve{q}_1, \alpha, \gamma, R \cup \{x\}, q_2)$
  case $q_1 \in Q_s$, and $q_2 \notin Q_o$: add $(\ddot{q}_1, \alpha, \gamma, R \cup \{x\}, q_2)$
end for
for all $\tau_x = (q_x, \alpha, \gamma, R, q_2) \in \delta_x$ **do**
  if $\bowtie \in \{<, \le\}$ **then**
    $\bar{q}_2 := q_2 \notin Q_o\ ?\ q_2\ :\ (x \in R\ ?\ \ddot{q}_2\ :\ \breve{q}_2)$
    replace τ_x with $(\widehat{q}_x, \alpha, \gamma \wedge lb(x), R \cup \{x\}, \bar{q}_2)$, $(\breve{q_x}, \alpha, \gamma, R \cup \{x\}, \bar{q}_2)$, and if $q_x \in Q_s$ add $(\ddot{q}_x, \alpha, \gamma \wedge lb(x), R \cup \{x\}, \bar{q})$
  else
    $\bar{q}_2 := q_2 \notin Q_o\ ?\ q_2\ :\ (x \in R\ ?\ \ddot{q}_2\ :\ \widehat{q}_2)$
    replace τ_x with $(\widehat{q}_x, \alpha, \gamma \wedge \neg lb(x), R, \bar{q}_2)$, and if $q_x \in Q_s$ add $(\ddot{q}_x, \alpha, \gamma \wedge \neg lb(x), R, \bar{q})$
  end if
end for
Assert: $\mathcal{L}(\mathcal{B}^0) = \mathcal{L}(\mathcal{B})$

To this end it computes the set *Paths* of all simple and lasso paths ρ_{path} satisfying three conditions: ρ_{path} starts in an initial state or in a state immediately after reset of x, no reset of x occurs along ρ_{path}, and ρ_{path} ends in a state from which there is a transition with a bound on x. Now we can split all the transitions into two disjoint sets: those which occur on a path from ρ_{path} (the set δ_o) and those which do not (the set δ_n). We can reset the variable x on the transitions from δ_n without changing the language of the BADC. Simultaneously, three other sets of states, namely Q_s, Q_o, and Q_x, are computed. Q_s is the set of states in which a path $\rho_{\text{path}} \in Paths$ starts. Q_o are states occurring along a path $\rho_{\text{fin}} \in Paths$, and finally Q_x are states in which a path $\rho_{\text{path}} \in Paths$ ends. Note that $Q_s, Q_x \subseteq Q_o$. These sets are used in the following steps of the algorithm.

Procedure split_into_layers (Algorithm 4) finishes the normalization of the variable x. It manipulates the rest of the transitions that may cause that x is not normalized, namely those from the set δ_o. The modification of the BADC is a bit more involved here and requires a replication of states. Each replica of the state bears a specific information about the actual value of x. Namely, we replace each state $q \in Q_o$ with two new states $\breve{q}$ and $\widehat{q}$. Moreover, if $q \in Q_s$, we introduce a new state $\ddot{q}$. The information associated with the replicas is intuitively characterized as follows:

- $\ddot{q}$-states: Whenever the state $q \in Q_s$ is entered via a transition with reset of x from state p in the original BADC, the state $\ddot{q}$ is entered in the transformed one from state p if $p \notin Q_o$ or from any replica of p if $p \in Q_o$. The value $\nu(x)$ remains the same in $\ddot{q}$ and in the corresponding q in the original BADC.
- $\breve{q}$-states: Let $x \bowtie d_x$ be the only degradation constraint which bounds x in $\mathcal{B}$. Whenever the state $q \in Q_o$ is entered from a state p in which $\nu(x) \models lb(x)$ via a transition without reset of x in the original BADC, the state $\breve{q}$ is entered in the transformed one from any replica of p (necessarily, $p \in Q_o$). Due to the monotonicity of degradation, starting from the state p the value of x remains less or less-or-equal than d_x until a reset of x. Therefore, we do not need to keep the value $\nu(x)$ in $\breve{q}$ the same as in q (it suffices to know that $\nu(x)$ remains below d_x). Thus we can add reset of x on each transition entering the $\breve{q}$-state, including the state $\breve{q}_x$.
- $\widehat{q}$-states: $\widehat{q}$-states are dual to $\breve{q}$-states. Whenever the state $q \in Q_o$ is entered from a state p in which $\nu(x) \not\models lb(x)$ via a transition without reset of x in the original BADC, the state $\widehat{q}$ is entered in the transformed one from $\widehat{p}$ and in case $p \in Q_s$ also from $\ddot{p}$. It cannot be entered from $\breve{p}$ as we know that $\nu(x) \models lb(x)$ in $\breve{p}$. The value $\nu(x)$ is the same in $\widehat{q}$-state

and in the corresponding q-state in the original BADC. Note that any transition leading to a $\widehat{q}$-state contains a bound of form $x > d_x$ or $x \geq d_x$.

Transitions entering q are naturally replaced by transitions entering $\ddot{q}, \breve{q}$ or $\widehat{q}$ keeping the above characteristics. Normal form is guaranteed by the fact that for every degradation variable, every transition in the resulting BADC either resets the value of x or contain a constraint of the form $x > d_x$ or $x \geq d_x$.

Lemma 4.4.24. *Algorithm 1 is sound and complete and gives an answer to the BADC normalization problem (Problem 4.4.3).*

Proof. The proof follows from the individual assertions placed in the Algorithms. The full proof can be found in Appendix A. □

The time complexity of the procedure one_constraint_on_variable (Algorithm 1) is $\mathcal{O}(|\delta| \cdot n)$, where $|\delta|$ is the number of transitions, and n denotes the overall number of occurrences of all degradation variables in the input BADC (i.e. $\Sigma_{x \in X}\ \Sigma_{\tau \in \delta}\ m$, where X is the set of degradation variables, and x is m-bounded in τ). The number of iterations of the cycle 4-9 of Algorithm 1 is n. Both procedures reset_where_possible(x) and split_into_layers(x) take $\mathcal{O}(|Q'| + |\delta'|)$ time, where $|Q'|$ is the number of states and $|\delta'|$ is the number of transitions in the modified BADC just before the procedures are performed. During the procedures, each state is replaced with at most 4 new ones and transition is replaced with at most 4 new ones. Altogether the worst-case complexity of the transformation is

$$\mathcal{O}(|\delta| \cdot n) + \mathcal{O}(\sum_{i=0}^{n} 4^i \cdot (|Q| + |\delta|)) = \mathcal{O}(2^{2n} \cdot (|Q| + |\delta|)),$$

where $|Q|$ is the number of states in the input BADC, and $|\delta|$, and n are defined as above.

Satisfying Trace Synthesis with BADC Specification Solution Overview

To sum up, the suggested solution to Problem 4.4.17 is as follows. The input of the algorithm is a TSD $\mathcal{T}$ and a BADC $\mathcal{B}$.

(i) Apply Algorithm 1 to the BADC $\mathcal{B}$ and obtain a normalized BADC $\mathcal{B}'$.

(ii) Create a product Büchi automaton $\mathcal{P} = \mathcal{T} \otimes \mathcal{B}'$ according to Def. 4.4.18.

(iii) Find an accepting run $\rho_{\mathcal{P}}$ in the product $\mathcal{P}$ and project $\rho_{\mathcal{P}}$ onto $\mathcal{T}$ to obtain a trace that is solution to Problem 4.4.17.

For an illustrative example of the transformation process see Figures 4.8, 4.9, and 4.10. In each step, we present the automaton after the removal of the states that are not reachable from the initial state.

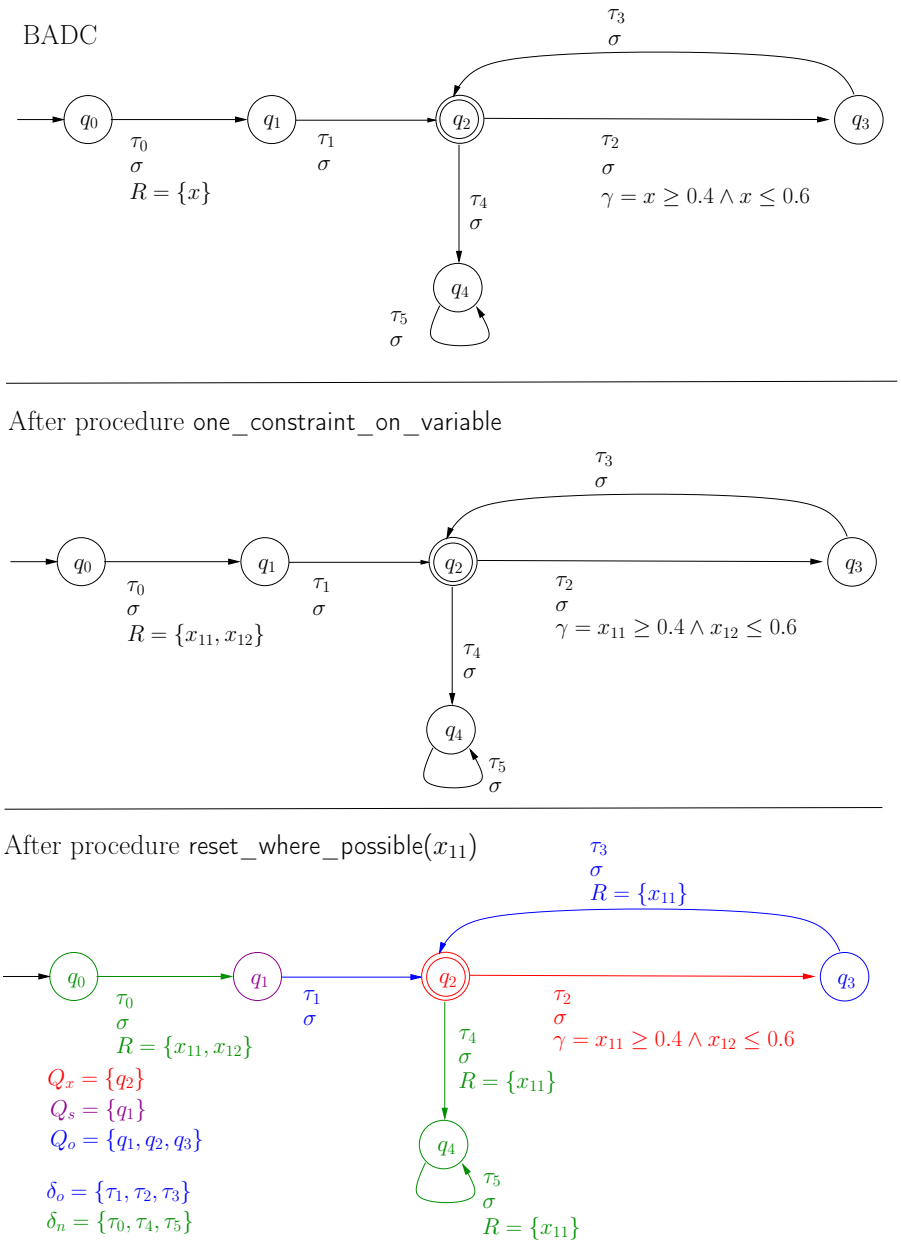
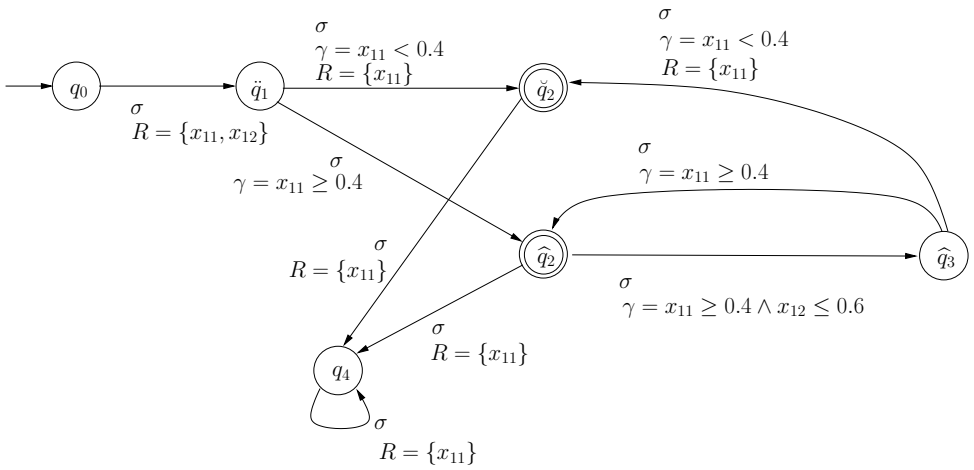


Figure 4.8: Example of Normalization I.

After procedure split_into_layers(x_{11})



After procedure reset_where_possible(x_{12})

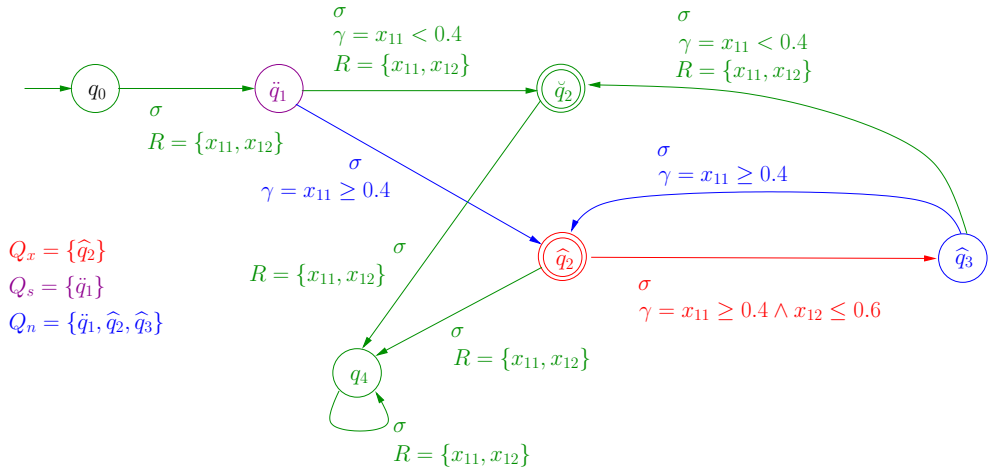


Figure 4.9: Example of Normalization II.

After procedure split_into_layers(x_{12})

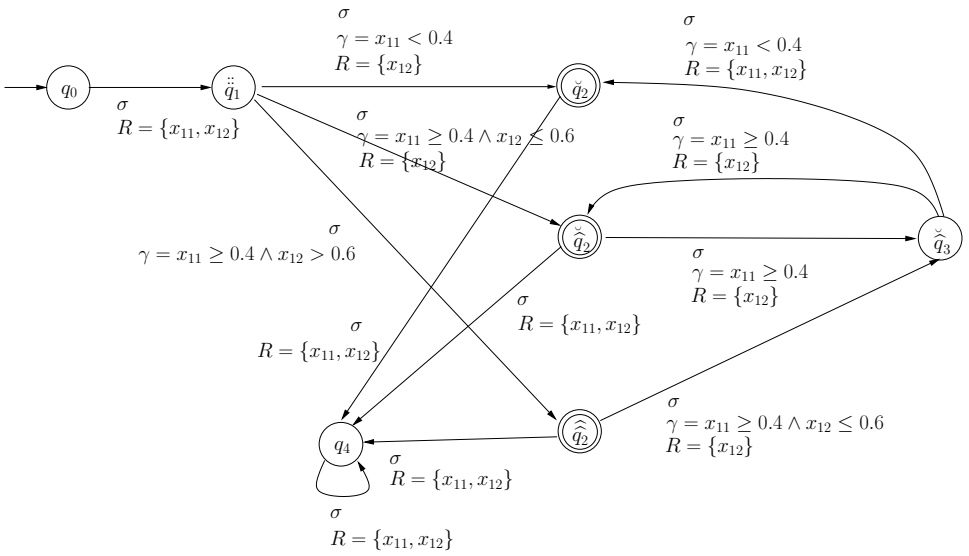


Figure 4.10: Example of Normalization III.

Correctness and Complexity

Theorem 4.4.25. *The suggested algorithm terminates and gives an answer to the satisfying trace synthesis problem as defined in Problem 4.4.17.*

Proof. Follows from Lemmas 4.4.20, 4.4.21, and 4.4.24. □

The complexity of the overall solution is given by the worst-case complexity of the normalization process and the fact, that the language non-emptiness check for Büchi automata is linear with respect to the size of the BA. Altogether, we have the time complexity in $\mathcal{O}(2^{2^n} \cdot |\mathcal{T}| \cdot |\mathcal{B}|)$, where $|\mathcal{T}|$ is the size of the input TSD, $|\mathcal{B}|$ is the size of the input BADC and n is the overall number of occurences of all degradation variables in $\mathcal{B}$.

Remark 4.4.26. *There is a way to construct the product automaton without normalization via modification of the procedure of the product automaton construction as follows. As soon as the value of a degradation variable drops below the minimal threshold occurring in the BADC, the value is tagged with a special flag denoting "below minimal threshold" and it is left with its current value and*

not manipulated in succeeding states anymore. This approach leads to a finite product automaton with $\mathcal{O}(|S| \cdot |Q| \cdot (\log_{step} min)^{|X|})$ *states, where* $|S|$ *is the number of states of a TSD,* $|Q|$ *is the number of states in an BADC before normalization, step is the maximal degradation constant different from 1 occurring in the TSD,* $|X|$ *is the number of degradation variables, and min is the overall minimal threshold occurring in the BADC.*

On the other hand, if the normalization is used, the number of states of the product is $\mathcal{O}(|S| \cdot |Q| \cdot \prod_{d \in D_N} \log_{step} min(d))$*, where* $|S|$*,* $|Q|$*, and step are as above,* D_N *is the set of degradation variables after normalization, and* $min(d)$ *is the minimal threshold connected with degradation variable d occurring in the BADC.*

The reason, why we have introduced normalization is that it helps to rapidly reduce the size of the product automaton in many cases. It is basically a heuristic to minimize the number of different values each degradation variable may get. For an example, see Figure 4.11 which illustrates an original BADC, its normalized form and a transition system. The product of the original BADC and the TSD has 239 states, whereas in the case of the normalized BADC the product has only 46 states. The normalization procedure adds resets of variables whenever it is possible. See, e.g., the self-loop on state q_1*.*

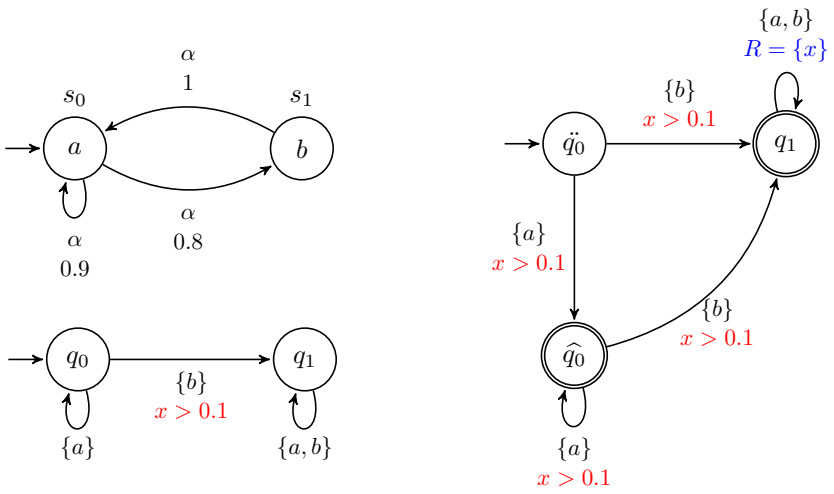


Figure 4.11: A TSD (top left), a BADC (bottom left), and its normalized version (right).

4.4.4 Problem Solution: Half-Bounded DLTL Specification

The timed automata approach presented in Section 4.4.2 gives answer to the

satisfying trace synthesis question (Problem 4.4.1) only with limited precision. On the other hand, the approach presented in Section 4.4.3 answers the problem with full precision, however, is able to deal only with automata-like specification formalism, that is, arguably less user-friendly than a logic-like formalism. In this section, we focus on full-precision DLTL model checking problem via translation of DLTL formulas into BADCs. We focus on a a special subclass of DLTL formulas with half-bounded intervals and solve Problem 4.4.1 for this DLTL fragment.

Definition 4.4.27 (Half-Bounded DLTL Formula)**.** *A half-bounded DLTL formula is a DLTL formula given according to the grammar*

$$\phi ::= true \mid \pi \mid \neg\phi \mid \phi \wedge \phi \mid \mathsf{X}_I\, \phi \mid \phi\, \mathsf{U}_I\, \phi,$$

where $\pi \in AP$, and I is an interval of form $(0,d], (0,d), (d,1]$, or $[d,1]$, where $d \in (0,1]$. For simplicity, we use $\bowtie d$, where $\bowtie \in \{\leq, <, \geq, >\}$ to denote interval I. In particular, we use $\leq d, < d, \geq d$, and $> d$ to denote interval $(0,d], (0,d), (d,1]$, and $[d,1]$, respectively.

Problem Statement 4.4.28 (DLTL to BADC Translation)**.** *Given a half-bounded DLTL formula ϕ find a BADC $\mathcal{B}_\phi$, such that $\mathcal{L}(\phi) = \mathcal{L}(\mathcal{B}_\phi)$.*

The algorithm we present for translation of DLTL formulas into BADCs is based on the standard algorithm for translation of LTL formulas into Büchi automata as given in [GPVW96]. However, several nontrivial issues have to be overcome to deal with the degradation constraints.

The whole process of translation consists of several succeeding steps. These are summarized below.

- First, a linear temporal logic formula ϕ is rewritten into such a form that the formula contains only the standard and constrained temporal operators U (until), R (release), and X (next), and at the same time all the $\neg$ (negation) operators are applied to propositional variables only.
- Second, a formula graph is built with a tableau-like procedure. Each graph node is defined by several sets of formulas. The key idea of the procedure employs several equalities such as the following one. Formula $\phi\, \mathsf{U}\, \psi$ is equal to $\psi \vee (\phi \wedge \mathsf{X}(\phi\, \mathsf{U}\, \psi))$. This equality allows to define what are the requirements on the current state of the run and what are the requirements on the next state of the run. For $\phi\, \mathsf{U}\, \psi$ subformula there are two decomposition options: either ψ has to hold true in the current state and there are no requirements regarding the next state, or ϕ has to hold true in the current state and $\phi\, \mathsf{U}\, \psi$ has to hold in the next state.

- Third, using the formula graph, a generalized Büchi automaton (GBA) is built.
- Finally, the GBA is transformed into a standard Büchi automaton accepting exactly the words satisfying the formula ϕ.

Normal Form

DLTL formula ϕ is *normalized* if it contains only temporal operators U, X, R, $\mathsf{U}_{\bowtie d}$, $\mathsf{X}_{\bowtie d}$, and $\mathsf{R}_{\bowtie d}$ and all negation operators apply to atomic propositions. Any DLTL formula can be normalized using the standard syntax equivalences and equivalences defined earlier in Section 4.3. Thus, without loss of generality, we may assume that from now on the given DLTL formula ϕ is normalized.

Formula Graph

Let us first informally present key ideas used in the construction of the formula graph for a normalized DLTL formula ϕ. The full algorithm is presented in Algorithm 5.

The basic data structure of the formula graph is called a *node*. Each node contains three important set of formulas, which are subformulas of the original formula ϕ. Intuitively, these describe temporal properties of suffices of runs of TSDs. The three sets are:

- *Old* is a set of formulas, that must hold in the node and have been already processed.
- *New* is a set of formulas, that must hold in the node and have not yet been processed.
- *Next* is a set of formulas that must hold in all immediate successors of nodes satisfying properties in *Old*.

Initially, the construction starts with a node, whose *Old* and *Next* are empty and *New* contains the formula ϕ. The nodes are iteratively expanded. Every time, a node q and a formula $\psi \in New(q)$ are chosen to be processed. The processed node is replaced with a new node (or several nodes). The sets *Old*, *New* and *Next* are inherited from the processed node, but ψ is moved from *New* to *Old* and *New* and *Next* are updated depending on the formula ψ. When there are no other formulas in $New(q)$, a new node p with $New(p) = Next(q)$ is created together with an edge leading from q to p.

The key idea in the translation process of DLTL formulas is that constraints on the level of degradation given in the formula ϕ can be captured through degradation constraints and resets associated with the edges between nodes in

the formula graph. More precisely, we associate a degradation variable with every unary, or binary operator $\mathsf{O}_{\bowtie d}$ when a formula $\mathsf{O}_{\bowtie d}\,\psi$, or $\phi\,\mathsf{O}_{\bowtie d}\,\psi$, respectively, is moved from *New* to *Old*. Let us now explain in more details how each of formulas $\mathsf{X}_{\bowtie d}\,\psi$, $\phi\,\mathsf{U}_{\bowtie d}\,\psi$, and $\phi\ \mathsf{R}_{\bowtie d}\,\psi$ is processed.

Let q and $\mathsf{X}_{\bowtie d}\,\psi \in \mathit{New}(q)$ be the currently processed node and formula, respectively. For each trace r satisfying $\mathsf{X}_{\bowtie d}\,\psi$ it holds that the level of degradation between $r(0)$ and $r(1)$ has to satisfy $\bowtie d$. Therefore, we associate a degradation variable x with the node q and reset x on each edge incoming to the node q. All outgoing edges from the current node are labeled with constraint $x \bowtie d$. Finally, we replace q with a new node q', such that formula the $\mathsf{X}_{\bowtie d}\psi$ is moved from $\mathit{New}(q')$ to $\mathit{Old}(q')$ and ψ is added to $\mathit{Next}(q')$. For illustration, see Figure 4.12.

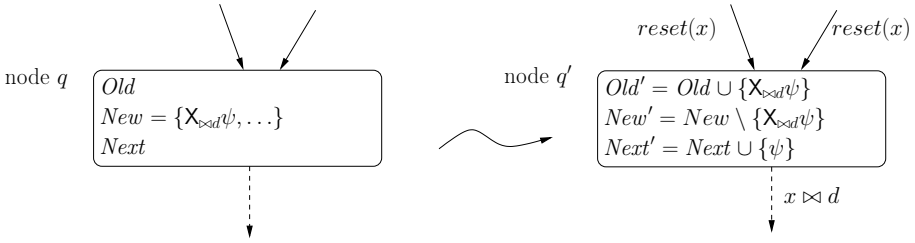


Figure 4.12: Expansion of a node containing a constrained next operator in its *New* set.

Let q and $\phi\,\mathsf{U}_{\bowtie d}\,\psi \in \mathit{New}(q)$ be the currently processed node and formula, respectively. Similarly as in the previous case, we need to measure degradation from the current node q and we do it with degradation variable x. We reset variable x on each edge incoming to the node q. There are two options: Either ψ is satisfied in the current node and the current degradation (*i.e.*, the value of x) satisfies $\bowtie d$ or ϕ is satisfied in the current node and in the next node it holds, that ϕ is satisfied until the moment when ψ is satisfied and the current degradation (*i.e.*,the value of x) in that moment satisfies $\bowtie d$. As we have to remember that degradation variable x is already initialized in the node q, we define an until operator with fixed degradation variable $\mathsf{U}^x_{\bowtie d}$:

$$\phi\,\mathsf{U}^x_{\bowtie d}\,\psi \equiv (x \bowtie d \wedge \psi) \vee \mathsf{X}(\phi\,\mathsf{U}^x_{\bowtie d}\,\psi).$$

The difference between $\mathsf{U}_{\bowtie d}$ and $\mathsf{U}^x_{\bowtie d}$ operators is the following: When $\phi\,\mathsf{U}_{\bowtie d}\,\psi$ is to be satisfied in the node q, we have to associate a variable x with this formula and reset it on all edges incoming to q. When $\phi\,\mathsf{U}^x_{\bowtie d}\,\psi$ is to be satisfied in the node q, it is already associated with a degradation variable x. Variable x cannot be reset, because it "measures" the degradation from some of the predecessors

of q. The operator $\mathsf{U}_{\bowtie d}$ can be expressed as:

$$\phi \,\mathsf{U}_{\bowtie d}\, \psi \equiv (x \bowtie d \wedge \psi) \vee \mathsf{X}(\phi \,\mathsf{U}^{x}_{\bowtie d}\, \psi),$$

where x is a degradation variable with value set to 1.

However, this approach would lead in some cases (such as in translation of formula $\mathsf{G}(\phi \,\mathsf{U}_{\bowtie d}\, \psi)$) into an *infinite* formula graph with an *infinite* number of degradation variables. Hence, we present ideas that allow us to introduce only finite number of degradation variables in the formula graph.

(a) Let ϕ be a translated formula and $\phi \,\mathsf{U}_{\gg d}\, \psi$ its subformula. We allocate exactly one degradation variable x with the subformula $\phi \,\mathsf{U}_{\gg d}\, \psi$ and whenever $\phi \,\mathsf{U}_{\gg d}\, \psi$ appears in $New(q)$ of a node q, we associate it with this x. The question is, how to proceed when variable x is needed to be reset in two different nodes as illustrated in Figure 4.13.

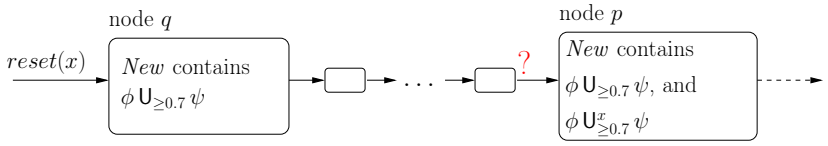


Figure 4.13: We cannot reset x at the edge labeled with ? as we would loose information about the degradation for the first occurrence of the until formula. At the same time we have to reset x there to measure the degradation for the second occurrence of the until formula.

We know that if $\phi \,\mathsf{U}^{x}_{\gg d}\, \psi$ is satisfied in a node q, then also $\phi \,\mathsf{U}_{\gg d}\, \psi$ is satisfied there. This is because the degradation is monotonically decreasing. Hence, whenever both $\phi \,\mathsf{U}_{\gg d}\, \psi \in New(q)$ and $\phi \,\mathsf{U}^{x}_{\gg d}\, \psi \in New(q)$, we remove $\phi \,\mathsf{U}_{\gg d}\, \psi$ from $New(q)$ completely. The satisfaction of $\phi \,\mathsf{U}^{x}_{\gg d}\, \psi$ will ensure satisfaction of $\phi \,\mathsf{U}_{\gg d}\, \psi$.

(b) A bit more complicated situation arises when we consider subformulas $\phi \,\mathsf{U}_{\ll d}\, \psi$.

We know, that if $\phi \,\mathsf{U}_{\ll d}\, \psi$ is satisfied in a node q, then also $\phi \,\mathsf{U}^{x}_{\ll d}\, \psi$ is satisfied there (because the degradation is monotonically decreasing). However, when both $\phi \,\mathsf{U}_{\ll d}\, \psi \in New(q)$ and $\phi \,\mathsf{U}^{x}_{\ll d}\, \psi \in New(q)$, we cannot simply ignore the formula $\phi \,\mathsf{U}^{x}_{\ll d}\, \psi$ and reset the variable x. This is due to the fact, that the complete satisfaction (meaning reaching ψ and satisfying $x \ll d$) of the formula $\phi \,\mathsf{U}^{x}_{\ll d}\, \psi$ can be reached earlier than satisfaction of $\phi \,\mathsf{U}_{\ll d}\, \psi$. This may cause that completing the satisfaction

of $\phi \mathsf{U}_{\ll d} \psi$ formula will be permanently postponed and degradation constraints in the formula graph will not be satisfied even if they should be, as illustrated in Figure 4.14.

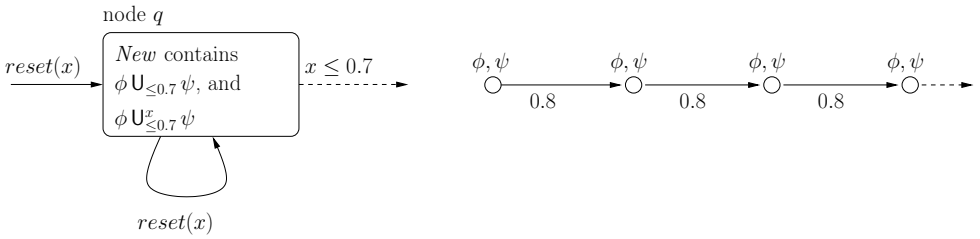


Figure 4.14: When translating a formula $\mathsf{G}(\phi \mathsf{U}_{\leq 0.7} \psi)$, the node q illustrated in left would never be left until the value of degradation variable x drops bellow 0.7. At the same time, x would be reset in every step. That means that reading the trace as produced by the run in right would keep the formula graph in the node q forever. Satisfaction of ψ would never been checked and the trace would not be accepted. However, the run obviously satisfies the formula $\mathsf{G}(\phi \mathsf{U}_{\leq 0.7} \psi)$.

Therefore, we allocate two degradation variables x and y with subformula $\phi \mathsf{U}_{\ll d} \psi$.

Instead of operator $\mathsf{U}^{x}_{\bowtie d}$, we have $\mathsf{U}^{xy}_{\ll d}$, which is defined analogously:

$$\phi \mathsf{U}^{xy}_{\ll d} \psi \equiv (x \ll d \wedge \psi) \vee \mathsf{X}(\phi \mathsf{U}^{xy}_{\ll d} \psi).$$

Similarly, $\phi \mathsf{U}_{\ll d} \psi$ can be then expanded as

$$\phi \mathsf{U}_{\ll d} \psi \equiv (x \ll d \wedge \psi) \vee \mathsf{X}(\phi \mathsf{U}^{xy}_{\ll d} \psi),$$

where x and y are degradation variables with value set to 1.

Assume, that variables x and y are reset in node q (*i.e.*, $\phi \mathsf{U}_{\ll d} \psi \in New(q)$, but $\phi \mathsf{U}^{xy}_{\ll d} \psi \notin New(q)$). Consider a node p, such that $\phi \mathsf{U}_{\ll d} \psi \in New(p)$ and $\phi \mathsf{U}^{xy}_{\ll d} \psi \in New(p)$. Instead of the variable x we reset the variable y. This way, the variable x captures the degradation from the node q, whereas y from the node p. When another node r appears, such that both $\phi \mathsf{U}_{\ll d} \psi \in New(r)$ and $\phi \mathsf{U}^{xy}_{\ll d} \psi \in New(r)$, we again reset y and thus x still captures the degradation from the node q, but y now captures the degradation from the node r. Then, when constraint $x \ll d$ is satisfied and ψ holds, the variables x and y interchange their roles until $y \ll d$ is satisfied. Now, when constraint $y \ll d$ is satisfied and ψ holds, we know that $\phi \mathsf{U}_{\ll d} \psi$ is satisfied not only for r, but also for p. This is illustrated in Figure 4.15.

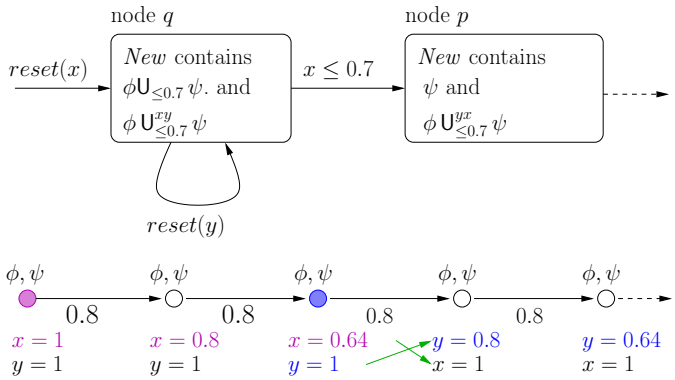


Figure 4.15: Illustration of the $\phi \mathsf{U}_{\ll d} \psi$ case. The value of x is not reset until $x \leq 0.7$. After, its value becomes unimportant and x can be reused to measure degradation from a different point. The value of y is reset whenever we need to start measuring degradation. In the fifth state of the run, we find out, that $\phi \mathsf{U}_{\leq 0.7} \psi$ is satisfied for the third (blue) state. Therefore, it is satisfied also for all states preceding this one.

Very similar ideas as in the case of the constrained until operator $\mathsf{U}_{\bowtie d}$ are followed in the case of the constrained release operator $\mathsf{R}_{\bowtie d}$.

The full details of the translation algorithm are given in the sequel. In the algorithm presentation, we follow the notation given by authors in [GPVW96]. Let us first summarize the data structures we use in Table 4.1 and the list of functions in Table 4.2. The algorithm itself is then presented in Algorithms 5–8.

Id:	unique name of the node;
Inc:	structure coding edges of the graph, *i.e.*, set of triples $(id, constraint, reset)$, such that each triple codes an incoming edge from the *Father* node (id) to the current node. When $id = init$, then the current node is an initial node (*init* is not a name of a node);
Old:	set of formulas, that must hold in the node and have been already processed;
New:	set of formulas, that must hold in the node and have not yet been processed;
Next:	set of formulas that must hold in all immediate successors of nodes satisfying properties in *Old*;
Out:	constraint and reset that must be on an outgoing edge $(constraint, reset)$;
Node:	a structure of form $(Id, Inc, New, Old, Next, Out)$;
Nodes:	the set of all nodes.

Table 4.1: The list of data structures.

$expand(n, Nodes)$:	returns a set of nodes after processing node n;
$new\_ID()$:	returns a fresh id;
$get\_Var(\varphi)$:	returns a set of degradation variables associated with subformula φ;
$Neg(\_)$:	returns a set of negated formulas for a given set of formulas;

Table 4.2: The list of functions.

Algorithm 5 $create\_graph(\varphi)$

1: $q := (new\_ID(), \{(init, true, \emptyset)\}, \{\varphi\}, \emptyset, \emptyset, (true, \emptyset))$
2: **return** $expand(q, \emptyset)$

Algorithm 6 $expand(q, Nodes)$

if $New(q) = \emptyset$ **then**
  if $\exists q' \in Nodes \mid Old(q') = Old(q), Next(q') = Next(q), Out(q') = Out(q)$ **then**
    $Inc(q') = Inc(q') \cup Inc(q)$
    return $Nodes$
  else
    $N = (new\_ID(), \{Id(q)\} \times \{Out(q)\}, Next(q), \emptyset, \emptyset, (\emptyset, true))$
    return $expand(N, Nodes \cup \{q\})$ /* q is a new node */
  end if
else
  switch
  case $\varphi \,\mathsf{U}_{\gg d}\, \psi \in New(q)$ and $\varphi \,\mathsf{U}^{x}_{\gg d}\, \psi \in New(q) \cup Old(q)$
    $New(q) = New(q) \setminus \{\varphi \,\mathsf{U}_{\gg d}\, \psi\}$
    return $expand(q, Nodes)$
  case $\varphi \mathsf{U}_{\ll d} \psi \in New(q)$ and $\varphi \mathsf{U}^{xy}_{\ll d} \psi \in New(q) \cup Old(q)$
    $New(q) = New(q) \setminus \{\varphi \mathsf{U}_{\ll d} \psi, \varphi \mathsf{U}^{xy}_{\ll d} \psi\}$
    $inc1 = \{(id, c \wedge x \ll d \wedge 1 \ll d, r) \mid (id, c, r) \in Inc(q)\}$
    $N1 = (new\_ID(), inc1, New(q) \cup \{\psi\}, Old(q) \cup \{\eta\}, Next(q), Out(q))$
    $inc2 = \{(id, c \wedge x \ll d, r \cup \{y\}) \mid (id, c, r) \in Inc(q)\}; next2 = Next(q) \cup \{\varphi \mathsf{U}^{yx}_{\ll d} \psi\}$
    $N2 = (new\_ID(), inc2, New(q) \cup \{\varphi, \psi\}, Old(q) \cup \{\eta\}, next2, Out(q))$
    $inc3 = \{(id, c, r \cup \{y\}) \mid (id, c, r) \in Inc(q)\}; next3 = Next(q) \cup \{\varphi \,\mathsf{U}^{xy}_{\ll d}\, \psi\}$
    $N3 = (new\_ID(), inc3, New(q) \cup \{\varphi\}, Old(q) \cup \{\eta\}, next3, Out(q))$
    return $expand(N1, expand(N2, expand(N3, Nodes)))$
  case $\varphi \mathsf{R}_{\ll d} \psi \in New(q)$ and $\varphi \mathsf{R}^{x}_{\ll d} \psi \in New(q) \cup Old(q)$
    $New(q) = New(q) \setminus \{\varphi \mathsf{R}_{\ll d} \psi\}$
    return $expand(q, Nodes)$
  case $\varphi \mathsf{R}_{\gg d} \psi \in New(q)$ and $\varphi \mathsf{R}^{x}_{\gg d} \psi \in New(q) \cup Old(q)$
    $New(q) = New(q) \setminus \{\varphi \mathsf{R}^{x}_{\gg d} \psi\}$
    return $expand(q, Nodes)$
  case otherwise **...**

Algorithm 7 $expand(q, Nodes)$ (continued I)

...

...

let η be the longest formula in $New(q)$; $New(q) = New(q) \setminus \{\eta\}$
switch η /* according to the shape of η */
case $(\eta \in (AP \cup Neg(AP) \cup \{true, \neg true\}))$
  if $(\eta == False \vee Neg(\eta) \in Old(q))$ **then**
    return Nodes
  else
    $N = (new\_ID(), Inc(q), New(q), Old(q) \cup \{\eta\}, Next(q), Out(q))$
    return $expand(N, Nodes)$
  end if
case $(\eta \equiv \varphi \vee \psi)$
  $N1 = (new\_ID(), Inc(q), New(q) \cup \{\varphi\}, Old(q) \cup \{\eta\}, Next(q), Out(q))$
  $N2 = (new\_ID(), Inc(q), New(q) \cup \{\psi\}, Old(q) \cup \{\eta\}, Next(q), Out(q))$
  return $expand(N2, expand(N1, Nodes))$ **...**
case $(\eta \equiv \varphi \wedge \psi)$
  $N = (new\_ID(), Inc(q), New(q) \cup \{\varphi, \psi\}, Old(q) \cup \{\eta\}, Next(q), Out(q))$
  return $expand(N, Nodes)$
case $(\eta \equiv \mathsf{X}_{\bowtie d}\, \varphi)$
  $x = get\_Var(\mathsf{X}_{\bowtie d}\, \varphi)$; $inc = \{(id, c, r \cup \{x\}) | (id, c, r) \in Inc(q)\}$
  $out = \{(c \wedge x \bowtie d, r) | (c, r) \in Out(q)\}$
  $N = (new\_ID(), inc, New(q), Old(q) \cup \{\eta\}, Next(q) \cup \{\varphi\}, out)$,
  return $expand(N, Nodes)$
case $(\eta \equiv \varphi\, \mathsf{U}_{\gg d}\, \psi)$
  $x = get\_Var(\varphi\, \mathsf{U}_{\gg d}\, \psi)$
  $inc1 = \{(id, c \wedge 1 \gg d, r) \mid (id, c, r) \in Inc(q)\}$
  $N1 = (new\_ID(), inc1, New(q) \cup \{\psi\}, Old(q) \cup \{\eta\}, Next(q), Out(q))$
  $inc2 = \{(id, c, r \cup \{x\}) \mid (id, c, r) \in Inc(q)\}; next2 = Next(q) \cup \{\varphi\, \mathsf{U}^{x}_{\gg d}\, \psi\}$
  $N2 = (new\_ID(), inc2, New(q) \cup \{\varphi\}, Old(q) \cup \{\eta\}, next2, Out(q))$
  return $expand(N1, expand(N2, Nodes))$
case $(\eta \equiv \varphi\, \mathsf{U}^{x}_{\gg d}\, \psi)$
  $inc1 = \{(id, c \wedge x \gg d, r) \mid (id, c, r) \in Inc(q)\}$
  $N1 = (new\_ID(), inc1, New(q) \cup \{\psi\}, Old(q) \cup \{\eta\}, Next(q), Out(q))$
  $next2 = Next(q) \cup \{\varphi\, \mathsf{U}^{x}_{\gg d}\, \psi\}$
  $N2 = (new\_ID(), Inc(q), New(q) \cup \{\varphi\}, Old(q) \cup \{\eta\}, next2, Out(q))$
  return $expand(N1, expand(N2, Nodes))$
case $(\eta \equiv \varphi\, \mathsf{U}_{\ll d}\, \psi)$
  $(x, y) = get\_Var(\varphi\, \mathsf{U}_{\ll d}\, \psi)$
  $inc1 = \{(id, c \wedge 1 \ll d, r) \mid (id, c, r) \in Inc(q)\}$
  $N1 = (new\_ID(), inc1, New(q) \cup \{\psi\}, Old(q) \cup \{\eta\}, Next(q), Out(q))$
  $inc2 = \{(id, c, r \cup \{x, y\}) \mid (id, c, r) \in Inc(q)\}; next2 = Next(q) \cup \{\varphi\, \mathsf{U}^{xy}_{\ll d}\, \psi\}$
  $N2 = (new\_ID(), inc2, New(q) \cup \{\varphi\}, Old(q) \cup \{\eta\}, next2, Out(q))$
  return $expand(N1, expand(N2, Nodes))$**...**

Algorithm 8 *expand*$(q, Nodes)$ (continued II)

...
  ...
    case $(\eta \equiv \varphi \,\mathsf{U}^{xy}_{\ll d}\, \psi)$
      $inc1 = \{(id, c \wedge x \ll d \wedge y \ll d, r) \mid (id, c, r) \in Inc(q)\}$
      $N1 = (new\_ID(), inc1, New(q) \cup \{\psi\}, Old(q) \cup \{\eta\}, Next(q), Out(q))$
      $inc2 = \{(id, c \wedge x \ll d \wedge, r) \mid (id, c, r) \in Inc(q)\}; next2 = Next(q) \cup \{\varphi \mathsf{U}^{yx}_{\ll d} \psi\}$
      $N2 = (new\_ID(), inc2, New(q) \cup \{\varphi, \psi\}, Old(q) \cup \{\eta\}, next2, Out(q))$
      $inc3 = \{(id, c, r) \mid (id, c, r) \in Inc(q)\}; next3 = Next(q) \cup \{\varphi \,\mathsf{U}^{xy}_{\ll d}\, \psi\}$
      $N3 = (new\_ID(), inc3, New(q) \cup \{\varphi\}, Old(q) \cup \{\eta\}, next3, Out(q))$
      return $expand(N1, expand(N2, expand(N3, Nodes)))$
    case $(\eta \equiv \varphi \,\mathsf{R}_{\bowtie d}\, \psi)$
      $x = get\_Var(\varphi \,\mathsf{R}_{\bowtie d}\, \psi)$
      $inc1 = \{(id, c \wedge \neg(1 \bowtie d), r) \mid (id, c, r) \in Inc(q)\}$
      $N1 = (new\_ID(), inc1, New(q) \cup \{\phi\}, Old(q) \cup \{\eta\}, Next(q), Out(q))$ **...**
      $N2 = (new\_ID(), Inc(q), New(q) \cup \{\varphi \wedge \psi\}, Old(q) \cup \{\eta\}, Next(q), Out(q))$
      $inc3 = \{(id, c \wedge \neg(1 \bowtie d), r \cup \{x\}) \mid (id, c, r) \in Inc(q)\}$
      $N3 = (new\_ID(), inc3, New(q), Old(q) \cup \{\eta\}, Next(q) \cup \{\varphi \,\mathsf{R}^{x}_{\bowtie d}\, \psi\}, Out(q))$
      $inc4 = \{(id, c, r \cup \{x\}) \mid (id, c, r) \in Inc(q)\}; next4 = Next(q) \cup \{\varphi \,\mathsf{R}^{x}_{\gg d}\, \psi\}$
      $N4 = (new\_ID(), inc4, New(q) \cup \{\psi\}, Old(q) \cup \{\eta\}, next4, Out(q))$
      return $expand(N1, expand(N2, expand(N3, expand(N4, Nodes))))$
    case $(\eta \equiv \varphi \,\mathsf{R}^{x}_{\bowtie d}\, \psi)$
      $x = get\_Var(\varphi \,\mathsf{R}_{\bowtie d}\, \psi)$
      $inc1 = \{(id, c \wedge \neg(x \bowtie d), r) \mid (id, c, r) \in Inc(q)\}$
      $N1 = (new\_ID(), inc1, New(q) \cup \{\phi\}, Old(q) \cup \{\eta\}, Next(q), Out(q))$
      $N2 = (new\_ID(), Inc(q), New(q) \cup \{\varphi \wedge \psi\}, Old(q) \cup \{\eta\}, Next(q), Out(q))$
      $inc3 = \{(id, c \wedge \neg(x \bowtie d), r) \mid (id, c, r) \in Inc(q)\}$
      $N3 = (new\_ID(), inc3, New(q), Old(q) \cup \{\eta\}, Next(q) \cup \{\varphi \,\mathsf{R}^{x}_{\bowtie d}\, \psi\}, Out(q))$
      $next4 = Next(q) \cup \{\varphi \,\mathsf{R}^{x}_{\gg d}\, \psi\}$
      $N4 = (new\_ID(), Inc(q), New(q) \cup \{\psi\}, Old(q) \cup \{\eta\}, next4, Out(q))$
      return $expand(N1, expand(N2, expand(N3, expand(N4, Nodes))))$
    end switch
  end switch
end if

The correctness of the formula graph construction is summarized in the following two lemmas that follow from the above description and the key identities presented in Table 4.3.

Lemma 4.4.29. *Given a formula ϕ, only a finite number of degradation variables is needed to build a formula graph for the formula.*

Lemma 4.4.30. *The expansion of each node of the formula graph corresponds to the semantics of the expanded subformula.*

$r \models \varphi \,\mathsf{U}\, \psi \iff r \models \psi \vee (\varphi \wedge \mathsf{X}(\varphi \,\mathsf{U}\, \psi))$

$r \models \varphi \,\mathsf{R}\, \psi \iff r \models (\varphi \wedge \psi) \vee (\psi \wedge \mathsf{X}(\varphi \,\mathsf{R}\, \psi))$

$r \models \varphi \,\mathsf{U}_{\bowtie d}\, \psi \iff r \models (x \bowtie d \wedge \psi) \vee r \models \varphi \wedge \mathsf{X}(\varphi \,\mathsf{U}^x_{\bowtie d}\, \psi)$

Proof: let us first define the semantics of the operator $\mathsf{U}^x_{\bowtie d}$

$$r^k \models \varphi \,\mathsf{U}^x_{\bowtie d}\, \psi \iff \exists j \geq k.(\mathbb{D}_r(j) \bowtie d \wedge r^j \models \psi \wedge \forall k \leq i < j.(r^i \models \varphi))$$

the identity for the operator $\mathsf{U}^x_{\bowtie d}$ can be expanded as

$$\begin{aligned} r^k \models \varphi \,\mathsf{U}^x_{\bowtie d}\, \psi &\iff \exists j \geq k.(r^j \models (x \bowtie d \wedge \psi) \wedge \forall k \leq i < j.(r^i \models \varphi)) \\ &\iff r^k \models (x \bowtie d \wedge \psi) \vee \\ &\qquad \exists k+1 \leq j.(r^j \models (x \bowtie d \wedge \psi) \wedge \forall k+1 \leq i < j.(r^i \models \varphi)) \\ &\iff r^k \models (x \bowtie d \wedge \psi) \vee r^k \models \varphi \wedge \mathsf{X}(\varphi \,\mathsf{U}^x_{\bowtie d}\, \psi) \end{aligned}$$

the identity for the operator $\mathsf{U}_{\bowtie d}$ can be expanded as

$$\begin{aligned} r \models \varphi \,\mathsf{U}_{\bowtie d}\, \psi &\iff \\ r^0 \models \varphi \,\mathsf{U}_{\bowtie d}\, \psi &\iff \exists j.(\mathbb{D}_r(j) \bowtie d \wedge r^j \models \psi \wedge \forall 0 \leq i < j.(r^i \models \varphi)) \\ &\iff \exists j.(r^j \models (x \bowtie d \wedge \psi) \wedge \forall 0 \leq i < j.(r^i \models \varphi)) \\ &\iff r \models (x \bowtie d \wedge \psi) \vee \exists 1 \leq j.(r^j \models (x \bowtie d \wedge \psi) \wedge \forall 1 \leq i < j.(r^i \models \varphi)) \\ &\iff r \models (x \bowtie d \wedge \psi) \vee (\varphi \wedge \mathsf{X}(\varphi \,\mathsf{U}^x_{\bowtie d}\, \psi)) \end{aligned}$$

$r \models \varphi \,\mathsf{R}_{\bowtie d}\, \psi \iff r \models \varphi \wedge (x \bowtie d \Rightarrow \psi) \vee r \models (x \bowtie d \Rightarrow \psi) \wedge \mathsf{X}(\varphi \,\mathsf{R}^x_{\bowtie d}\, \psi)$

Proof: let us first define the semantics of the operator $\mathsf{R}^x_{\bowtie d}$

$$r^k \models \varphi \,\mathsf{R}^x_{\bowtie d}\, \psi \iff \forall j.(\mathbb{D}_r(j) \bowtie d \Rightarrow (r^j \models \psi \vee \exists 0 \leq i < j.(r^i \models \varphi))$$

the identity for the operator $\mathsf{R}^x_{\bowtie d}$ can be expanded as

$$\begin{aligned} r^k \models \varphi \,\mathsf{R}^x_{\bowtie d}\, \psi &\iff \forall j.((r^j \models x \bowtie d) \Rightarrow (r^j \models \psi \vee \exists 0 \leq i < j.(r^i \models \varphi)) \\ &\iff r^k \models \varphi \wedge (x \bowtie d \Rightarrow \psi) \vee \\ &\qquad r^k \models (x \bowtie d \Rightarrow \psi) \wedge \\ &\qquad\quad \forall k+1 \leq j.((r^j \models x \bowtie d) \Rightarrow (r^j \models \psi \vee \exists 0 \leq i < j.(r^i \models \varphi)) \\ &\iff r^k \models \varphi \wedge (x \bowtie d \Rightarrow \psi) \vee r^k \models (x \bowtie d \Rightarrow \psi) \wedge \mathsf{X}(\varphi \,\mathsf{R}^x_{\bowtie d}\, \psi) \end{aligned}$$

the identity for the operator $\mathsf{R}_{\bowtie d}$ can be expanded as

$$\begin{aligned} r \models \varphi \,\mathsf{R}_{\bowtie d}\, \psi &\iff \\ r^0 \models \varphi \,\mathsf{R}_{\bowtie d}\, \psi &\iff \forall j.(\mathbb{D}_r(j) \bowtie d \Rightarrow (r^j \models \psi \vee \exists 0 \leq i < j.(r^i \models \varphi)) \\ &\iff \forall j.(x \bowtie d \Rightarrow (r^j \models \psi \vee \exists 0 \leq i < j.(r^i \models \varphi)) \\ &\iff r \models (x \bowtie d \Rightarrow \psi) \wedge \forall 1 \leq j.r^j \models (x \bowtie d \Rightarrow (r^j \models \psi \vee \exists 0 \leq i < j.(r^i \models \varphi))) \\ &\iff r \models \varphi \wedge (x \bowtie d \Rightarrow \psi) \vee r \models (x \bowtie d \Rightarrow \psi) \wedge \mathsf{X}(\varphi \,\mathsf{R}^x_{\bowtie d}\, \psi) \end{aligned}$$

Table 4.3: Table of key identities.

Formula Graph to Generalized BADC

Definition 4.4.31 (Generalized BADC)**.** *A generalized Büchi automaton with degradation constraints is a tuple $\mathcal{G} = (Q, Q_{init}, \Sigma, X, \delta, \mathcal{F})$, where Q, Q_{init}, Σ, X, and δ are defined as in BADC, and $\mathcal{F} \subseteq 2^{2^Q}$ is a generalized Büchi acceptance condition. $\mathcal{F} = \{F_1, \ldots, F_n\}$ is a set of sets of accepting states.*

A run is defined exactly the same as for a BADC. An *accepting run* of a

generalized BADC is a run $\rho = (q_0, \nu_0)(q_1, \nu_1)\ldots$, such that for each acceptance set $F \in \mathcal{F}$ it holds that $q_i \in F$ for infinitely many indexes i.

A generalized BADC is obtained from the formula graph as follows:

- The set of states Q is the set of nodes with a new initial node called *init*
- $\Sigma = 2^{AP}$
- $\tau = (q, \sigma, \gamma, R, q') \in \delta$ if there exists an edge e between q and q' labeled with input $\sigma \in \Sigma$ which satisfies restriction given by the set $((AP \cup \neg AP) \cap Old(q'))$. γ and R are the degradation constraints and resets associated with the edge e, respectively.
- $\mathcal{F} = \{F_1, \ldots, F_n, F'_1, \ldots, F'_n\}$. For each $\phi \,\mathsf{U}\, \psi$ subformula we define $F_i = \{q \in Q \mid \phi\, U\, \psi \notin Old(q) \vee \psi \in Old(q)\}$, For each $\phi \,\mathsf{U}_{\bowtie d}\, \psi$ subformula associated with degradation variable x (or variables x, y) we define $F'_i = \{q \in Q \mid$ none of $\phi\, U_{\bowtie d}\, \psi$, $\phi\, U^{x}_{\bowtie d}\, \psi$, $\phi\, U^{xy}_{\bowtie d}\, \psi$, $\phi\, U^{yx}_{\bowtie d}\, \psi$ belongs to $Old(q)$, or $\psi \in Old(q)\}$

Generalized BADC to BADC

The translation from the generalized BADC to BADC follows the same principles as the translation from the generalized BA to BA. The key idea is to make a separate layer of the BADC for each set acceptance $F \in \mathcal{F}$. In i-th layer, only the set F_i is accepting. When a state from F_i is reached, the computation is moved to the $(i+1)$-th layer (or, more precisely $((i+1) \bmod |\mathcal{F}|)$-th layer). This way we force the resulting Büchi automaton to visit each of the set $F \in \mathcal{F}$ infinitely many times.

Lemma 4.4.32 (DLTL to BADC Translation Correctness)**.** *The above presented algorithm is sound and complete and gives an answer to the DLTL to BADC translation problem (Problem 4.4.28).*

Proof. Follows from the correctness of construction in [GPVW96], proof of Lemma 4.4.29, proof of Lemma 4.4.30, and the key identities. □

Note, that because the number of degradation variables needed to capture a single degradation constraint is constant, the worst case complexity of the overall translation remains exponential as in [GPVW96].

The solution to the satisfying trace synthesis problem with specification given as a half-bounded DLTL formula is now completed: After the translation of the DLTL formula into a BADC, the algorithm from Section 4.4.3 is applied and the result is directly obtained.

4.5 Probability Viewed as a Degrading Quality

In this section we study probability viewed as a degrading quality. First, we show that Markov Decision Processes (see Def. 3.7.1) are a specialized version of transition systems with degradation and that there exist two MDPs that are indistinguishable by any LTL, PCTL, or PCTL* formula. Second, we show that the two MDPs that can be distinguished with a DLTL formula. Thus, we prove the incomparable expressiveness of DLTL with respect to LTL, PCTL, and PCTL*.

Let $\mathcal{T} = (S, S_{\text{init}}, Act, T, AP, L, D)$ be a transition system with degradation subject to the following restrictions.

- $\forall s \in S, \forall \alpha \in Act : \sum_{t=(s,\alpha,s')\in T} D(t) \in \{0, 1\}$, and
- S_{init} is a singleton.

When we think of probability as of a system quality that degrades in time, the transition systems with degradation restricted as above are syntactically equivalent to Markov decision processes, where the initial state s_{init} is the state in S_{init} and the probability transition function $P : S \times Act \times S \to [0, 1]$ is given as $P(s, \alpha, s') = D\big((s, \alpha, s')\big)$, for all $(s, \alpha, s') \in T$, and $P(s, \alpha, s') = 0$, for all $(s, \alpha, s') \notin T$.

Let us now consider MDPs $\mathcal{M}$ and $\mathcal{M}'$, as illustrated in Figure 4.16.

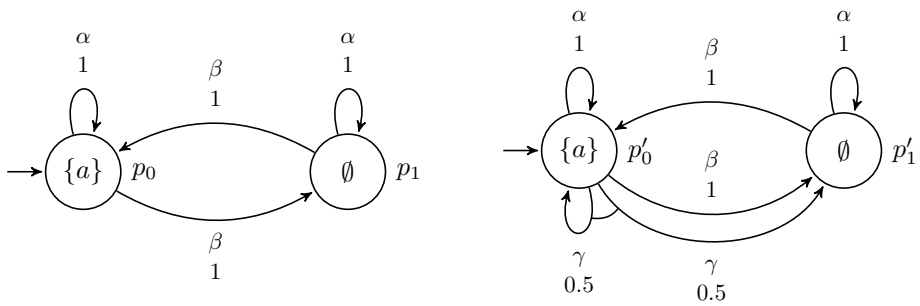


Figure 4.16: MDPs $\mathcal{M}$ (left) and $\mathcal{M}'$ (right) indistinguishable by any LTL, PCTL, or PCTL* formula.

First, for the MDP $\mathcal{M} = (S = \{s_0, s_1\}, s_0, Act = \{\alpha, \beta\}, P, \{a\}, L)$ in Figure 4.16 (left), we show that the minimal and the maximal probability of an arbitrary set of traces starting at a particular state and satisfying a linear temporal property is always either 0 or 1.

Observation 4.5.1. *Let η be an arbitrary scheduler for $\mathcal{M}$. Then the Markov chain induced by η is $\mathcal{M}_\eta = (S^+, p_0, P_\eta, AP, L_\eta)$, where*

$$P_\eta(s_0 \dots s_n, s_0 \dots s_n s_{n+1}) = \begin{cases} 1 & \text{if } s_n = s_{n+1} \text{ and } \eta(s_0 \dots s_n) = \alpha \text{ or} \\ & \text{if } s_n \neq s_{n+1} \text{ and } \eta(s_0 \dots s_n) = \beta \\ 0 & \text{otherwise,} \end{cases}$$

and $L_\eta(s_0 \dots s_n) = L(s_n)$.

Note that for each state $s_0 \dots s_n$ in $\mathcal{M}_\eta$ there is exactly one state $s_0 \dots s_n s_{n+1}$ such that $P_\eta(s_0 \dots s_n, s_0 \dots s_n s_{n+1}) > 0$. In other words, there is exactly one trace $r = (s_0)(s_0 s_1)(s_0 s_1 s_2) \dots$ of $\mathcal{M}_\eta$ starting at s_0. The set of all traces starting at s_0 in $\mathcal{M}_\eta$ is $\{r\}$ and its probability is 1. Similarly, there is exactly one trace $r = (s_0 s_1)(s_0 s_1 s_2)(s_0 s_1 s_2 s_3) \dots$ of $\mathcal{M}_\eta$ starting at $s_0 s_1$ and the probability of the set of all traces $\{r\}$ starting at $s_0 s_1$ is equal to 1.

Let us consider the language $\mathcal{L}$ over the alphabet $2^{\{a\}}$ representing a set of all words satisfying a linear temporal property. For each symbol $\sigma \in 2^{\{a\}}$ we distinguish three possible cases:

(i) there is no word in $\mathcal{L}$ starting with σ, *i.e.,* $\mathcal{L} \cap \{\sigma \cdot w \mid w \in 2^{\{a\}}\} = \emptyset$

(ii) $\mathcal{L}$ contains all words starting with σ, *i.e.,* $\{\sigma \cdot w \mid w \in 2^{\{a\}}\} \subseteq \mathcal{L}$, or

(iii) $\exists\, w_1 = \sigma \cdot v_1 \in \mathcal{L}$ and $\exists\, w_2 = \sigma \cdot v_2 \notin \mathcal{L}$, *i.e.,*
$\{\sigma \cdot w \mid w \in 2^{\{a\}}\} \cap \mathcal{L} \neq \emptyset \wedge \{\sigma \cdot w \mid w \in 2^{\{a\}}\} \cap co - \mathcal{L} \neq \emptyset$

To simplify the following discussion we denote by symbol u the state p_0 of $\mathcal{M}$ in case of $\sigma = \{a\}$ and the state p_1 otherwise, *i.e.,* if $\sigma = \emptyset$.

Lemma 4.5.2. *Suppose there is no word in $\mathcal{L}$ starting with the symbol σ. Then both the minimal and the maximal probability of the set of traces of $\mathcal{M}$ starting at u with words in $\mathcal{L}$ is 0. In other words, there is no scheduler μ for $\mathcal{M}$, such that the word produced by the trace under μ belongs to $\mathcal{L}$.*

Proof. The proof follows directly from the fact that each trace $r = u\alpha_0 s_1 \alpha_1 \dots$ of $\mathcal{M}$ that starts at u produces the word $L(u)L(s_1)L(s_2) \dots \notin \mathcal{L}$. □

Lemma 4.5.3. *Suppose $\mathcal{L}$ contains all words starting with σ. Then the minimal and the maximal probability of the set of traces of $\mathcal{M}$ starting at u with words in $\mathcal{L}$ is 1.*

Proof. For each trace $r = u\alpha_0 s_1 \alpha_1 s_2 \alpha_2 \dots$ of $\mathcal{M}$ starting at u it holds that the produced word $L(u)L(s_1)L(s_2) \dots = \gamma L(s_1)L(s_2) \dots$ is present in $\mathcal{L}$. Thus the probability of the set of traces starting at u with words in $\mathcal{L}$ is 1 for all schedulers η for $\mathcal{M}$. □

Lemma 4.5.4. *Let us suppose that there exist $w_1 = \sigma \cdot v_1 \in \mathcal{L}$ and $w_2 = \sigma \cdot v_2 \notin \mathcal{L}$ for some $v_1, v_2 \in (2^{AP})^\omega$. Then the maximal probability of the set of traces of $\mathcal{M}$ starting at u with words in $\mathcal{L}$ is 1 and the minimal probability is 0.*

Proof. We define schedulers η_1 and η_2 for $\mathcal{M}$ such that the words produced by the traces starting at u in the induced Markov chain $\mathcal{M}_{\eta_1}$ and $\mathcal{M}_{\eta_2}$ are w_1 and w_2, respectively.

Let $w_1 = \sigma\sigma_1\sigma_2 \ldots$. The scheduler η_1 is defined by the prescription (assuming that $s_0 = u$)

$$\eta_1(s_0 \ldots s_n) = \begin{cases} \alpha & \text{if } L(s_0 \ldots s_n) = \sigma_{n+1} \\ \beta & \text{if } L(s_0 \ldots s_n) \neq \sigma_{n+1}. \end{cases}$$

The scheduler η_1 unambiguously determines the only trace in $\mathcal{M}_{\eta_1}$ and the word produced by this trace is w_1. This fact together with Observation 4.5.1 implies that the maximal probability of the set of traces of $\mathcal{M}$ starting at u with words in $\mathcal{L}$ is 1.

For the minimal probability and the scheduler η_2 the arguments are similar. □

We summarize the results given by Lemmas 4.5.2 - 4.5.4 in Table 4.4. Note that any language $\mathcal{L}$ satisfies exactly one of the three cases given on the first three lines of the table and exactly one of the three cases given on the second three lines of the table. Therefore, given an arbitrary language $\mathcal{L}$, the minimal and the maximal probability of the set of traces of $\mathcal{M}$ that produce words from $\mathcal{L}$ can be completely determined using the table.

	min	max
	traces from p_0	
$\mathcal{L} \cap \{\{a\} \cdot w \mid w \in 2^{\{a\}}\} = \emptyset$	0	0
$\{\{a\} \cdot w \mid w \in 2^{\{a\}}\} \subseteq \mathcal{L}$	1	1
$\{\{a\} \cdot w \mid w \in 2^{\{a\}}\} \cap \mathcal{L} \neq \emptyset \wedge \{\{a\} \cdot w \mid w \in 2^{\{a\}}\} \cap co-\mathcal{L} \neq \emptyset$	0	1
	traces from p_1	
$\mathcal{L} \cap \{\emptyset \cdot w \mid w \in 2^{\{a\}}\} = \emptyset$	0	0
$\{\emptyset \cdot w \mid w \in 2^{\{a\}}\} \subseteq \mathcal{L}$	1	1
$\{\emptyset \cdot w \mid w \in 2^{\{a\}}\} \cap \mathcal{L} \neq \emptyset \wedge \{\emptyset \cdot w \mid w \in 2^{\{a\}}\} \cap co-\mathcal{L} \neq \emptyset$	0	1

Table 4.4: Summary of results, Lemma 4.5.2 - 4.5.4

Let us now consider the MDP $\mathcal{M}'$ in Figure 4.16 (right). Using similar arguments as for the MDP $\mathcal{M}$ we obtain the very same results about probability bounds for all languages for the MDP $\mathcal{M}'$. For the summary see again Table 4.4.

The minimal and the maximal probabilities of the set of traces starting at the initial states p_0 and p'_0 producing words in $\mathcal{L}$ are the same for the MDP $\mathcal{M}$ and the MDP $\mathcal{M}'$, respectively. The same observation holds for the states p_1 and p'_1. Thus there is a one-to-one correspondence between the states p_0 and p'_0 and also between the states p_1 and p'_1. Therefore MDPs $\mathcal{M}$ and $\mathcal{M}'$ cannot be distinguished neither by qualitative verification nor by quantitative verification with any LTL formula.

Furthermore, if ϕ is a CTL path formula then both the minimal and the maximal probability of the set of traces satisfying ϕ is always either 0 or 1 for all the states both in $\mathcal{M}$ and $\mathcal{M}'$. Hence, for any PCTL or PCTL* formula $P_{\bowtie p}\phi$ it holds that $\mathcal{M} \models P_{\bowtie p}\phi \Leftrightarrow \mathcal{M}' \models P_{\bowtie p}\phi$. As a result, the difference between $\mathcal{M}$ and $\mathcal{M}'$ cannot be captured by any PCTL or PCTL* formula.

Now we are to define a quantitative linear property which allows us to distinguish the Markov decision processes $\mathcal{M}$ and $\mathcal{M}'$. A DLTL formula

$$\neg(\mathsf{X}_{[0,0.7]}\neg a)$$

is satisfied in $\mathcal{M}$, but not in $\mathcal{M}'$. In $\mathcal{M}$, there exists no run satisfying $\mathsf{X}_{[0,0.7]}\neg a$, because the level of degradation between $r(0)$ and $r(1)$ is 1 for all traces r. On the other hand, there is a trace $r' = p'_0 p'_0 \ldots$ of $\mathcal{M}'$ satisfying $\mathsf{X}_{[0,0.7]}\neg a$. The level of degradation between $r'(0)$ and $r'(1)$ is 0.5.

DLTL allows us to capture quite different aspects than the usual probabilistic logics. Whereas there, we look for probability of a whole set of traces that satisfy a given property, in DLTL approach we aim at prefixes of individual traces and measure how much the probability degrades if the prefix is extended with a transition. In other words, with DLTL we are able to express the amount of contribution to the target probability of a set of traces that is brought by the set of traces exhibiting the same finite prefix. Furthermore, unlike in probabilistic LTL, the requirements on the level of degradation can be nested in DLTL formulas.

4.6 Conclusions

In this chapter we have aimed at quantitative properties of systems with degradation. Degradation phenomenon is an inherent property of many computer-based systems and by suggesting a modeling, specification and verification framework, we allow for capturing and analysis new kind of system qualities, that, to our best knowledge, have not been addressed before.

We have introduced transition systems with degradation as a modeling formalism and a version of linear temporal logic that allows for specification of requirements on the level of degradation of individual system traces. We have

shown a connection between systems with degradation and timed automata and we use MITL model checking algorithm to solve DLTL model checking and deterministic control strategy synthesis problem. The solution suffers from two drawbacks. First, the verification problem can be answered only with limited precision, and second, higher precision causes rapidly higher computational demands. We have addressed these issues by introducing an automata-like formalism called Büchi automata with degradation constraints and a full-precision verification algorithm against this type of specification. Further, for half-bounded DLTL formulas, we have developed a translation algorithm to BADC, completing thus the solution to model-checking problem for the half-bounded DLTL fragment. The model checking methods we have suggested at the same time answer the deterministic control strategy synthesis problem. Furthermore, we have shown how to interpret probability as a degrading quality and that the suggested DLTL can be used to reason about MDPs and that the expressive power of DLTL is incomparable with the expressivness of probabilistic LTL, PCTL and PCTL*.

A possible future research directions include introduction of a direct translation process from all DLTL formulas into BADCs, control strategy synthesis for nondeterministic systems with degradation from DLTL specifications as well as dealing with continuous and hybrid systems with degradation.

4.7 Notes

The material presented in this chapter builds on joint works with the author's advisor Ivana Černá and the author's consultant Jiří Barnat [BČT12b, BČT12a, BČT09]. The author of this thesis has been the leading author of all of these works. The concept of systems with degradation has been introduced in [BČT09] as well as the verification procedure for the case when the specification is given in the form of a BADC and the BADC normalization. The timed automata approach has been presented in [BČT12b, BČT12a] and the translation from DLTL to BADC has appeared in [BČT12b]. The definition of BADCs, the normalization procedure and a part of the section analyzing the relation of systems with degradation to MDPs have also been presented in the author's Master Thesis [Tům].

Chapter 5

Robot Path Planning for Optimal Surveillance

In this chapter, we aim to formalize and solve a particular control strategy synthesis problem motivated by scenarios arising in autonomous mobile robotics. Namely, we focus on searching for an optimal path for a robot given a high-level LTL specification of its mission. We consider that the mission specification must repeatedly reach a so-called optimizing proposition and our aim is to minimize the maximum time between the successive satisfying instances of the optimizing proposition.

First, we motivate and informally introduce the problem through a data gathering robot example in Section 5.1. The formal statement of the problem including the definition of the robot motion model and the form of the robotic mission specification follows in Section 5.2.1. Next, in Section 5.2.2, we provide the solution to the problem exploiting ideas from automata-based model checking. Unlike in the "ordinary" model checking and control strategy synthesis, where an arbitrary accepting run of a product automaton is sought, here we look for a run that is optimal with respect to a carefully tailored cost function. We have implemented and experimentally evaluated the suggested framework on a robotic testbed. The results of the experiments are summarized in Section 5.2.4.

Finally, in Section 5.3, we conclude with several remarks and directions for future work.

5.1 Motivation

In robot path planning with a complex high-level specification, the basic task to solve is to find an arbitrary path that satisfies the specification. The goal of this chapter is to plan the *optimal* motion of a robot subject to temporal logic constraints. In particular, our choice of the cost function to be optimized is motivated by problems in monitoring and data gathering. In short, the cost function quantifies the time between satisfying instances of a single *optimizing proposition*. This problem arises in many applications where a mobile robot has to perform a sequence of operations subject to external constraints.

Example 5.1.1 (Data Gathering Robot II)**.** *In a persistent data gathering task, a mobile robot is required to periodically gather data at several locations and then visit a different set of upload sites to transmit the data. Referring to our experimental testbed illustrated Figure 5.1, we would like to enable tasks such as "Repeatedly gather data at locations* g_1*,* g_2*, and* g_3*. Upload data at either* u_1 *or* u_2 *after each data-gather. Follow the road rules, and avoid the road connecting* i_4 *to* i_2*." We wish to determine a robot motion that completes the task and minimizes a cost function, such as the maximum time between data uploads.*

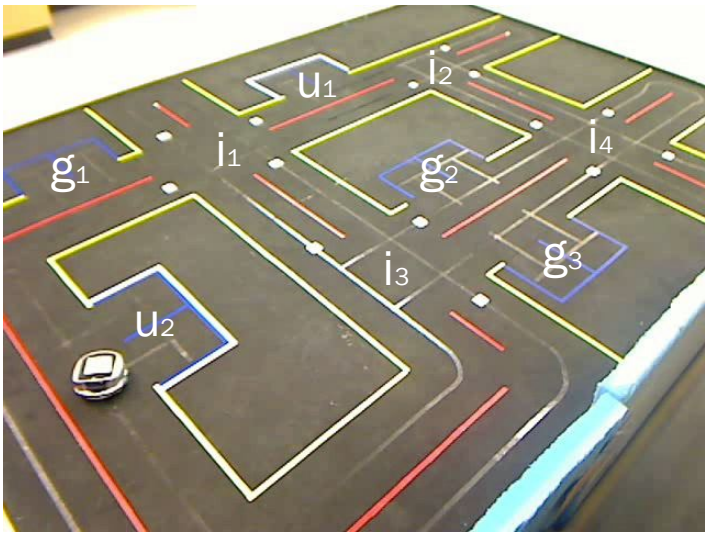


Figure 5.1: A robotic environment consisting of roads, intersections and parking lots.

5.2 Optimal Surveillance Problem

5.2.1 Problem Statement

Consider a single robot in an arbitrary environment, modeled as a weighted deterministic transition system $\mathcal{T} = (S, s_{\text{init}}, T, AP, L, W)$ (as defined in Section 3.2). Each state of the transition system corresponds to a region of the environment, and each transition represents the capability of the robot to move in between the regions. A trace of the transition system starting at s_{init} defines the corresponding path of the robot in the environment. We assume that there are no transitions leading from and to the states representing regions where obstacles are present. Therefore, each trace of the robot is obstacle-free. The time to take transition $(s, s') \in T$ (*i.e.,* the time for the robot to travel from s to s' in the environment) is given by $W\big((s, s')\big)$.

Example 5.2.1 (Data Gathering Robot II)**.** *Modeling the motion of the vehicle in the road network from Example 5.1.1 is depicted in Figure 5.2 and proceeds as follows. The set of states S is the set of labels assigned to the intersections, parking lots, and branching points between the roads and parking lots. The transition relation T shows how the regions are connected and the transitions' labels give distances between them (measured in centimeters). In our testbed the robot moves at constant speed, and thus the distances and travel times are equivalent. The robot follows basic road rules, i.e., it can only move on the right hand lane of a road and it cannot make a U-turn at an intersection. To capture this, we model each intersection as four different states. Note that, in reality, each state in S has associated a set of motion primitives, and the selection of a motion primitive (e.g., go_straight, turn_right) determines the transition to one unique next states. This motivates our assumption that the weighted transition system is deterministic, and therefore its inputs can be removed.*

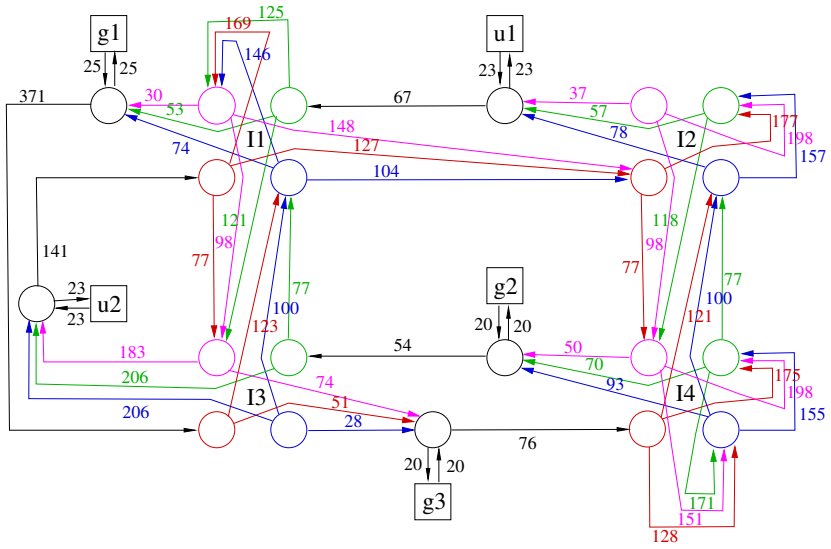


Figure 5.2: The weighted transition system for the road network in Figure 5.1.

To define our problem, we assume that there is an atomic proposition $\pi_{\text{opt}} \in AP$, called the *optimizing proposition*. We consider the specification of the robotic mission given as an LTL formula of the form

$$\phi := \varphi \wedge \mathsf{G}\,\mathsf{F}\,\pi_{\text{opt}}. \tag{5.1}$$

The formula φ can be any LTL formula over AP. The second part of the formula specifies that the proposition π_{opt} must be satisfied infinitely often. In other

words, one of the regions labeled with π_{opt} has to be periodically surveyed. The choice of such a tasks will ensure well-posedness of our optimization.

Let each trace r of $\mathcal{T}$ start at time $\mathsf{t} = 0$, and assume that there is at least one trace satisfying an LTL formula ϕ in the form of Eq. (5.1). Associated with each trace $r = s_0 s_1 \ldots$ there is a sequence of time instances $\mathbb{T} := \mathsf{t}_0\mathsf{t}_1\mathsf{t}_2 \ldots$, where $\mathsf{t}_0 = 0$, and t_i denotes the time at which state s_i is reached ($\mathsf{t}_{i+1} = \mathsf{t}_i + W\big((s_i, s_{i+1})\big)$, for all $i \geq 0$). From this time sequence we can extract all time instances at which the proposition π_{opt} is satisfied. We let $\mathbb{T}_{\pi_{\text{opt}}}$ denote the sequence of satisfying instances of the proposition π_{opt}.

Our goal is to synthesize an infinite trace r (*i.e.,* a robot path) satisfying LTL formula in the form of Eq. (5.1), and minimizing the cost function

$$Cost(r) = \limsup_{i \to +\infty} \left(\mathbb{T}_{\pi_{\text{opt}}}(i+1) - \mathbb{T}_{\pi_{\text{opt}}}(i) \right), \tag{5.2}$$

where $\mathbb{T}_{\pi_{\text{opt}}}(i)$ is the ith satisfying time instance of proposition π_{opt}. Note that a finite cost in (5.2) enforces that $\mathsf{G}\,\mathsf{F}\,\pi_{\text{opt}}$ is satisfied. Thus, the specification that appears in ϕ ensures that any satisfying run has finite cost.

We are now ready to formally state our problem.

Problem Statement 5.2.2 (Optimal Surveillance)**.** *Given*

- *a weighted deterministic transition system* $\mathcal{T} = (S, s_{\text{init}}, T, AP, L, W)$*;*
- *an LTL over its set of atomic propositions* AP *in form* (5.1)*; and*
- *an optimizing proposition* $\pi_{\text{opt}} \in AP$*,*

find a trace r *of* $\mathcal{T}$ *minimizing the cost* $Cost(r)$ *in* (5.2)*.*

Remark 5.2.3 (Comments on Problem Statement)**.** *LTL formula form: Suppose that we want to minimize the time between satisfying instances of a disjunction of propositions* $\bigvee_{i \in I} \pi_i$*. We can write this in the formula* (5.1) *by defining a new proposition* π_{opt} *which is satisfied at each state in which a* π_i *is satisfied.*

Cost function form: The transition system produces infinite runs and cost function (5.2) *evaluates the steady-state time between satisfying instances of* π_{opt}*. This form of the cost is motivated by persistent monitoring tasks, where we seek to optimize the long-term behavior. In the upcoming sections we design an algorithm which minimizes the time to reach the optimal steady-state: Thus, the runs produced will achieve the cost in* (5.2) *in finite time. In addition, in Remark 5.2.3 we discuss how we can optimize alternative cost functions that consider both transient and steady-state behavior.*

5.2.2 Problem Solution

In this section we describe our solution to the optimal surveillance problem (Problem 5.2.2). We leverage ideas from the automata-theoretic approach to model checking. In short, we first define a weighted product automaton and then cast a graph algorithm to find an optimal path that projects onto an optimal trace of the robot's transition system.

The Weighted Product Automaton

Consider a deterministic weighted transition system $\mathcal{T} = (S, s_{\text{init}}, T, AP, L, W)$, a proposition $\pi_{\text{opt}} \in AP$, and an LTL formula $\phi = \varphi \wedge \mathsf{GF}\,\pi_{\text{opt}}$ over AP in form (5.1), translated into a Büchi automaton $\mathcal{B}_\phi = (Q, q_{\text{init}}, \Sigma, \delta, F)$. Inspired by the construction of standard product automaton as in Section 3.5, we introduce a *weighted product automaton*, that is suitably defined for our problem.

Definition 5.2.4 (Product Automaton for Optimal Surveillance)**.** *The product automaton $\mathcal{P} = \mathcal{T} \otimes \mathcal{B}_\phi$ of the transition system $\mathcal{T}$ and the Büchi automaton $\mathcal{B}_\phi$ is defined as the tuple $\mathcal{P} = (Q_\mathcal{P}, q_\mathcal{P}^{\text{init}}, \delta_\mathcal{P}, F_\mathcal{P}, W_\mathcal{P}, Q_\mathcal{P}^{\pi_{\text{opt}}})$, where*

- *$Q_\mathcal{P} = S \times Q$ is a finite set of states;*
- *$q_\mathcal{P}^{\text{init}} = (s_{\text{init}}, q_{\text{init}})$ is the initial state;*
- *$\delta_\mathcal{P} \subseteq Q_\mathcal{P} \times Q_\mathcal{P}$ is a transition relation, where $\big((s,q),(s',q')\big) \in \delta_\mathcal{P}$ if and only if*
  - *$(s,s') \in T$ and*
  - *$(q, L(s), q') \in \delta$;*
- *$F_\mathcal{P} = S \times F$ is a set of accepting states;*
- *$W_\mathcal{P} : \delta_\mathcal{P} \to \mathbb{R}_{>0}$, is a weight function, where*

$$W_\mathcal{P}\big(((s,q),(s',q'))\big) = W\big((s,s')\big), \text{ for all } \big((s,q),(s',q')\big) \in \delta_\mathcal{P};$$

- *$Q_\mathcal{P}^{\pi_{\text{opt}}} \subseteq Q_\mathcal{P}$ is a set of states in which the proposition π_{opt} holds true. In other words, $(s,q) \in Q_\mathcal{P}^{\pi_{\text{opt}}}$ if and only if $\pi_{\text{opt}} \in L(s)$.*

The product automaton as defined above can be seen as a Büchi automaton with a trivial input alphabet enhanced with transition weights and the set $Q_\mathcal{P}^{\pi_{\text{opt}}}$. Since the alphabet is trivial, we omit it.

Similarly as in the transition system, we associate with each run $\rho_\mathcal{P} = p_0 p_1 p_2 \ldots$ of $\mathcal{P}$, a sequence of time instances $\mathbb{T}_\mathcal{P} := \mathsf{t}_0 \mathsf{t}_1 \mathsf{t}_2 \ldots$, where $\mathsf{t}_0 = 0$, and t_i denotes the time at which the ith state of the run is reached (*i.e.,*

$\mathtt{t}_{i+1} = \mathtt{t}_i + W_{\mathcal{P}}\big((p_i, p_{i+1})\big)$, for all $i \geq 0$). From this time sequence we can extract a sequence $\mathbb{T}_{\mathcal{P}}^{\pi_{\text{opt}}}$, containing time instances $\mathtt{t}_i$, where $p_i \in Q_{\mathcal{P}}^{\pi_{\text{opt}}}$ (*i.e.,* $\mathbb{T}_{\mathcal{P}}^{\pi_{\text{opt}}}$ is a sequence of satisfying instances of the optimizing proposition π_{opt} in $\mathcal{T}$). The cost of a run $\rho_{\mathcal{P}}$ of the product automaton $\mathcal{P}$ (which corresponds to cost function (5.2) on transition system $\mathcal{T}$) is

$$Cost_{\mathcal{P}}(\rho_{\mathcal{P}}) = \limsup_{i \to +\infty} \left(\mathbb{T}_{\mathcal{P}}^{\pi_{\text{opt}}}(i+1) - \mathbb{T}_{\mathcal{P}}^{\pi_{\text{opt}}}(i)\right). \tag{5.3}$$

The following lemmas helps us to characterize Problem 5.2.2 as a graph problem for the product automaton graph.

Lemma 5.2.5. *At least one of the accepting runs $\rho_{\mathcal{P}}$ of $\mathcal{P}$ that minimizes cost function $Cost_{\mathcal{P}}(\rho_{\mathcal{P}})$ in (5.3) is in prefix-suffix structure.*

Proof. Let $\rho_{\mathcal{P}}$ be an accepting run of $\mathcal{P}$ that minimizes the cost $Cost_{\mathcal{P}}(\rho_{\mathcal{P}})$ and is not in prefix-suffix structure. We will prove the existence of an accepting run $\varrho_{\mathcal{P}}$ in prefix-suffix structure, such that $Cost_{\mathcal{P}}(\varrho_{\mathcal{P}}) \leq Cost_{\mathcal{P}}(\rho_{\mathcal{P}})$. The idea behind the proof is that an accepting state must occur infinitely many times on $\rho_{\mathcal{P}}$. We then show that we can extract a finite sequence of states starting and ending at this accepting state which can be repeated to form a periodic suffix whose cost is no larger than $Cost_{\mathcal{P}}(\rho_{\mathcal{P}})$.

To begin, there exists a state $f \in F_{\mathcal{P}}$ occurring on $\rho_{\mathcal{P}}$ infinitely many times. Run $\rho_{\mathcal{P}}$ consists of a prefix $\rho_{\mathcal{P}}^{\text{pref}}$ ending at the state f followed by an infinite, non-periodic suffix $\rho_{\mathcal{P}}^{\text{suf}}$ starting at the state f reached by the prefix. The suffix $\rho_{\mathcal{P}}^{\text{suf}}$ can be viewed as infinite number of finite sequences of states of form $fp_1p_2 \dots p_n f$, where $p_i \neq f$ for any $i \in \{1, \dots, n\}$. Let *Sufs* denote the set of all finite sequences of states of the mentioned form occurring on the suffix $\rho_{\mathcal{P}}^{\text{suf}}$.

Note, that each sequence in the set *Sufs* has to contain at least one occurrence of a state from $Q_{\mathcal{P}}^{\pi_{\text{opt}}}$. To see this, assume by way of contradiction that there is a sequence $fp_1p_2 \dots p_n f$ that does not contain any state from $Q_{\mathcal{P}}^{\pi_{\text{opt}}}$. The prefix $\rho_{\mathcal{P}}^{\text{pref}}$ followed by infinitely many repetitions of this sequence is indeed an accepting run of $\mathcal{P}$. However, if projected onto run of $\mathcal{T}$, formula $\mathsf{G}\,\mathsf{F}\,\pi_{\text{opt}}$ and thus also formula ϕ is violated, contradicting the model checking principle.

Similarly as for infinite traces, we associate with each finite sequence of states of length n a sequence of time instances $\mathbb{T}_{\mathcal{P}} := \mathtt{t}_0\mathtt{t}_1\mathtt{t}_2 \dots \mathtt{t}_n$, where $\mathtt{t}_0 = 0$, and $\mathtt{t}_i$ denotes the time at which the ith vertex in the run is reached ($\mathtt{t}_{i+1} = \mathtt{t}_i + W_{\mathcal{P}}\big((p_i, p_{i+1})\big)$, for all $0 \leq i < n$). From this time sequence we can extract a sequence $\mathbb{T}_{\mathcal{P}}^{\pi_{\text{opt}}}$, containing time instances $\mathtt{t}_i$, where $p_i \in Q_{\mathcal{P}}^{\pi_{\text{opt}}}$.

For each finite sequence $\rho_{\text{fin}} \in \mathit{Sufs}$ with n states and k occurrences of a state from $Q_{\mathcal{P}}^{\pi_{\text{opt}}}$ we define the following three costs

- $c^{f \rightsquigarrow}(\rho_{\text{fin}}) = \mathbb{T}_{\mathcal{P}}^{\pi_{\text{opt}}}(0) - \mathbb{T}_{\mathcal{P}}(0)$

- $c(\rho_{\text{fin}}) = \max_{i\in\{0,\dots,k-1\}} \left(\mathbb{T}_{\mathcal{P}}^{\pi_{\text{opt}}}(i+1) - \mathbb{T}_{\mathcal{P}}^{\pi_{\text{opt}}}(i)\right)$
- $c^{\rightsquigarrow f}(\rho_{\text{fin}}) = \mathbb{T}_{\mathcal{P}}(n) - \mathbb{T}_{\mathcal{P}}^{\pi_{\text{opt}}}(k)$.

Further, we define an equivalence relation $\sim$ over *Sufs* as follows. Let $\rho_{\text{fin}}, \rho'_{\text{fin}} \in$ *Sufs*. $\rho_{\text{fin}} \sim \rho'_{\text{fin}}$ if and only if

- $c^{f\rightsquigarrow}(\rho_{\text{fin}}) = c^{f\rightsquigarrow}(\rho'_{\text{fin}})$,
- $c(\rho_{\text{fin}}) = c(\rho'_{\text{fin}})$, and
- $c^{\rightsquigarrow f}(\rho_{\text{fin}}) = c^{\rightsquigarrow f}(\rho'_{\text{fin}})$.

Costs $c^{f\rightsquigarrow}$, c, and $c^{\rightsquigarrow f}$ can be extended to $c_{\sim}^{f\rightsquigarrow}$, $c_{\sim}$, and $c_{\sim}^{\rightsquigarrow f}$ in a natural way. For example, we define $c_{\sim}^{f\rightsquigarrow}([\rho_{\text{fin}}]_{\sim}) = c^{f\rightsquigarrow}(\rho_{\text{fin}})$, where $\rho_{\text{fin}} \in [\rho_{\text{fin}}]_{\sim}$. The other two costs are defined analogously.

Let us extract a set $Sufs^{\text{inf}}/_{\sim}$ from the set of equivalence classes $Sufs/_{\sim}$ such that each class in $Sufs^{\text{inf}}/_{\sim}$ is infinite or contains a finite sequence of states that is repeated in $\rho_{\mathcal{P}}$ infinitely many times. As a consequence, for each class $[\rho_{\text{fin}}]_{\sim}$ in $Sufs^{\text{inf}}/_{\sim}$, it holds that $c_{\sim}([\rho_{\text{fin}}]_{\sim}) \leq Cost_{\mathcal{P}}(\rho_{\mathcal{P}})$. The set $Sufs/_{\sim}$ is finite, because there is only a finite number of different values of costs. Furthermore, accepting run $\rho_{\mathcal{P}}$ is infinite and thus $Sufs^{\text{inf}}/_{\sim}$ is nonempty.

Let $[\varrho_{\text{fin}}]_{\sim} \in Sufs^{\text{inf}}/_{\sim}$ now be a class such that $c_{\sim}^{f\rightsquigarrow}([\varrho_{\text{fin}}]_{\sim})$ is minimal among the classes from $Sufs^{\text{inf}}/_{\sim}$. Each time a finite sequence in $[\varrho_{\text{fin}}]_{\sim}$ appears in $\rho_{\mathcal{P}}$, it is followed by another finite sequence of states. Consider, that infinitely many times the "following" sequence comes from a class $([\rho_{\text{fin}}]_{\sim}) \in Sufs^{\text{inf}}/_{\sim}$. Then, we must have $c^{\rightsquigarrow f}([\varrho_{\text{fin}}]_{\sim}) + c^{f\rightsquigarrow}([\rho_{\text{fin}}]_{\sim}) \leq Cost_{\mathcal{P}}(\rho_{\mathcal{P}})$. But, $c^{f\rightsquigarrow}([\rho_{\text{fin}}]_{\sim}) \geq c^{f\rightsquigarrow}([\varrho_{\text{fin}}]_{\sim})$, and thus $c^{\rightsquigarrow f}([\varrho_{\text{fin}}]_{\sim}) + c^{f\rightsquigarrow}([\varrho_{\text{fin}}]_{\sim}) \leq Cost_{\mathcal{P}}(\rho_{\mathcal{P}})$.

Thus we can build the run $\varrho_{\mathcal{P}}$ as the prefix $\rho_{\mathcal{P}}^{\text{pref}}$ followed by a periodic suffix $\varrho_{\mathcal{P}}^{\text{suf}}$, which is obtained by infinitely many repetitions of an arbitrary sequence $\varrho_{\text{fin}} \in [\varrho_{\text{fin}}]_{\sim}$. Run $\varrho_{\mathcal{P}}$ is in prefix-suffix structure and for its suffix $\varrho_{\mathcal{P}}^{\text{suf}}$ it also holds $Cost_{\mathcal{P}}(\varrho_{\mathcal{P}}) = \max_{i\in\mathbb{N}} \left(\mathbb{T}_{\mathcal{P}}^{\pi_{\text{opt}}}(i+1) - \mathbb{T}_{\mathcal{P}}^{\pi_{\text{opt}}}(i+1)\right) = \max\left(c(\varrho_{\text{fin}}), c^{f\rightsquigarrow}(\varrho_{\text{fin}}) + c^{\rightsquigarrow f}(\varrho_{\text{fin}})\right) \leq Cost_{\mathcal{P}}(\rho_{\mathcal{P}})$. □

Definition 5.2.6 (Suffix Cost)**.** *The cost of the periodic suffix $p_0p_1\dots p_np_0p_1\dots$ of a run $\rho_{\mathcal{P}}$ in the prefix-suffix structure is defined as follows.*

Let $\mathsf{t}_{0,0}, \mathsf{t}_{0,1}, \dots, \mathsf{t}_{0,n}, \mathsf{t}_{1,0}, \mathsf{t}_{1,1}\dots$ be the sequence of times at which the vertices of the suffix are reached on run $\rho_{\mathcal{P}}$. Extract the sub-sequence $\mathbb{T}_{\mathcal{P}}^{\text{suf}}$ of times $\mathsf{t}_{i,j}$, where $p_j \in Q_{\mathcal{P}}^{\pi_{\text{opt}}}$ (i.e., the satisfying instances of proposition π_{opt} in transition system $\mathcal{T}$). Then, the cost of the suffix is

$$Cost_{\mathcal{P}}^{\text{suf}}(\rho_{\mathcal{P}}) = \max_{i\in\mathbb{N}} (\mathbb{T}_{\mathcal{P}}^{\text{suf}}(i+1) - \mathbb{T}_{\mathcal{P}}^{\text{suf}}(i)). \tag{5.4}$$

From the definition of the product automaton cost $Cost_{\mathcal{P}}$ and the suffix cost $Cost^{\text{suf}}_{\mathcal{P}}$ we obtain the following result.

Proposition 5.2.7 (Cost of a Run)**.** *Given a run $\rho_{\mathcal{P}}$ in prefix-suffix structure and its suffix $p_0p_1 \dots p_np_0p_1 \dots$, the value of the cost function $Cost_{\mathcal{P}}(\rho_{\mathcal{P}})$ is equal to the cost of the suffix $Cost^{\text{suf}}_{\mathcal{P}}(\rho_{\mathcal{P}})$.*

Our aim is to synthesize a trace r of $\mathcal{T}$ minimizing the cost function $Cost(r)$ in (5.2) and ensuring that the word produced by this run will be accepted by the automaton $\mathcal{B}_\phi$. This goal now translates to generating a run $\rho_{\mathcal{P}}$ of $\mathcal{P}$, such that the run satisfies the Büchi condition $F_{\mathcal{P}}$ and minimizes cost function $Cost_{\mathcal{P}}(\rho_{\mathcal{P}})$ in (5.3). Furthermore, to find a satisfying run $\rho_{\mathcal{P}}$ that minimizes $Cost_{\mathcal{P}}(\rho_{\mathcal{P}})$, it is enough to consider runs in prefix-suffix structure (see Lemma 5.2.5). From Lemma 5.2.7 it follows that the whole problem reduces to finding a periodic suffix $\rho^{\text{suf}}_{\mathcal{P}} = fp_1p_2 \dots p_nfp_1 \dots$ in $\mathcal{P}$, such that:

(i) f is reachable from an initial state $q^{\text{init}}_{\mathcal{P}}$,
(ii) $f \in F_{\mathcal{P}}$ (*i.e.,* f is an accepting state), and
(iii) the cost $Cost^{\text{suf}}_{\mathcal{P}}(\rho_{\mathcal{P}})$ of the suffix $\rho^{\text{suf}}_{\mathcal{P}}$ is minimal among all the suffixes satisfying (i) and (ii).

Finally, we can find the shortest prefix in $\mathcal{P}$ that starts at an initial state $q^{\text{init}}_{\mathcal{P}}$ and ends at the state f in the suffix $\rho^{\text{suf}}_{\mathcal{P}}$. By concatenating the prefix and suffix, we obtain an optimal run in $\mathcal{P}$. By projecting the optimal run onto the states of $\mathcal{T}$, we obtain a solution to our stated optimal surveillance problem (Problem 5.2.2).

Graph Algorithm for Optimal Suffix Synthesis

We now focus on finding an optimal suffix in the product automaton. We cast this problem as a path optimization on a product automaton graph. To do this, let us define some terminology.

The weighted product automaton can be viewed as a weighted directed graph $G = (V, E, W)$ that consists of a vertex set V given by the set of states $Q_{\mathcal{P}}$, an edge set $E \subseteq V \times V$, given by the transition relation $\delta_{\mathcal{P}}$ and a weight function $W : E \to \mathbb{R}_{>0}$ given by the weight function $W_{\mathcal{P}}$. The usual graph structures, such as path, simple path and cycle are defined in the expected way, analogously as for the non-weighted version of a directed graph (see Section 3.4).

Given a vertex set $O \subseteq V$, consider a cycle $\rho_{\text{cycle}} = v_1 \dots v_k$ containing at least one vertex in O. Let $(i_1, i_2, \dots, i_m)$ be the ordered set of indexes, such that the corresponding vertexes are elements of O (*i.e.,* indexes with order $i_1 < i_2 < \dots < i_m$, such that $v_j \in O$ if and only if $j \in \{i_1, i_2, \dots, i_m\}$). Then, the

O-bottleneck length is

$$\max_{l\in\{1,\ldots,m\}} \sum_{j=i_l}^{i_{l+1}-1} W\big((v_j, v_{j+1})\big),$$

where $i_{m+1} = i_1$. In words, we O-bottleneck distance is defined as follows.

Definition 5.2.8 (O-Bottleneck Length)**.** *Given a weighted directed graph $G = (V, E, W)$, and a vertex set $O \subseteq V$, the O-bottleneck length of a cycle in G is the maximum distance between successive appearances of an element of O on the cycle.*[1]

The *bottleneck length* of a cycle is defined as the maximum length edge on the cycle [KV07]. In contrast, the O-bottleneck length measures distances between vertices in O. With the terminology in place, our goal is to solve the *shortest O-bottleneck problem*:

Problem Statement 5.2.9 (Shortest O-bottleneck)**.** *Given*

- *a weighted graph $G = (V, E, W)$; and*
- *two vertex sets $F, O \subseteq V$,*

find a cycle in G containing at least one vertex in F, with minimum O-bottleneck length.

Our solution to Problem 5.2.9, called the min_bottleneck_cycle algorithm, is shown in Algorithm 9. It utilizes Dijkstra's algorithm (see, *e.g.,* [KV07]) for computing shortest paths between pairs of vertices (called shortest_path), and a slight variation of Dijkstra's algorithm for computing shortest bottleneck paths between pairs of vertices (called shortest_bot_path).

Procedure shortest_path takes as inputs a graph $G = (V, E, W)$, a set of source vertices $A \subseteq V$, and a set of destination vertices $B \subseteq V$. It outputs a distance matrix $Dist \in \mathbb{R}^{|A|\times|B|}$, where the entry $Dist(i, j)$ gives the shortest-path distance from A_i to B_j. It also outputs a predecessor matrix $Pred \in V^{|A|\times|V|}$, where $Pred(i, j)$ is the predecessor of j on a shortest path from A_i to B_j. For a vertex $v \in V$, the shortest path from v to v is defined as the shortest cycle containing v. If there does not exist a path between vertices, then the distance is $+\infty$. Procedure shortest_bot_path has the same inputs as shortest_path, but it outputs paths which minimize the maximum edge length, rather than the sum of edge lengths.

1. If the cycle does not contain an element of O, then its O-bottleneck length is defined as $+\infty$.

Algorithm 9 min_bottleneck_cycle(G, O, F)

Input: A weighted directed graph $G = (V, E, W)$, and vertex subsets O and F
Output: A cycle in G which contains at least one vertex in F and minimizes the O-bottleneck length.

1: Compute shortest paths between vertices in O:
$(Dist, Pred) = \mathsf{shortest\_path}(G, O, O)$;
2: Define a weighted graph $G_{\pi_{\text{opt}}}$ with vertices O and adjacency matrix given by $Dist$.
3: Compute shortest O-bottleneck paths between vertices in O:
$(Dist_{\text{bot}}, Pred_{\text{bot}}) = \mathsf{shortest\_bot\_path}(G_{\pi_{\text{opt}}}, O, O)$;
4: Compute shortest paths from each vertex in F to each vertex in O, and from each vertex in O to each vertex in F:
$(Dist_{F\to O}, Pred_{F\to O}) = \mathsf{shortest\_path}(G, F, O)$;
$(Dist_{O\to F}, Pred_{O\to F}) = \mathsf{shortest\_path}(G, O, F)$.
5: Set $Dist_{F\to O}(i, j) = 0$ and $Dist_{O\to F}(j, i) = 0$ for all i, j such that $F_i = O_j$.
6: For each triple $(f, o_1, o_2) \in F \times O \times O$, set

$$Cost(f, o_1, o_2) = \begin{cases} Dist_{F\to O}(f, o_1) + Dist_{O\to F}(o_2, f) & \text{if } f \neq o_1 = o_2 \\ \max\left(Dist_{F\to O}(f, o_1) + Dist_{O\to F}(o_2, f), Dist_{\text{bot}}(o_1, o_2)\right), & \text{otherwise} \end{cases}$$

7: Find the triple (f^*, o_1^*, o_2^*) that minimizes $Cost(f, o_1, o_2)$.
8: If minimum cost is $+\infty$, then output "no cycle exists." Else, output cycle by extracting the path from f^* to o_1^* using $Pred_{F\to O}$, the path from o_1^* to o_2^* using $Pred_{\text{bot}}$ and $Pred$, and the path from o_2^* to f^* using $Pred_{O\to F}$.

In the algorithm, one has to take special care that cycle lengths are computed properly when $f = o_1$, $o_1 = o_2$, or $f = o_2$ on line 6. This is done by setting some entries of $D_{F\to S}$ and $D_{S\to F}$ to zero in step 4, and by defining the cost differently when $f \neq s_1 = s_2$ in step 5.

Example 5.2.10 (Shortest O-bottleneck Cycle)**.** *Figure 5.3 (left) shows an example input to the algorithm. The graph contains 12 vertices, with one vertex (diamond) in F, and four vertices (square) in S. Figure 5.3 (right) shows the optimal solution as produced by the algorithm. The bottleneck occurs between the square vertices immediately before and after the diamond vertex.*

The optimal_run *Algorithm*

We are now ready to combine the results from the previous section to present a solution to Problem 5.2.2. The solution, the optimal_run algorithm, is summarized in Algorithm 10.

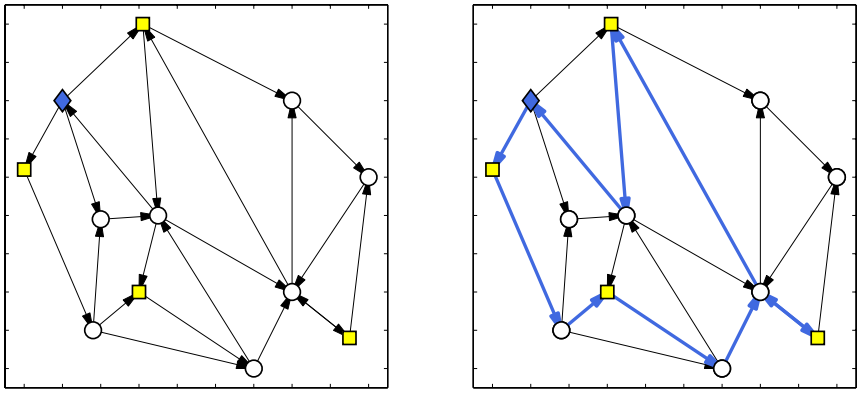


Figure 5.3: The left figure shows an example of an input to the min_bottleneck_cycle algorithm. In the directed graph, the edge weights are given by the Euclidean distance. The set F is a singleton given by the diamond. The vertices in S are drawn as yellow squares. The right figure shows a cycle with minimum O-bottleneck length optimal cycle using thick (blue) edges.

Algorithm 10 optimal_run$(\mathcal{T}, \phi)$

Input: A weighted deterministic TS $\mathcal{T}$, and an LTL specification ϕ in form (5.1).
Output: A run in $\mathcal{T}$ which satisfies ϕ and minimizes (5.2).

1: Convert ϕ to a Büchi automaton $\mathcal{B}_\phi$.
2: Compute the product automaton $\mathcal{P} = \mathcal{T} \otimes \mathcal{B}_\phi$ according to Def. 5.2.4.
3: Compute the cycle min_bottleneck_cycle$(G, Q_{\mathcal{P}}^{\pi_{\text{opt}}}, F_{\mathcal{P}})$, where $G = (Q_{\mathcal{P}}, \delta_{\mathcal{P}}, W_{\mathcal{P}})$.
4: Compute a shortest path from $q_{\mathcal{P}}^{\text{init}}$ to the cycle.
5: Project the complete run (path and cycle) onto a trace of $\mathcal{T}$

Correctness and Complexity

Lemma 5.2.11 (min_bottleneck_cycle Correctness)**.** *The* min_bottleneck_cycle *algorithm solves the shortest O-bottleneck problem (Problem 5.2.9).*

Proof. Every valid cycle must contain at least one element from F and at least one element from O. Let $\rho_{\text{cycle}} = v_1 v_2 \dots v_k v_1$, be a valid cycle, and without loss of generality let $v_1 \in F$. From this cycle we can extract the triple $(v_1, v_a, v_b) \in F \times O \times O$, where $v_a, v_b \in O$, and $v_i \notin O$ for all $i < a$ and for all $i > b$. (Note that, $a = b = 1$ is possible.)

Consider a cycle ρ_{cycle} with corresponding triple (f, o_1, o_2), and let $Len(\rho_{\text{cycle}})$ denote its O-bottleneck length. It is straightforward to verify, using the definition of O-bottleneck length, that $Len(\rho_{\text{cycle}}) \geq Cost(f, o_1, o_2)$.

The cycle computed in step 5 (as given by the four predecessor matrices) takes the shortest path from f to o_1, the shortest O-bottleneck path from o_1 to

o_2, and the shortest path from o_2 to f. However, the shortest path from f to o_1 (and from o_2 to f) may contain other vertices from O. Thus, the o-bottleneck length of this cycle, denoted $L(f, o_1, o_2)$, satisfies

$$Len(f, o_1, o_2) \leq Cost(f, o_1, o_2) \leq Len(\rho_{\text{cycle}}), \tag{5.5}$$

implying that $Cost(f, o_1, o_2)$ upper bounds the length of the computed cycle. However, if we take ρ_{cycle} to be a cycle with minimum length, then necessarily $Len(\rho_{\text{cycle}}) \leq Len(f, o_1, o_2)$. Hence, equation (5.5) implies that for an optimal cycle, $Len(f, o_1, o_2) = Cost(f, o_1, o_2) = Len(\rho_{\text{cycle}})$. Thus, by minimizing the cost function in step 5 we compute the minimum length cycle. □

Finally, we characterize the complexity of the min_bottleneck_cycle algorithm. Let n, m, n_O, and n_F, be the number of vertices (edges) in the sets V, E, O, and F, respectively. Dijkstra's algorithm can be implemented to compute shortest paths from a source vertex $v \in V$, to all other vertices in V in $\mathcal{O}(n \log n + m)$ run time. Thus, for sparse graphs (which includes many transition systems), the run time is $\mathcal{O}(n \log n)$.

Step 1 of the algorithm requires n_O calls to Dijkstra's algorithm, and has run time $\mathcal{O}(n_O(n \log(n) + m))$. Step 3 requires n_O calls to Dijkstra's algorithm on a smaller graph $G_O = (O, E_O, W_O)$, and has run time $\mathcal{O}(n_O(n_O \log(n_O) + |E_O|))$. Step 4 has run time $\mathcal{O}(n_F(n \log(n) + m))$. Finally, step 5 and 6 require searching over all $n_F \cdot n_O^2$ possibilities, and have run time $\mathcal{O}(n_F n_O^2)$. Since $|E_O| \leq n_O^2$, the run time in general is given by $\mathcal{O}\big((n_O + n_F)(n \log n + m + n_O^2)\big)$.

To sum up, the run time of the min_bottleneck_cycle algorithm is

$$\mathcal{O}\big((n_O + n_F)(n \log n + m + n_O^2)\big).$$

Thus, in the worst-case, the run time is $\mathcal{O}(n^3)$. For sparse graphs with $n_O, n_F \ll n$, the run time is $\mathcal{O}\big((n_O + n_F) n \log n\big)$.

Theorem 5.2.12 (Correctness of optimal_run)**.** *The optimal_run algorithm solves Problem 5.2.2.*

Proof. The correctness of the proposed optimal_run algorithm follows directly from Lemma 6.2.6, and Theorem 5.2.11. □

The worst-case computational complexity of the optimal_run algorithm can be characterized as follows. Any LTL formula ϕ can be translated into a Büchi automaton in time $2^{\mathcal{O}(|\phi|)}$ computation time [BK08]. The worst-case size of the Büchi automaton, *i.e.,* the number of states, is also $2^{\mathcal{O}(|\phi|)}$. The size of the product obtained in step 2 of the optimal_run algorithm (Algorithm 9) is

therefore in $\mathcal{O}(|T| \cdot 2^{\mathcal{O}(|\phi|)})$, where $|T|$ is the number of states in the transition system. Then, the worst-case complexity of the optimal_run algorithm is $\mathcal{O}(|T|^3 \cdot 2^{\mathcal{O}(|\phi|)})$.

Thus, the worst-case complexity is quite restrictive, being exponential in the size of the LTL formula. However, many rich robot behaviors can be described using relatively small LTL formulas. In addition, the time required to compute the Büchi automaton, and the size of the Büchi automaton, are frequently much smaller than the worst-case bound. In the following section we show that the proposed approach can be used to generate robot motion plans that satisfy rich requirements in complex environments.

5.2.3 Alternative Problem Formulations

First, the optimal_run algorithm optimizes the cost of the repeated suffix. For the prefix, we simply find the shortest path from an initial state to the suffix. However, the cost of the prefix is not optimized. This is due to the fact that the cost function *Cost* in (5.2) was chosen with persistent monitoring tasks in mind, where the long-term behavior is of interest. However, in some applications, the transient behavior may be of interest. In this case we can define an alternative cost function *Cost*′ for the traces of $\mathcal{T}$:

$$Cost'(r) = \sup_{i \in \mathbb{N}} \left(\mathbb{T}_{\pi_{\text{opt}}}(i+1) - \mathbb{T}_{\pi_{\text{opt}}}(i) \right). \tag{5.6}$$

Then, we can consider two alternative problems: (i) Find a trace r minimizing the cost $Cost'(r)$; or (ii) Find a run r that minimizes the cost $Cost'(r)$ among all the runs minimizing the cost $Cost(r)$. Both problems can be solved by slightly modifying the min_bottleneck_cycle algorithm.

First, we can slightly alter the proof of Lemma 6.2.6 to show that there is a run in prefix-suffix form that minimizes *Cost*′. By appropriately defining the cost of the prefix, we can also show that the cost *Cost*′ is equal to the maximum of the prefix cost and the suffix cost. Then, to solve problems (i) and (ii) we add a step to the min_bottleneck_cycle algorithm in which we compute the shortest bottleneck path from each initial state $v_0 \in V$ to each state $s \in S$. We record the cost of the path from v_0 to s as $Cost_p(v_0, s)$. For problem (i) we alter step 6 to find the tuple $(v_0^*, f^*, v_1^*, v_2^*)$ that minimizes $\max\{Cost_p(v_0, v_1), Cost(f, v_1, v_2)\}$. For problem (ii) we alter step 6 to find the tuple $(v_0^*, f^*, v_1^*, v_2^*)$ that minimizes $Cost_p(v_0, v_1)$ among the tuples that minimize $Cost(f, v_1, v_2)$. Finally, we remove step 4 from the optimal_run algorithm.

Second, the suggested problem formulation and solution can be easily extended to address the following problem defined for systems with degradation

from Chapter 4.

Consider a deterministic TSD $\mathcal{T} = (S, S_{\text{init}}, Act, T, AP, L, D)$ and an LTL formula ϕ . We define the semantics of LTL over traces of TSD as follows. Given a trace $r = s_0\alpha_0 s_1\alpha_1$, then $r \models \phi$ if and only if the word $L(s_0)L(s_1)L(s_2)\ldots \models \phi$.

Each trace of $\mathcal{T}$ starts with the initial level of degradation $d = 0$. Assume that there is at least one trace satisfying an LTL formula ϕ in the form of Eq. (5.1). Associated with a trace $r = s_0\alpha_0 s_1\alpha_1 \ldots$ there is a sequence of degradation levels $\mathbb{D}_r(0)\mathbb{D}_r(1)\mathbb{D}_r(2)\ldots$, where $\mathbb{D}_r(0) = 0$, and $\mathbb{D}_r(i)$ denotes the level of degradation up to the state s_i on the trace r. From this sequence we can extract all instances at which the proposition π_{opt} is satisfied. We let $\mathbb{D}_{\pi_{\text{opt}}}$ denote the sequence of satisfying instances of the proposition π_{opt}.

Our goal is to synthesize an infinite trace r (*i.e.,* a robot path) satisfying LTL formula (5.1), and minimizing the cost function

$$Cost_{\mathbb{D}}(r) = \limsup_{i \to +\infty} \left(\frac{\mathbb{D}_{\pi_{\text{opt}}}(i+1)}{\mathbb{D}_{\pi_{\text{opt}}}(i)} \right), \tag{5.7}$$

where $\mathbb{D}_{\pi_{\text{opt}}}(i)$ is the ith satisfying time instance of proposition π_{opt}.

Now, consider an LTL formula ϕ in the form of Eq. 5.1. The problem we would like to solve is as follows.

Problem Statement 5.2.13. *Given*

- *a TSD $\mathcal{T}$;*
- *an LTL over its set of atomic propositions AP in form* (5.1)*; and*
- *an optimizing proposition $\pi_{\text{opt}} \in AP$,*

find a trace r of $\mathcal{T}$ minimizing the cost $Cost_{\mathbb{D}}(r)$ from Eq. 5.7.

Note that the well-posedness of the Problem is guaranteed by the form of the LTL formula as in Eq. 5.8. Its solution is then just a straightforward twist in the solution we presented; the cost of a finite path defined as a sum of weights will now be defined as a product of degradations along the path.

5.2.4 Data Gathering Robot Example and Experiments

In this section, we present an implementation of the optimal_run algorithm on a physical testbed. We focus on a data gathering mission in which a robot must repeatedly gather data at interesting locations, and then upload it at designated sites. We also present a case-study which outlines several different robot missions, and how they can be expressed in LTL. The purpose of this section is to i) demonstrate the utility of the proposed approach in generating complex motion

plans; ii) illustrate the expressivity of LTL and the class of optimizations considered in this paper; ii) highlight the subtleties and challenges that arise when expressing a desired behavior in LTL; and, iv) provide numerical data on the complexity and computation time of our proposed approach.

The Road-Network Testbed

We have implemented the optimal_run algorithm on the road network shown in Figure 5.1. This network is a collection of roads, intersections, and parking lots (which serve as data gather and upload locations), connected by a simple set of rules (*e.g.,* a road connects two (not necessarily different) intersections, the parking lots can only be located on the side of a road). The city is easily reconfigurable through re-taping. The robot used is a Khepera III miniature car. Figure 5.4 shows two snapshots of the robot moving the road network.

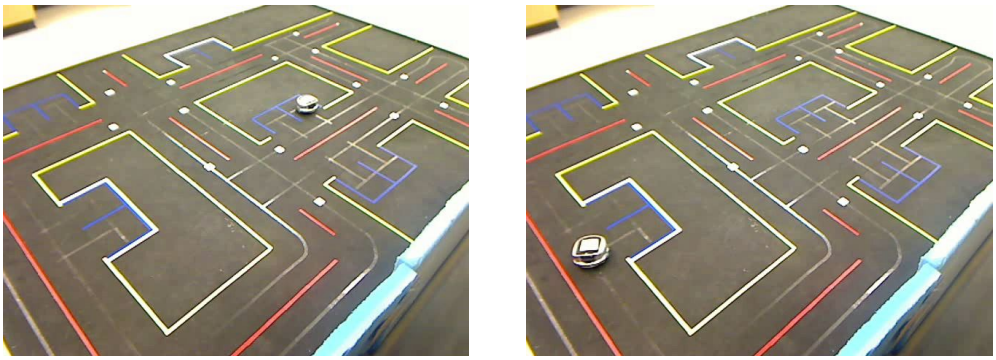


Figure 5.4: Two snapshots of the robot in road network. In the left figure the robot is gathering data at g_2, and in the right figure it is about to upload data at u_2.

The car can sense when entering an intersection from a road, when entering a road from an intersection, when passing in front of a parking lot, when it is correctly parked in a parking space, and when an obstacle is dangerously close. The car is programmed with motion and communication primitives allowing it to safely drive on a road, turn in an intersection, and park. The car can communicate through Wi-Fi with a desktop computer, which is used as an interface to the user (*i.e.,* to enter the specification) and to perform all the computation necessary to generate the control strategy. Once computed, this is sent to the car, which executes the task autonomously by interacting with the environment.

An Experimental Case Study on Data Gathering Missions

In our experiments, we have considered data gathering missions of the following form. Parking lots u_1 and u_2 in Figure 5.2 are data upload locations (light (yellow) shaded regions in Figure 5.5) and parking lots g_1, g_2, and g_3 are data gather locations (dark (green) shaded regions in Figure 5.5). The optimizing proposition π_{opt} in LTL formula (5.1) is

$$\pi_{\text{opt}} := \mathsf{u}_1 \vee \mathsf{u}_2, \tag{5.8}$$

i.e., we want to minimize the time between data uploads. Assuming infinite runs of the robot in the environment, we are able to describe the motion requirements as LTL formulas, where atomic propositions are simply names of the parking lots.

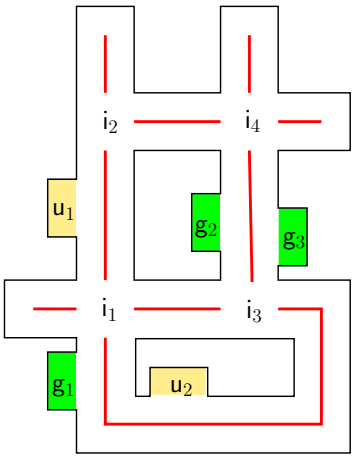


Figure 5.5: Schematic illustration of the road network. For each road, the median is shown as a red line. The robot must drive on the right-hand side of the road (*i.e.,* the right-hand side of the median). Intersections are labelled i_1 through i_4. Data gather locations, labeled g_1, g_2, and g_3, are shaded green (dark). Data upload locations, labeled u_1 and u_2, are shaded yellow (light).

We consider seven different data gathering cases. Each case describes a data gathering mission, and the cases are roughly ordered in increasing complexity. For each case we have computed the optimal run according the to optimal_run algorithm, and have implemented the run on our testbed. In Table 5.1 we summarize the key statistics for each case. The summary data consists of i) the maximum distance between uploads on the optimal path, ii) the maximum time between uploads observed in the robot experiment, iii) the number of states in the Büchi automaton, iv) the number of states in the product automaton,

case	length (m)	time (min) travel	# of states BA	# of states product	time (sec) LTL to BA	time (sec) computation
A	6.23	2.5	3	78	~ 1	~ 1
B	6.23	2.5	7	182	~ 1	~ 1
C	9.13	3.6	11	286	~ 1	~ 1
D	9.13	3.6	17	442	~ 1	~ 1
E	9.13	3.6	49	1274	~ 1	~ 8
F	10.48	4.1	34	884	~ 1	~ 2
G	9.50	3.7	34	884	~ 1	~ 2

Table 5.1: A summary of the seven data gathering cases.

v) the time to translate the LTL formula into a Büchi automaton, and vi) the time to compute the optimal path in the product automaton. The computations were performed on a desktop computer with a 2.8GHz quad core processor and 8GB of RAM. We utilized the LTL2BA software by [GO01] to translate an LTL formula to a Büchi automaton.

Case A. To begin, let us consider the following mission. Repeatedly visit data gather locations (g_1, g_2 or g_3) to gather data and repeatedly visit upload locations (u_1 or u_2) to upload data. The objective is to minimize the time between visits to data upload locations, and therefore the optimizing proposition π_{opt} is given by the LTL formula in (5.8). We can specify this behavior as the following LTL formula:

$$\phi_A = \mathsf{G}\,\mathsf{F}\,(\mathsf{g}_1 \vee \mathsf{g}_2 \vee \mathsf{g}_3) \wedge \mathsf{G}\,\mathsf{F}\,\pi_{\text{opt}}.$$

Using the optimal_run algorithm, we compute the robot path shown in Figure A. This figure is interpreted as follows. The figure consists of a sequence of environment snapshots, read from left to right. Each snapshot shows a robot path as a line which starts and ends at a data upload location. The starting point of the robot path on the $(i+1)$th snapshot is given by endpoint of the path on the ith snapshot. The endpoint of the final snapshot connects with the starting point of the first snapshot. Thus, the infinite robot path is obtained by cycling through these snapshots.

The time to run the algorithm, and the value of the cost function are summarized in Table 5.1.

Case B. Looking at results of Case A, we see that the robot does not always gather new data before visiting an upload location (in Figure A the robot visits two upload locations in a row). To eliminate this behavior, we should

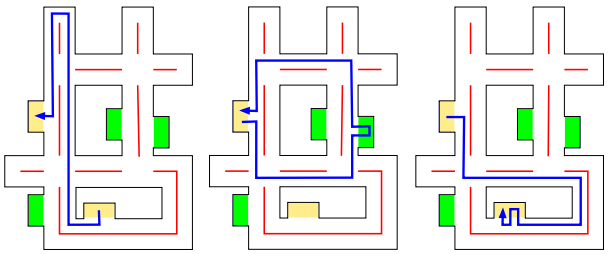


Figure A: The robot path (shown as lines with arrows) for Case A. Green (dark shaded) areas are data-gathering locations, and yellow (light shaded) areas are upload locations. The robot periodically follows the path which is composed of the illustrated fragments as seen from left to right.

specify that the robot can only visit a data upload location if it has just gathered data. This can be specified as follows:

$$\phi_B = \phi_A \wedge \mathsf{G}\left((\mathsf{u}_1 \vee \mathsf{u}_2) \Rightarrow \mathsf{X}\left((\neg \mathsf{u}_1 \wedge \neg \mathsf{u}_2)\,\mathsf{U}\,(\mathsf{g}_1 \vee \mathsf{g}_2 \vee \mathsf{g}_3)\right)\right)$$

The corresponding robot path is shown in Figure B.

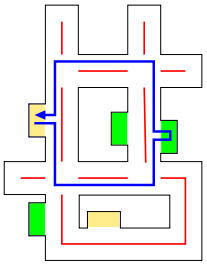


Figure B: The robot path for Case B. Note that the robot does not visit two upload locations without visiting a download locations in between. The value of the optimization function is the same as in Case A (6.23 meters).

Case C. In some situations the data gather locations g_1, g_2 and g_3 may contain different information, and thus it is beneficial to periodically visit each of them. To specify this, we can build on Case B and write the following formula:

$$\begin{aligned}\phi_C = {} & \mathsf{G}\,\mathsf{F}\,\mathsf{g}_1 \wedge \mathsf{G}\,\mathsf{F}\,\mathsf{g}_2 \wedge \mathsf{G}\,\mathsf{F}\,\mathsf{g}_3 \wedge \\ & \mathsf{G}\left((\mathsf{u}_1 \vee \mathsf{u}_2) \Rightarrow \mathsf{X}\left((\neg \mathsf{u}_1 \wedge \neg \mathsf{u}_2)\,\mathsf{U}\,(\mathsf{g}_1 \vee \mathsf{g}_2 \vee \mathsf{g}_3)\right)\right) \wedge \mathsf{G}\,\mathsf{F}\,\pi_{\text{opt}}\end{aligned}$$

Using the optimal_run algorithm, the computed path of the robot is shown in Figure C. The time to run the algorithm, and the value of the cost

function are summarized in Table 5.1. Note that this more restrictive formula results in a larger cost function value than in Cases A or B.

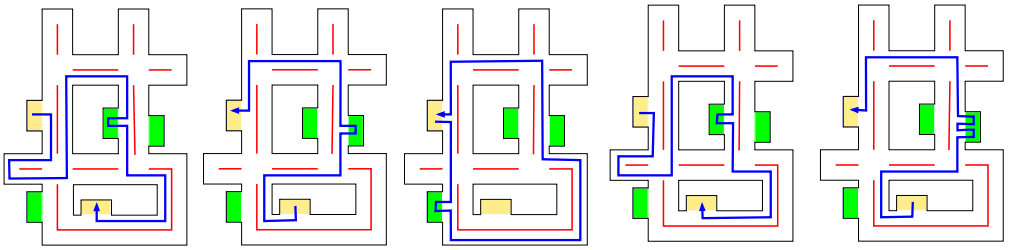


Figure C: The robot path for Case C. Note that the robot visits all three download locations and only visits an upload location if it has just gathered data.

Case D. Notice that in the last snapshot of Figure C, the robot visits data gather location g_3 twice in a row. Such behavior does not increase the value of the cost function, but may not be desirable in some circumstances. We can eliminate this behavior by specifying that the robot must visit an upload location after gathering data:

$$\phi_D = \phi_C \wedge \mathsf{G}\left((\mathsf{g}_1 \vee \mathsf{g}_2 \vee \mathsf{g}_3) \Rightarrow \mathsf{X}\left(\neg(\mathsf{g}_1 \vee \mathsf{g}_2 \vee \mathsf{g}_3)\,\mathsf{U}\,(\mathsf{u}_1 \vee \mathsf{u}_2)\right)\right)$$

The new path of the robot is shown in Figure D. Note from Table 5.1 that the maximum distance between uploads does not change from Case C to Case D.

Case E. Now, suppose that we would like to require an equal number of visits to each data gather location. We can observe that in Case D, some of the gather locations are visited more frequently than the others. To formalize this idea of equality, we can specify an order in which the data gather locations should be visited: g_3, g_1, and g_2, in this order. The syntax for specifying this order is somewhat complicated, and involves the nested until operators. The specification becomes

$$\begin{aligned}
\phi_E = {} & (\neg\mathsf{g}_1 \wedge \neg\mathsf{g}_2)\,\mathsf{U}\,\mathsf{g}_3 \wedge \\
& \mathsf{G}\,\big(\mathsf{g}_3 \Rightarrow \mathsf{X}((\neg\mathsf{g}_2 \wedge \neg\mathsf{g}_3)\,\mathsf{U} \\
& \quad (\mathsf{g}_1 \wedge \mathsf{X}\,((\neg\mathsf{g}_1 \wedge \neg\mathsf{g}_3)\,\mathsf{U} \\
& \quad (\mathsf{g}_2 \wedge \mathsf{X}\,((\neg\mathsf{g}_1 \wedge \neg\mathsf{g}_2)\,\mathsf{U}\,\mathsf{g}_3)))))\big) \wedge \\
& \mathsf{G}\,\big((\mathsf{u}_1 \vee \mathsf{u}_2) \Rightarrow \mathsf{X}\,((\neg\mathsf{u}_1 \wedge \neg\mathsf{u}_2)\,\mathsf{U}\,(\mathsf{g}_1 \vee \mathsf{g}_2 \vee \mathsf{g}_3))\big) \wedge \\
& \mathsf{G}\,\big((\mathsf{g}_1 \vee \mathsf{g}_2 \vee \mathsf{g}_3) \Rightarrow \mathsf{X}\,(\neg(\mathsf{g}_1 \vee \mathsf{g}_2 \vee \mathsf{g}_3)\,\mathsf{U}\,(\mathsf{u}_1 \vee \mathsf{u}_2))\big) \wedge \\
& \mathsf{G}\,\mathsf{F}\,\pi_{\text{opt}}
\end{aligned}$$

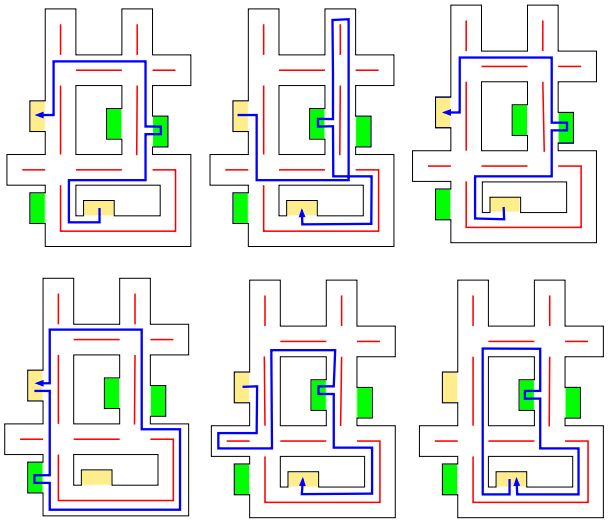


Figure D: The robot path for case D. One may observe that snapshots 1,2 and 6 are redundant and it would be sufficient to periodically repeat snapshots 3,4,5 to satisfy the formula. Such "aesthetic" changes do not improve the value of cost function.

The robot path for this case is shown in Figure E.

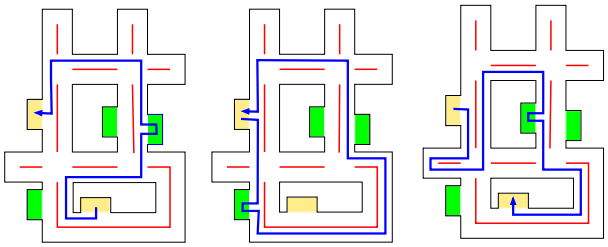


Figure E: The robot path for Case E. The robot visits g_1, then g_2 and then g_3, periodically. The value of optimization function is 9.13 meters, which is the same as in Case D.

Case F. We can also specify safety constraints for the robot. For example, consider the objective of Case D with the additional constraint that the road connecting i_4 to i_2 (illustrated in pink in Figure F) should be avoided. In this case, the specification becomes

$$\phi_F = \phi_D \wedge \mathsf{G}\,\neg(\mathsf{i}_4 \wedge \mathsf{X}\,\mathsf{i}_2)$$

The robot path for this case is shown in Figure F.

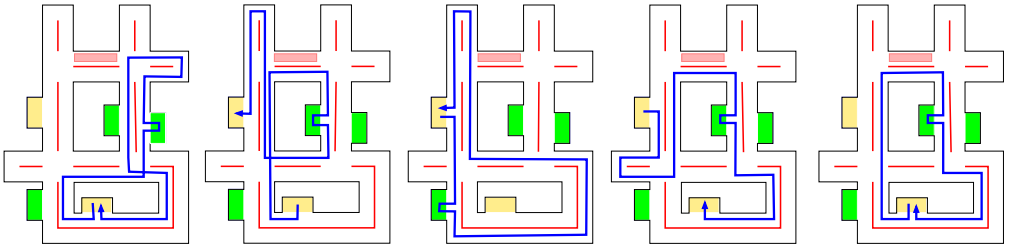


Figure F: The robot path for Case F. The robot never uses the road connecting i_4 to i_2. The value of optimization function is 10.48 meters, which is more than in Case D.

Case G. Another type of constraint may be that data from location g_3 must be uploaded at location u_2. The specification from Case D can easily be extended to incorporate this constraint:

$$\phi_G = \phi_D \wedge \mathsf{G}\,(\mathsf{g}_3 \Rightarrow (\neg \mathsf{u}_1\,\mathsf{U}\,\mathsf{u}_2)).$$

The robot path for this case is shown in Figure G. Note that from Table 5.1, the cost function value for this case lies between that from Case D and from Case F.

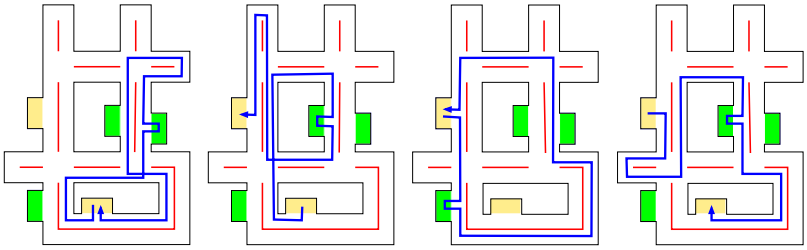


Figure G: The robot path for Case G. The robot uploads the data in u_2 after gathering them in g_3. The value of optimization function is 9.5 meters, which again exceeds that of Case D.

Remark 5.2.14 (Modeling Robot Navigation Errors)**.** *In implementing the robot paths on our testbed, there were instances in which the robot failed to make the proper transition. This occurred when the robot was following a road, turning at intersections, or entering/exiting data gather and upload locations. For example, in 50 trials of each motion primitive we observed 3 failures when performing left turns, 1 failure when performing right turns, 3 failures when entering a gather/upload location, and 1 failure when exiting a gather/upload location. When such failures occur, the robot enters a different state than expected. Our current method does not allow the robot to recover in these situations.*

Such failures can be modeled and dealt with formally by allowing for non-determinism and/or probabilistic transitions. For example, if in our experimental setup we observe that by applying a right turn motion primitive in an intersection may result in going straight through it, then we associate both going straight and right turn outcomes to this motion primitive. The transition system describing the motion of the robot in the environment becomes non-deterministic. If, in addition, we can quantify the success and failure rates of the motion primitives at different locations in environment, then we could model the motion of the robot as a Markov Decision Process. In fact, the optimal surveillance problem for MDPs has been later considered in [DSBR11], where the authors solve the problem of optimizing the expected cost. However, their solution is only sub-optimal in general.

5.3 Conclusions

In this chapter we have presented a method for planning the optimal motion of a robot subject to temporal logic constraints. Motivated by persistent monitoring and data gathering applications, we considered temporal logic specifications which contain a single *optimizing proposition* that must be repeatedly satisfied. We developed an algorithm for computing the optimal robot path that minimizes the maximum time between satisfying instances of the optimizing proposition. Experimental results show the applicability of this approach for a robot moving in a city-like environment. We also formulate the optimizing problem for systems with degradation and discuss that its solution is an easy adaptation of the suggested approach.

This work has focused on the min-max cost function formulation since it gives a hard guarantee on the time between satisfying instances. However, there are other relevant costs, such as the average time between satisfying instances. It seems likely that the approach used in this paper could be extended to solve alternate cost functions, and in our future work we will explore this direction.

5.4 Notes

This chapter is based on papers [STBR11, STBR10], which are joint work with Calin Belta and Daniela Rus, and Stephen L. Smith, who has been the leading author of this work. He and Alphan Ulusoy are also the authors of the implementation of the suggested solution. Many thanks go to Yushan Chen who helped us with the experimental setup of the platform.

The work presented in this chapter has been extended later in [DSBR11] towards the use of Markov decision processes as models of robot motion. The

authors propose a solution that provides, in general, a sub-optimal solution. However, for certain fragments, such as the surveillance fragment that we define in [CTB12, CTUB13], the solution is guaranteed to be optimal. This approach have been later enhanced in our works [CTB12, CTUB13], where we have considered unknown elements in the environment that can be partially learned during the robot's execution. Another known extension of our work is the optimal surveillance for multi-robot system, where synchronization of multiple robots needs to be addressed. Results on this topic have been presented in [USD+11, USDB12].

Chapter 6

Least-violating Robot Path Planning

This chapter is devoted to finding a least-violating robotic trace motivated by situations in robot path planning, when a given specification cannot be satisfied as a whole. We start with several motivation remarks and examples in Section 6.1.

In Section 6.2, we focus on planning with a set of long-term LTL *tasks* that are assigned a reward each. We define a metric called *trace reward* to measure quality of a trace with respect to the satisfaction of the set of tasks and we propose an algorithm to find an optimal, *i.e.,* a least-violating trace. We demonstrate our approach through a simulation rescue mission case study.

On the other hand, Section 6.3 deals with planning with a simple "Go from A to B." task together with a set of rules that need to be obeyed. Unlike in the previous case, here, we need a finer characterization than satisfaction or violation of a rule, taking into consideration for how long or how much a particular rule is violated. Therefore, we define a quantitative metric called the *level of unsafety* as a suitable metric to distinguish in between "better" and "worse" traces. We present a solution finding an optimal solution in terms of level of unsafety and demonstrate our approach on an illustrative case study.

Finally, we conclude in Section 6.4.

6.1 Motivation

The motivation for the problems addressed in this chapter comes from robot motion planning in a urban-like environment. We consider quite common scenarios, when the robotic mission cannot be accomplished as a whole, *e.g.,* due to conflicts in the specification and/or external constraints, such as obstacles present in the environment. Even if we cannot reach the desired goal fully, we might still be interested in satisfying the most important pieces of the mission and in finding a *least-violating* plan, as demonstrated through the following two examples.

Example 6.1.1 (Data Gathering Robot III)**.** *Consider a data gathering robot*

in an environment and a mission to be accomplished consisting of four tasks. The first one is "Periodically visit the data gather locations A and B and periodically visit the data upload location C," the second one says "Always gather the data in location A before delivering the data in location C," the third one is "Always gather the data in location B before delivering the data in region C," and finally the fourth one is "Always gather data only once before visiting a data upload location, i.e., never gather data twice in a row without uploading them in between." Although we would like to accomplish all four tasks, it is clear that this is not feasible as the second, the third and the fourth task are in conflict. However, instead of reporting that there is no feasible plan, we would like to obtain a robot path that is as close to satisfaction of the tasks as possible, taking into account priorities of individual tasks. For instance, if the fourth specification is the least important, the expected robot behavior would be to periodically gather data in A, then gather data in B and, then upload the collected data in C. On the other hand, if the fourth task is more important than the second and the third ones, the expected behavior would be to repeatedly gather data in A, upload them in C, then gather data in B and upload them in C.

Example 6.1.2 (Autonomous Vehicle)**.** *Consider an autonomous car and a set of road rules that are commonly required to be followed. For instance, the road rules include "Never enter the left lane." or "Never enter the sidewalk.". When driving towards a goal location, it is quite common that obstacles, such as parked cars blocking the right lane are present in the road environment and that one of the mentioned rules needs to be temporarily violated. At the same time, it is clear that the latter mentioned rule is of a higher priority and the rule to be disobeyed is the earlier one, if possible.*

In this work, we focus on two different classes of robotic missions. Intuitively, the first one includes missions that consist of individual tasks that can be either accomplished in long-term or not, such as the tasks from Example 6.1.1. The second type of missions is given as a set of rules that can be obeyed or violated, for a short moment, or forever, as in Example 6.1.2. The formal explanation of the two considered problem statements follows in Sections 6.2 and 6.3.

6.2 Planning with Conflicting Long-term Tasks

This section aims at formalizing and solving the first mentioned direction in the least-violating planning. We define *trace reward* as a suitable metric to measure the quality of a trace with respect to the specification given as a set of LTL formulas that are assigned a reward, or a priority, each. Leveraging ideas

from automata-based model checking, we find a trace that maximizes the trace reward.

6.2.1 Problem Statement

Let us consider a robot moving in a partitioned environment with its motion capabilities modeled as a labeled transition system $\mathcal{T} = (S, s_{\text{init}}, T, AP, L)$ from Section 3.2. Similarly as in Chapter 5, each region of the environment is modeled as a state of the transition system and the robot's ability to move between two regions is represented as a transition between the corresponding states. In case several controlled robots are placed in the environment, the states of the transition systems encode positions of all the robots in the environmental regions *i.e.,* for k robots, a state corresponds to an k-tuple of regions, where the i-th element of the tuple is the region in which the i-th robot is placed. The transitions between the states reflect the simultaneous motion capabilities of all the robots. The labeling function L maps each state of the transition system to a subset of atomic propositions from AP that hold true in this state, such as "Vehicle x is in a safe region."

A set of high-level missions to be accomplished by the robotic system is expressed as a set of LTL formulas $\Phi = \{\phi_1, \ldots, \phi_n\}$ over AP with priorities of their satisfaction determined by a reward function $rew : \Phi \to \mathbb{N}$. The value $rew(\phi_i)$ represents the reward that is gained if specification ϕ_i is accomplished. Without loss of generality, from now on, we assume that $rew(\phi_i) \geq rew(\phi_j)$, for all $1 \leq i \leq j \leq n$.

Given a trace r of the transition system $\mathcal{T}$, we define *trace reward* as the sum of the rewards of all formulas from Φ that are satisfied on this run.

Definition 6.2.1 (Trace Reward)**.** *Reward of a trace r of $\mathcal{T}$ is*

$$Rew_{\mathcal{T}}(r) = \sum_{\{\phi_i \,|\, r \models \phi_i\}} rew(\phi_i). \tag{6.1}$$

We are now ready to formally state our problem of finding "the best" trace of $\mathcal{T}$, *i.e.,* the *least-violating* motion of the robot (or the robots) in the environment with respect to the given set of mission specifications.

Problem Statement 6.2.2 (Maximal-rewarding trace)**.** *Given*

- *a transition system $\mathcal{T} = (S, s_{\text{init}}, T, AP, L)$;*
- *a set of LTL formulas $\Phi = \{\phi_1, \ldots, \phi_n\}$ over AP; and*
- *a reward function $rew : \Phi \to \mathbb{N}$,*

find a trace r of $\mathcal{T}$ that maximizes $Rew_{\mathcal{T}}(r)$ from Eq. (6.1).

Remark 6.2.3. *Note, that if $rew(\phi_i) = 2^{n-i}$, for each formula $\phi_i \in \Phi$, then the set Φ is in fact ordered according to the standard lexicographic ordering. In other words, it is always more important to satisfy ϕ_i than $\phi_{i+1} \wedge \ldots \wedge \phi_n$.*

A straightforward solution to Problem 6.2.2 is to consider all the possible subsets $\Phi_I = \{\phi_i \mid i \in I\}$, $I \subseteq \{1, \ldots, n\}$ of formulas from Φ and to find a trace r_I of $\mathcal{T}$ satisfying Φ_I if such a trace exists. The search can be done using one of the known model-checking algorithms (*e.g.,* the automata-based algorithm from Section 3.5). A trace r_I maximizing $Rew_{\mathcal{T}}(r_I)$ among the found ones maps to the desired robot path. However, this brute-force solution is not efficient as it requires up to 2^n model-checking procedure runs in the worst case.

In this section, we suggest a method to alleviate the high computational demand of this straightforward solution. The main idea builds on automata-based approach to model-checking. We construct a single weighted Büchi automaton $\mathcal{B}$ for formula $\bigwedge_{\phi_i \in \Phi} \phi_i$ and capture the rewards of the LTL formulas through its weights. Then, a weighted product automaton $\mathcal{P} = \mathcal{T} \otimes \mathcal{B}$ is built and an optimal accepting run of $\mathcal{P}$ is sought using a modification of the nested-DFS algorithm, with the computational complexity only slightly worse in comparison to the original nested-DFS. Roughly speaking, instead of up to 2^n model-checking procedure runs, we perform only a single execution of an altered model-checking algorithm.

6.2.2 Problem Solution

This section introduces our solution to Problem 6.2.2 in detail. First, we present the construction of the weighted Büchi automaton $\mathcal{B}$ and the weighted product automaton $\mathcal{P}$. Second, the modified nested-DFS is given. Third, we discuss the solution correctness, completeness and complexity.

Construction of the Weighted Automata

Consider the set of mission specifications $\Phi = \{\phi_1, \ldots, \phi_n\}$ that are translated (*e.g.,* using the algorithm from [GO01]) into generalized Büchi automata

$$\begin{aligned} \mathcal{G}_{\phi_1} &= (Q_1, q_{\text{init},1}, \Sigma, \delta_1, \mathcal{F}_1 = \{F_1^1, \ldots, F_1^{m_1}\}), \ldots \\ \ldots, \mathcal{G}_{\phi_n} &= (Q_n, q_{\text{init},n}, \Sigma, \delta_n, \mathcal{F}_n = \{F_n^1, \ldots, F_n^{m_n}\}), \end{aligned}$$

respectively. Without loss of generality, we assume that $\mathcal{G}_{\phi_1}, \ldots, \mathcal{G}_{\phi_n}$ are all non-blocking. We build the weighted Büchi automaton $\mathcal{B}$ leveraging ideas from translation of generalized Büchi automata to Büchi automata and from construction

of a Büchi automaton for language intersection of several Büchi automata (see Section 3.4).

Definition 6.2.4 (Weighted Büchi automaton)**.** *A weighted Büchi automaton* $\mathcal{B} = (Q, q_{\text{init}}, \Sigma, \delta, F, W)$ *is defined as follows:*

- $Q = Q_1 \times \ldots \times Q_n \times$
  $\{(0,0),(1,1),\ldots,(1,m_1),(2,1)\ldots,(2,m_2),\ldots,(n,1),\ldots,(n,m_n)\}$*;*
- $q_{\text{init}} = (q_{\text{init},1},\ldots,q_{\text{init},n},(0,0))$*;*
- $\tau = \big((q_1,\ldots,q_n,(j,l)),\sigma,(q_1',\ldots,q_n',(j',l'))\big) \in \delta$ *if* $(q_i,\sigma,q_i') \in \delta_i$*, for all* $i \in \{1,\ldots,n\}$*, and*

  (i) $(j,l) = (0,0)$ *and*

  (a) $(j',l') = (0,0)$*. Then* $W\big(\tau\big) = 0$*.*

  (b) $j' > 0, l' = 1$*. Then* $W\big(\tau\big) = rew(\phi_{j'})$*.*

  (ii) $j \neq 0$ *and*

  (a) $(j',l') = (j,l)$ *and* $q_j \notin F_j^l$*. Then* $W\big(t\big) = 0$*.*

  (b) $l \neq m_j, (j',l') = (j,l+1)$ *and* $q_j \in F_j^l$*. Then* $W\big(t\big) = 0$*.*

  (c) $l = m_j, j < n$*,* $j \leq j'$*,* $l' = 1$ *and* $q_j \in F_j^l$*. Then* $W\big(t\big) = rew(\phi_{j'})$*.*

  (d) $l = m_j, (j',l') = (0,0)$*, and* $q_j \in F_j^l$*. Then* $W\big(t\big) = 0$*.*

- $F = Q_1 \times Q_2 \times \ldots \times Q_n \times \{(0,0)\}$*.*

An example of the construction of $\mathcal{B}$ is illustrated in Figure 6.1.

With a slight abuse of notation, we use

$$W(q_i \ldots q_k) = \sum_{j=i}^{k-1} W\big((q_j, w(j), q_{j+1})\big)$$

to denote the sum of the weights between the states of subsequence $q_i \ldots q_k$ of a run $\rho = q_0 \ldots q_i \ldots q_k \ldots$ over $w = w(0)w(1)\ldots$.

The automaton $\mathcal{B}$ accepts all words satisfying specifications $\bigwedge_{\phi_i \in \Phi_I} \phi_i$, for all $\Phi_I \subseteq \Phi$. The weights determine the "quality" of a particular run, *i.e.,* they capture which formulas are satisfied by this run. Formally, the purpose of weights is summarized in the following lemma.

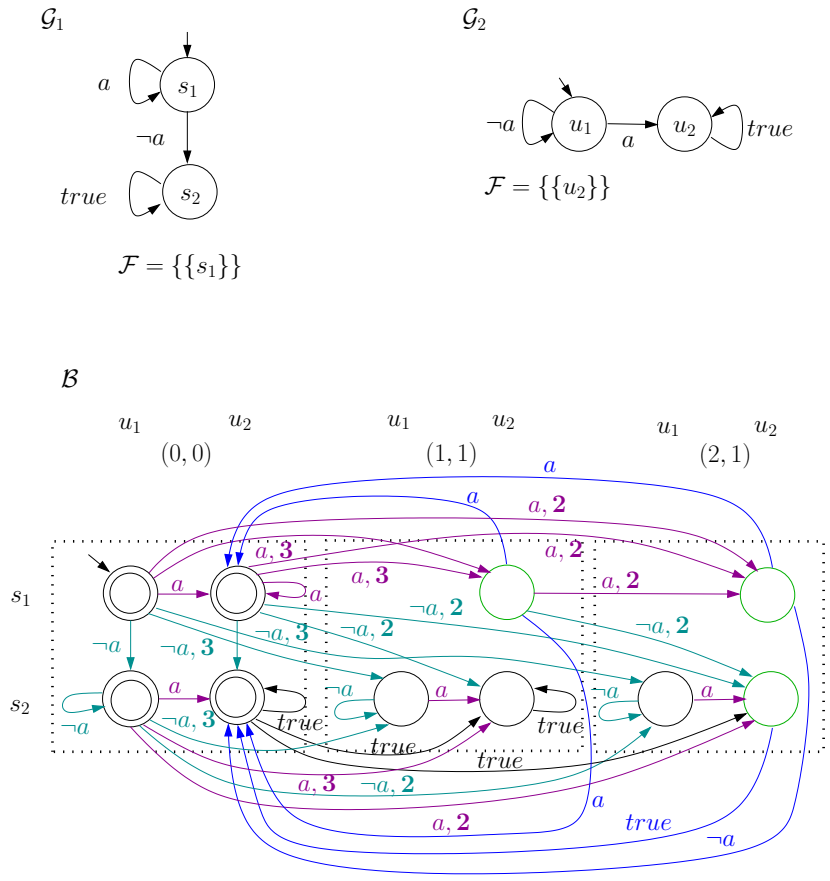


Figure 6.1: An example of generalized Büchi automatoa $\mathcal{G}_1$ and $\mathcal{G}_2$ for formulas $\mathsf{G}\, a$ and $\mathsf{F}\, a$, respectively and their weighted Büchi automaton $\mathcal{B}$ constructed according Def. 6.2.4. For simplicity, the transitions are labeled with atomic propositions and their negations instead of subsets of atomic propostions. For instance, a transition labeled with a represents all the transitions labeled with subsets of atomic propositions containing a. A transition labeled with *true* represents all $2^{|\Sigma|}$ transitions labeled with all subsets of atomic propositions. The set of states of $\mathcal{B}$ is $\{(s_i, u_j, (k,l)) \mid i \in \{1,2\}, j \in \{1,2\}, \text{ and } (k,l) \in \{(0,0)(1,1)(2,1)\}\}$, however, only states reachable from the initial state of $\mathcal{B}$ are depicted. The transitions are labeled with weights. A missing weight represents that the weight is equal to 1.

Definition 6.2.5 (Run Reward)**.** *The reward of a run ρ of $\mathcal{B}$ over w is*

$$Rew_{\mathcal{B}}(\rho) = \max\big\{R \mid R = W(q_i \dots q_l) \text{ for infinitely many fragments } q_i \dots q_l \in Frag(\rho)\big\}.$$

Intuitively, a run ρ over w can be split into a sequence of fragments that is associated with a respective sequence of fragment weights. The run reward is equal to the maximal weight that appears in the sequence of fragment weights infinitely many times.

Lemma 6.2.6. *Consider a word $w = w(0)w(1)\ldots$, where $w \models \Phi_I$ and $w \not\models \phi$, for all $\phi \notin \Phi_I$. There exists an accepting run $\rho = q_0 q_1 \ldots$ of $\mathcal{B}$ over w, such that $Rew_{\mathcal{B}}(\rho) = \sum_{\phi_i \in \Phi_I} rew(\phi_i)$. Furthermore, for each accepting run $\rho' = q'_0 q'_1 \ldots$ of $\mathcal{B}$ over w it holds, that $Rew_{\mathcal{B}}(\rho') \leq \sum_{\phi_i \in \Phi_I} rew(\phi_i)$.*

Proof. If $w \models \Phi_I$ then there is an accepting run $\rho_i = q_0 q_1 \ldots$, for all $\phi_i \in \Phi_I$. Let $I = \{i_1, \ldots, i_j\}$. According to the construction of $\mathcal{B}$ (Def. 6.2.4), there exists a run $\rho = p_0 p_1 \ldots$ of $\mathcal{B}$, such that each fragment $p_k \ldots p_{k'} \in Frag(\rho)$ satisfies the following.

$$
\begin{aligned}
layer(p_{l_1}) &= (0,0),\\
layer(p_{(l_1+1)}) &= \ldots = layer(p_{l_2}) = (i_1, 1),\\
&\ldots\\
layer(p_{l_3}) &= \ldots = layer(p_{l_4}) = (i_1, |\mathcal{F}_{i_1}|),\\
layer(p_{(l_4+1)}) &= \ldots = layer(p_{l_5}) = (i_2, 1),\\
&\ldots\\
layer(p_{l_6}) &= \ldots = layer(p_{l_7}) = (i_2, |\mathcal{F}_{i_2}|),\\
&\ldots\\
layer(p_{(l_8)}) &= \ldots = layer(p_{l_9}) = (i_j, 1),\\
&\ldots\\
layer(p_{l_{10}}) &= \ldots = layer(p_{l_{11}}) = (i_j, |\mathcal{F}_{i_j}|),\\
layer(p_{(l_{11}+1)}) &= (0,0)
\end{aligned}
$$

where $p_{l_1} = p_k$, $p_{(l_{11}+1)} = p_{k'}$. The total weight of such a fragment and hence also the reward of ρ is equal to $\sum_{\phi_i \in \Phi_I} \phi_i$ directly from the construction of $\mathcal{B}$.

On the other hand, assume that there exists a run $\rho' = q'_0 q'_1 \ldots$ of $\mathcal{B}$ over w such that $Rew_{\mathcal{B}}(\rho') > \sum_{\phi_i \in \Phi_I} rew(\phi_i)$. From the construction of the automaton $\mathcal{B}$, this means that there exist infinitely many fragments $p_k \ldots p_{k'} \in Frag(\rho')$ with their weight larger than $\sum_{\phi_i \in \Phi_I} rew(\phi_i)$. Therefore, there exist $\phi_l \notin \Phi_I$, and states $p'_1, \ldots, p'_{|\mathcal{F}_l|}$ of $\mathcal{B}$, such that $layer(p'_j) = (l, j)$, for all $j \in \{1, \ldots, |\mathcal{F}_l|\}$. Thus, the run ρ' can be projected to an accepting run of $\mathcal{G}_{\phi_l}$ over w, which is in contradiction with our assumption that $w \not\models \phi_l$ for all $\phi_l \notin \Phi_I$. □

The second step in our algorithm is the construction of a product automaton $\mathcal{P} = \mathcal{T} \otimes \mathcal{B} = (Q_{\mathcal{P}}, p_{\text{init}}, \delta_{\mathcal{P}}, F_{\mathcal{P}}, W_{\mathcal{P}})$ as follows.

- $Q_{\mathcal{P}} = S \times Q$;
- $Q_{\mathcal{P}}^{\text{init}} = \{(s, q) \mid s \in S_{\text{init}} \text{ and } q \in Q_{\text{init}}\}$;
- $((s, q), (s', q')) \in \delta_{\mathcal{P}}$ if there exists $\alpha \in Act$, such that $(s, \alpha, s') \in T$ and $(q, L(s), q') \in \delta$;
- $W_{\mathcal{P}}\big(((s, q), (s', q'))\big) = W\big((q, L(s), q')\big)$; and
- $F_{\mathcal{P}} = S \times F$.

Based on Lemma 6.2.6, the product automaton satisfies the following:

Lemma 6.2.7. *Let r be a trace of $\mathcal{T}$. Then, there exists a run $\rho_{\mathcal{P}}$ of $\mathcal{P}$ such that the reward for the corresponding trace $r = proj(\rho_{\mathcal{P}})$ on $\mathcal{T}$ satisfies $Rew_{\mathcal{T}}(r) = Rew_{\mathcal{B}}(\rho_{\mathcal{P}})$. Moreover, $Rew_{\mathcal{T}}(r) \geq Rew_{\mathcal{B}}(\rho'_{\mathcal{P}})$ for all $\rho'_{\mathcal{P}}$ with $proj(\rho'_{\mathcal{P}}) = r$.*

Proof. The proof follows directly from Lemma 6.2.6 and the fact that for each trace r that produces a word $w = w(0)w(1)\ldots$ accepted by a run $\rho = q_0 q_1 \ldots$ of $\mathcal{B}$, there exists an accepting run $\rho_{\mathcal{P}} = p_0 p_1 \ldots$ of $\mathcal{P}$, such that $W_{\mathcal{P}}\big((p_i, p_{i+1})\big) = W\big((q_i, w(i), q_{i+1})\big)$ and $q_i \in F \iff p_i \in F_{\mathcal{P}}$, for all $i \geq 0$. □

Lemma 6.2.8. *For each run $\rho_{\mathcal{P}}$ there exists a run $\rho'_{\mathcal{P}}$ in prefix-suffix structure, such that $Rew_{\mathcal{B}}(\rho_{\mathcal{P}}) = Rew_{\mathcal{B}}(\rho'_{\mathcal{P}})$.*

Proof. Because $\rho_{\mathcal{P}} = p_0 p_1 \ldots$ is infinite, there exist a state $p \in F_{\mathcal{P}}$ that appears on $\rho_{\mathcal{P}}$ infinitely many times and there exist a fragment $p \ldots p'$ starting in p such that $W(p \ldots p') = Rew_{\mathcal{B}}(\rho_{\mathcal{P}})$. Because p occurs on $\rho_{\mathcal{P}}$ infinitely many times, p is reachable from p'. Therefore, run $\rho_{\mathcal{P}}$ is a sequence of states $\rho_{\mathcal{P}} = p_0 p_1 \ldots p \ldots p' \ldots p \ldots$. Let $\rho'_{\mathcal{P}} = p_0 p_1 \ldots (p \ldots p' \ldots p)^{\omega}$. Run $\rho'_{\mathcal{P}}$ is in prefix-suffix structure, it is accepting and $Rew_{\mathcal{B}}(\rho'_{\mathcal{P}}) = Rew_{\mathcal{B}}(\rho_{\mathcal{P}})$. □

The two lemmas above provide us with guidance on computing the trace of $\mathcal{T}$ with the maximal reward: it is enough to compute a run of $\mathcal{P}$, in the prefix-suffix structure, that maximizes $Rew_{\mathcal{B}}(\rho_{\mathcal{P}})$ and project this run onto a trace of $\mathcal{T}$. This is stated in the following proposition.

Proposition 6.2.9. *Let $r = s_0 s_1 \ldots$ be a trace of $\mathcal{T}$, such that $r \models \Phi_I$ and $r \not\models \phi$, for all $\phi \notin \Phi_I$. Then, there exists an accepting run $\rho_{\mathcal{P}} = (s_0, q_0)(s_1, q_1)\ldots$ of $\mathcal{P}$ such that*

(i) $\rho_{\mathcal{P}}$ is in prefix-suffix structure and

(ii) $Rew_{\mathcal{B}}(\rho_{\mathcal{P}}) = \sum_{\phi_i \in \Phi_I} \phi_i$.

The remaining task is to find a run $\rho_{\mathcal{P}}$ satisfying the condition (i) of Proposition 6.2.9 and maximizing $Rew_{\mathcal{B}}(\rho_{\mathcal{P}})$.

We now reduce the problem to graph search on the product automaton graph. Particularly, we view the product automaton$\mathcal{P}$ as a weighted directed graph $(Q_{\mathcal{P}}, \delta_{\mathcal{P}}, W_{\mathcal{P}})$ that consists of a vertex set $Q_{\mathcal{P}}$, an edge set $\delta_{\mathcal{P}}$ and a weight function $W_{\mathcal{P}} : \delta_{\mathcal{P}} \to \mathbb{N}$. The usual graph structures, such as path, simple path and cycle are defined in the expected way, analogously as for the non-weighted version of a directed graph (see Section 3.4). Furthermore, we extend the notion of fragments to graph cycles. Let $\rho_{\text{cycle}} = p_0 \dots p_m$. Then

$$Frag(\rho_{\text{cycle}}) = \{p_i \dots p_l \mid \ p_i, p_l \in F_{\mathcal{P}}, 0 \leq i \leq l \leq m \text{ and } p_j \notin F_{\mathcal{P}}, \forall i < j < l\}.$$

We also define *the maximal simple distance* from $q_f \in F$ to q in a weighted Büchi automaton $\mathcal{A}$ is the maximal sum of edge weights on a simple path $q_i \dots q_l$ from $q_i = q_f$ to $q_l = q$, such that $q_j \notin F$, for all $i < j < l$.

Finally, we are ready to cast the problem as the search for a reachable cycle ρ_{cycle} (a repeated run suffix) in the product automaton graph $\mathcal{P}$ beginning (and thus also ending) in an accepting state that maximizes the value

$$Rew_{\mathcal{B}}(\rho_{\text{cycle}}) = \max_{p_i \dots p_l \in Frag(\rho_{\text{cycle}})} W(p_i \dots p_l) \tag{6.2}$$

among all such cycles. The following lemma helps narrow down the search even further, showing that it is enough to search for a particular type of cycle.

Lemma 6.2.10. *Given a cycle ρ_{cycle} in $\mathcal{P}$ and a fragment $p_i \dots p_l \in Frag(\rho_{\text{cycle}})$, there exists a simple path $p_i \dots p_l$, such that $W(p_i \dots p_l) = 0$.*

Proof. From the construction of the automaton $\mathcal{B}$, it follows that if there is a simple path from $(q_1, \dots, q_n, (0,0)) \in Q$ to $(q'_1, \dots, q'_n, (i,j)) \in Q$ in the automaton $\mathcal{B}$, then there exists a simple path from $(q_1, \dots, q_n, (0,0))$ to $(q'_1, \dots, q'_n, (0,0))$ that contains only states $\mathbf{q} \in Q$, such that $layer(\mathbf{q}) = (0,0)$. Thus, if there is a simple path from $(s, q_1, \dots, q_n, (0,0)) \in Q_{\mathcal{P}}$ to $(s', q'_1, \dots, q'_n, (i,j)) \in Q_{\mathcal{P}}$ in the product automaton $\mathcal{P}$, then there exists a simple path from $(s, q_1, \dots, q_n, (0,0))$ to $(s', q'_1, \dots, q'_n, (0,0))$ that contains only states $p \in Q_{\mathcal{P}}$, such that $layer(p) = (0,0)$ The reward of such a simple path is 0. □

Thanks to Lemma 6.2.10, it is enough to search for a cycle ρ_{cycle} maximizing $Rew_{\mathcal{B}}(\rho_{\text{cycle}})$ in (6.2), such that $W(p_i \dots p_l) = 1$, for all fragments $p_i \dots p_l \in Frag(\rho_{\text{cycle}})$, but one. Hence, without loss of generality, we can consider only cycles $\rho_{\text{cycle}} = p_i \dots p_l p_{l+1} \dots p_i$ such that $W(p_i \dots p_l) \neq 1$ only for the first fragment $p_i \dots p_l$ of the cycle. Such a cycle can be found by adapting standard nested depth-first search algorithm as we will show in the following section.

Proposition 6.2.11. *A maximal-reward trace of $\mathcal{T}$ can be obtained as a projection $proj(p_0 \dots p_l) \cdot (proj(\rho_{\text{cycle}}))^\omega$ of a path $p_0 \dots p_l$ and a cycle $\rho_{\text{cycle}} = p_{l+1} \dots p_i p_{i+1} \dots p_{l+1}$, such that*

- $p_0 = p_{\text{init}}$, $(p_l, p_{l+1}) \in \delta_{\mathcal{P}}$, $p_{l+1} \in F_{\mathcal{P}}$,
- $W(p_j \dots p_k) = 1$, *for all fragments* $p_j \dots p_k \in Frag(p_{i+1} \dots p_{l+1})$, *and*
- $W(p_{l+1} \dots p_i)$ *for the first fragment* $p_{l+1} \dots p_i \in Frag(\rho_{\text{cycle}})$ *of the cycle is maximized.*

Weighted Nested Graph Search

Let us now aim at search for a path $p_0 \dots p_l$ followed by a cycle $p_{l+1} \dots p_{l+1}$ satisfying conditions of Proposition 6.2.11. The solution is summarized in Algorithms 11–13.

First, let us focus on a solution of the following sub-problem: Given an accepting state $p_f \in F_{\mathcal{P}}$, find a cycle ρ_{cycle} that maximizes value in Eq. (6.2) among all cycles that begin and end in p_f. A modification of breath-first graph search algorithm can be used to do so as described in Algorithm 13. Intuitively, the algorithm systematically searches the graph $\mathcal{P}$ and maintains in $p.dist$ the approximation of the maximal simple distance from p_f to p, for each state p. The correctness of the algorithm relies on the fact, that when p is processed on line 2 of the procedure propagate (Algorithm 14), the value of $p.dist$ is set to the actual maximal simple distance from p_f to p. When all states that are reachable from state p_f are visited in Algorithm 14, the second phase (lines 14-25) of Algorithm 13 is executed to check whether p_f is also reachable from p, considering p one by one in descending order of their $p.dist$.

Second, the cycle satisfying conditions of Proposition 6.2.11 can be found by running Algorithm 13 from each $p_f \in F_{\mathcal{P}}$ reachable from the initial state, potentially traversing the whole graph $|F_{\mathcal{P}}|$-times. However, leveraging ideas from the nested DFS algorithm, the complexity can be reduced. The idea is to run Algorithm 13 from states in $F_{\mathcal{P}}$ in particular order that ensures the states visited during previous executions of Algorithm 13 do not need to be visited again. The correctness of this idea is based on the correctness of the standard nested DFS [CVWY92b] and the following lemma.

Lemma 6.2.12. *Let $p'_f \in F_{\mathcal{P}}$ be reachable from $p_f \in F_{\mathcal{P}}$ and $p \in Q_{\mathcal{P}}$ be reachable from both p_f and p'_f. If there exists a cycle ρ_{cycle} from state $p_f \in F_{\mathcal{P}}$ containing state p, then there exists a cycle ρ'_{cycle} from $p'_f \in F_{\mathcal{P}}$ with reward $Rew_{\mathcal{B}}(\rho'_{\text{cycle}}) \geq Rew_{\mathcal{B}}(\rho_{\text{cycle}})$.*

Proof. Because p_f is reachable from p, p is reachable from p'_f, and p'_f is reachable from p_f, then p_f is reachable from p'_f. Therefore, there exists a cycle $\rho'_{\text{cycle}} = p'_f \dots p_f \dots p \dots p_f \dots p'_f$, where $p_f \dots p \dots p_f = \rho_{\text{cycle}}$. Clearly $Rew_{\mathcal{B}}(\rho'_{\text{cycle}}) \geq Rew_{\mathcal{B}}(\rho_{\text{cycle}})$. □

Algorithm 11 weighted_nested_search($\mathcal{P}$)

Input: product automaton $\mathcal{P} = (Q_{\mathcal{P}}, p_{\text{init}}, \delta_{\mathcal{P}}, F_{\mathcal{P}}, W_{\mathcal{P}})$
Output: solution to Problem 6.2.2

$weight\_max = 0; prefix\_max = \varepsilon; cycle\_max = \varepsilon$
$stack\_outer = \text{empty}; visited\_outer = \varnothing$
$visited\_inner = \varnothing; visited\_ps = \varnothing$
for all $p \in Q_{\mathcal{P}}$ **do**
  $p.dist = 0; p.pred = \perp$
end for
$run := \mathsf{DFS}(\mathcal{P}, p_{\text{init}})$
if $run \neq \varepsilon$ **then**
  return $trace := proj(run)$
else
  return find_arbitrary_trace($\mathcal{T}$)
end if

Algorithm 12 DFS($\mathcal{P}, p$)

$stack\_outer$.push(p); $visited\_outer := visited\_outer \cup \{p\}$
repeat
  $p' := stack\_outer$.top()
  if successors(p') $\setminus$ $visited\_outer \neq \varnothing$ **then**
    pick $p'' \in$ successors(p') $\setminus$ $visited\_outer$
    $stack\_outer$.push(p'')
    $visited\_outer := visited\_outer \cup \{p''\}$
  else
    $stack\_outer$.pop()
    if $p' \in F_{\mathcal{P}}$ **then**
      $p'.dist := 0$; $p'.pred = \perp$
      $cycle :=$ longest_cycle_search($\mathcal{P}, p'$)
      if $p'.dist > weight\_max$ **then**
        $weight\_max := p'.dist; cycle\_max := cycle$
        $prefix\_max :=$ reverse($stack\_outer$)
      end if
    end if
  end if
until ($stack\_outer = \text{empty} \vee weight\_max = n$)
return $prefix\_max \cdot (cycle\_max)^{\omega}$

Algorithm 13 longest_cycle_search$(\mathcal{P}, p_f)$

Input: product automaton $\mathcal{P}$, accepting state $p_f \in F_{\mathcal{P}}$
Output: cycle $(p_f, \ldots, p_f)$ maximizing Eq. 6.2 (if one exists)

```
queue_curr := (p_f)
for all 1 ≤ i ≤ n do
    queues_all[i] := empty
end for
to_search_from := ∅
propagate(P, queue_curr, queues_all, search_from)
for all 1 ≤ i ≤ n do
    queue_curr := queues_all[i]
    if queue_current ≠ empty then
        propagate(P, queue_curr, queues_all, search_from)
    end if
end for
cycle := ε
order search_from decreasingly according to p.dist
while search_from ≠ empty do
    p := search_from.top_and_pop()
    path_suf := find_path(P, p, p_f)
    if path ≠ ε then
        p_f.dist := p.dist; path_pref := ε
        repeat
            path_pref := (p.pred) · (path_pref); p := p.pred
        until p = p_f
        return cycle := (path_pref) · (path_suf)
    end if
end while
return cycle := ε
```

The overall solution can be summarized as follows:

(i) Each of the formulas $\phi_i \in \Phi$ is translated into a generalized Büchi automaton $\mathcal{G}_{\phi_i}$

(ii) A weighted Büchi automaton $\mathcal{B}$ is built (see Def. 6.2.4)

(iii) A weighted product automaton $\mathcal{P} = \mathcal{T} \otimes \mathcal{B}$ is constructed (see Def. 3.5.2).

(iv) Algorithm 11 is run on $\mathcal{P}$.

Correctness and Complexity

Based on Lemmas 6.2.6–6.2.12 and Propositions 6.2.9–6.2.11, the soundness and completeness properties of the algorithm are summarized in the following theorem.

Algorithm 14 propagate($\mathcal{P}, queue\_curr, queues\_all, search\_from$)

```
1: repeat
2:    p := queue_curr.front_and_pop()
3:    if p ∉ visited_inner then
4:       visited_inner := visited_inner ∪ {p}
5:       for all p' ∈ succs(p) do
6:          if p'.dist < p.dist + W_P(p, p') then
7:             p'.dist := p.dist + W_P(p, p'); p'.pred := p
8:             if layer(p') = 0 ∧ p' ∉ search_from then
9:                search_from = search_from ∪ {p'}
10:            end if
11:            if layer(p') = layer(p) then
12:               queue_curr.push(p')
13:            else if layer(p') = i for some i ≥ 1 then
14:               queues_all[i].push(p')
15:            end if
16:         end if
17:      end for
18:   end if
19: until queue_curr = empty
```

Theorem 6.2.13 (Correctness and Completness)**.** *Given a transition system $\mathcal{T}$, a set of LTL formulas Φ and the reward function rew, the suggested algorithm returns solution to the maximal-rewarding trace problem (Problem 6.2.2).*

Let $|\mathcal{T}|$ and $|\Phi|$ denote the size of the input transition system and the size of the missions specification, respectively. The computational complexity of Algorithm 11 is in $\mathcal{O}(|\mathcal{P}| \cdot \log |\mathcal{P}|)$, where $|\mathcal{P}|$ is the size of the product automaton, which is in $\mathcal{O}(|\mathcal{T}| \cdot 2^{\mathcal{O}(|\Phi|)})$.

The translation from an LTL formula ϕ into a generalized Büchi automaton can be done in in $2^{\mathcal{O}(|\phi|)}$ time and space. In particular, one of the well-known translation algorithms [GO01] transforms ϕ into a generalized Büchi automaton with at most $2^{|\phi|}$ states and $|\phi|$ sets in its acceptance condition. If the obtained GBAs for specifications $\phi_1 \dots \phi_n \in \Phi$ are all non-blocking, the worst-case size of $\mathcal{B}$ is $2^{(|\Phi|)} \cdot (1 + |\Phi|)$. On the other hand, in case k of the obtained GBAs are all blocking, the worst-case size of $\mathcal{B}$ is $2^{(|\Phi|+k)} \cdot (|\Phi|+k+1)$. Although the size of the resulting GBA is exponential with respect to the size of the input specification, the sizes of the individual formulas are usually small and in many cases, the GBAs are significantly smaller than the worst-case bound. Many optimizations techniques have been also developed among the formal methods literature to reduce the sizes of the GBAs.

The size of the product automaton $\mathcal{P}$ is $|\mathcal{T}| \cdot |\mathcal{B}|$ in the worst case, with

at most $|\mathcal{T}| \cdot 2^{(|\Phi|+k)}$ in one layer, where k is the number of blocking GBAs obtained in translation of the formulas from Φ. The cumulative number of steps made in sorting the set $search\_from$ on line 14 of Algorithm 13 is bounded by $\mathcal{O}(|L_{\mathcal{P}}| \cdot \log |L_{\mathcal{P}}|)$, where $|L_{\mathcal{P}}| = \{p \in Q_{\mathcal{P}} \mid layer(p) = (0,0)\}$ is the size of the initial layer of $\mathcal{P}$. Altogether, the complexity of Algorithm 11 is in $\mathcal{O}(|\mathcal{P}| + |L_{\mathcal{P}}| \cdot \log |L_{\mathcal{P}}|)$.

In contrast, the "brute-force" approach that tries to find a trace satisfying Φ_I, for each $\Phi_I \subseteq \Phi$ has the worst case complexity characterized as follows. A Büchi automaton $\mathcal{B}_{\Phi_I}$ for Φ_I can be constructed with $2^{|\Phi_I|} \cdot |\Phi_I|$ number of states in the worst case. A nested DFS algorithm is then run on $\mathcal{P} = \mathcal{T} \otimes \mathcal{B}_{\Phi_I}$, reaching complexity $\mathcal{O}(|\mathcal{P}|)$. Hence, the solution is linear with respect to the size of $|\mathcal{T}| \cdot \sum_{\Phi_I \subseteq \Phi} 2^{|\Phi_I|} \cdot |\Phi_I|$.

The benefit of our algorithm (Algorithm 11) in comparison to the brute-force solution increases with the increasing number of *non-blocking* GBAs obtained from the translation from LTL formulas. Note, that for some LTL formulas, the smallest existing corresponding GBA is non-blocking. In particular many useful specifications, such as $\mathsf{F}\phi$ (reachability), $\mathsf{G}\,\mathsf{F}\,\phi$ (surveillance), $\mathsf{G}\,\mathsf{F}\,\phi_1 \Rightarrow \mathsf{G}\,\mathsf{F}\,\phi_2$ (reactivity), $\mathsf{G}\,(\phi_1 \Rightarrow \mathsf{F}\,\phi_2)$ (response), or $\mathsf{F}\,(\phi_1 \wedge \mathsf{F}\,\phi_2)$ (sequencing), where ϕ, ϕ_1, ϕ_2 are arbitrary Boolean combinations of atomic propositions, belong to this class.

6.2.3 Rescue Mission Example

To demonstrate our approach, we consider an example of a complex military rescue mission. Assume that friendly units $\mathsf{F}_1, \mathsf{F}_2, \mathsf{F}_3$ have been captured in an enemy territory. They are guarded by enemy units (called targets) $\mathsf{T}_1, \ldots, \mathsf{T}_7$, which need to be engaged before an autonomous vehicle can proceed to pick up the captured friendly units and bring them to the friendly base. A particular configuration is depicted in Figure 6.2.(a). While friendly units F_1 and F_2 can be rescued by engaging targets $\mathsf{T}_1, \mathsf{T}_4$, and $\mathsf{T}_2, \mathsf{T}_6, \mathsf{T}_7$, respectively, unit F_3 can be rescued by engaging targets T_3 and T_2. Suppose that we have two unmanned aerial vehicles (UAVs) V_1 and V_2 and an autonomous ground vehicle V_3 under our command, with their capabilities and weaknesses as described below.

- V_1 can engage $\mathsf{T}_1, \mathsf{T}_3$, is vulnerable to $\mathsf{T}_2, \mathsf{T}_5$, and can engage $\mathsf{T}_4, \mathsf{T}_6, \mathsf{T}_7$ at the cost of self-destruction (*i.e.,* it can be sacrificed to engage a target $\mathsf{T}_4, \mathsf{T}_6$, or T_7).

- V_2 can engage $\mathsf{T}_2, \mathsf{T}_5$, is vulnerable to $\mathsf{T}_1, \mathsf{T}_3$, and can engage $\mathsf{T}_4, \mathsf{T}_6, \mathsf{T}_7$ at the cost of self-destruction.

- V_3 can pickup and transport $\mathsf{F}_1, \mathsf{F}_2, \mathsf{F}_3$, but is vulnerable to all active targets.

The mission is to rescue and pickup the friendly units $\mathsf{F}_1, \mathsf{F}_2$ and F_3 and bring them to the base (Base). At the same time, the goal is not to loose any of the vehicles $\mathsf{V}_1, \mathsf{V}_2, \mathsf{V}_3$. Let $\mathsf{p}_{\mathsf{V}_i}^{\mathsf{F}_j}$, $\mathsf{p}_{\mathsf{V}_k}^{\mathsf{Base}}$ and $\mathsf{a}_{\mathsf{V}_\ell}$ denote the atomic propositions "Vehicle V_k is at the location of the friendly unit F_j", "Vehicle V_i is at Base", and "Vehicle V_ℓ is active", respectively. Individual goals are expressed as LTL formulas (see Table 6.1) and assigned priorities through the reward function. The reward function, among others, specifies that saving the friendly units is more important than not loosing the vehicles $\mathsf{V}_1, \mathsf{V}_2, \mathsf{V}_3$. Note, that because enemy target T_7 cannot be destroyed by any of the vehicles at no cost to their integrity, at least one vehicle must be sacrificed to save the friendly units. Although not so obvious, one can also observe that friendly units F_1 and F_2 cannot both be rescued.

Mission Specification	LTL Formula ϕ	$rew(\phi)$
Pickup F_i, and bring it to Base, for all $i \in \{1,2,3\}$	$\mathsf{F}\,(\,\mathsf{p}_{\mathsf{V}_3}^{\mathsf{F}_i} \wedge \mathsf{F}\,(\,\mathsf{p}_{\mathsf{V}_3}^{\mathsf{Base}}\,)\,)$, for $i \in \{1,2,3\}$	10
Do not pick up F_3 before picking up F_i, for all $i \in \{1,2\}$	$\mathsf{G}\,(\,\mathsf{p}_{\mathsf{V}_3}^{\mathsf{F}_3} \Rightarrow \mathsf{G}\,(\,\neg\,\mathsf{p}_{\mathsf{V}_3}^{\mathsf{F}_i}\,)\,)$, for $i \in \{1,2\}$	10
Do not lose V_k and bring V_k to Base, for all $k \in \{1,2,3\}$	$\mathsf{G}\,(\,\neg\,\mathsf{a}_{\mathsf{V}_k}\,) \wedge \mathsf{F}\,(\,\mathsf{p}_{\mathsf{V}_k}^{\mathsf{Base}}\,)$, for $k \in \{1,2,3\}$	1

Table 6.1: Mission specification.

In order to validate our algorithm, we developed a `C++` implementation which takes as an input a deterministic transition system and a list of generalized Büchi automata obtained from the LTL formulas with the use of an off-the-shelf tool such as LTL2BA [GO13]. The reward gained if the optimal control strategy of the vehicles is applied is 22 units, as expected. Figures 6.2 (top right) to 6.2 (bottom right) illustrate different stages of the system run. First, vehicles V_1 and V_2 engage enemy targets T_1 and T_5, respectively (Figure 6.2 (top right)). Then, V_1 destroys enemy target T_3 before launching a self-destructive attack on T_4 (Figure 6.2 (bottom left)). Later, vehicle V_2 engages enemy target T_2, and vehicle V_3 proceeds to pickup F_1 and F_3, in that order (see Figure 6.2 (bottom right)). Finally, the remaining vehicles return to Base.

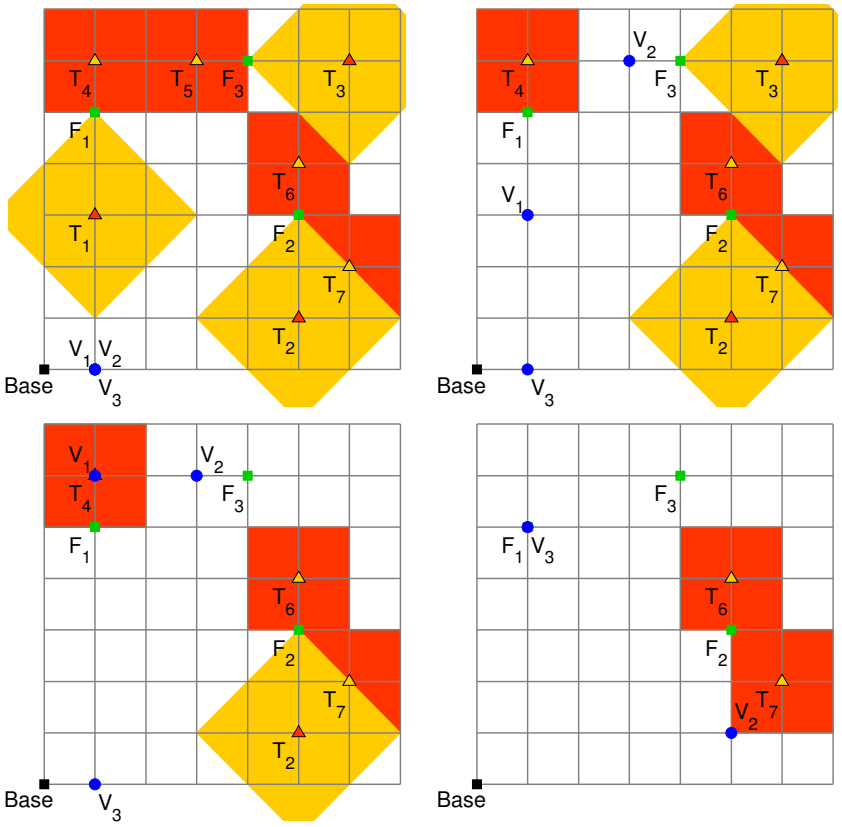


Figure 6.2: An example of a mission featuring conflicting specifications. The captured friendly units F_1, F_2, and F_3 are shown as green squares and the enemy units (the targets) $T_1, \ldots, T_7$ are illustrated as triangles. The respective firing ranges of the targets are depicted as yellow and red squares around the targets. The friendly vehicles V_1, V_2, V_3 are the blue dots, that can move along the edges of the rectangular grid. A visit of vehicle V_1 or V_2 into a location with a target is considered an engagement of the target. On the other hand, vehicle V_1, V_2, or V_3 entering a region within the firing range of a target to which it is vulnerable results in the loss of the vehicle. These rules are captured through irreversible transitions of the underlying state transition system.

6.3 Planning with Road Rules

This section aims at formalizing and solving the problem of least-violating control strategy synthesis for systems with a set of safety rules that cannot be simultaneously obeyed. We define the level of unsafety as a suitable metric to measure the quality of a trace with respect to the specification given as a set of finite automata that are assigned priorities. Leveraging ideas from automata-based model checking, we find a trace that minimizes the level of unsafety.

6.3.1 Problem Statement

In this section, we first explain informally the metrics that we use to measure the quality of a trace and then, after presenting some intermediate definitions, we provide a formal description of our problem.

Let us consider a robot moving in a partitioned environment with its motion capabilities modeled as a deterministic weighted transition system $\mathcal{T} = (S, s_{\text{init}}, R, AP, L, W)$ (see Section 3.2). Similarly as in Chapter 5 and Section 6.2 of this chapter, we model each region of the environment as a state of the transition system and and the robot's ability to move between the adjacent regions through the transition relation. The weight function assigns each transition a weight, such as distance or travel time. Our robot mission is simply "Go from A to B." while abiding by a set of (safety) rules that are assigned a priority each. Therefore, unlike in the rest of this thesis, in this section we will consider finite semantics of the transition system.

Consider a trace that starts at the region with label A and reaches that with label B. Suppose for the moment that there is a single safety rule, which we denote by ψ. Although its priority, denoted by $\varpi(\psi)$, is not important for now, its role will become clear shortly when we consider multiple safety rules. Examples of such rules include "Never enter region C." and "Region D can only be visited after visiting region E.".

We assume that ψ can be given in any form, as long as it can be translated into a finite automaton $\mathcal{A}_\psi$. In particular, it can be a regular expression or a syntactically safe FLTL formula that is interpreted over finite words (see Section 3.3). For notational simplicity, we denote by $r \models \psi$ the fact that the trace r of $\mathcal{T}$ satisfies property ψ, *i.e.,* that $r \models \mathcal{A}_\psi$. Furthermore, in the sequel, we tacitly assume that the empty trace of $\mathcal{T}$ and the empty word ε always satisfy any safety rule ψ. That is, by convention, an empty run does not violate any safety rules.

We define the *level of unsafety* $\lambda(r, \psi)$ of a finite trace r of the transition system $\mathcal{T}$ as the minimum number of transitions in r that must "vanish" so

that $r \models \psi$ holds, weighted by the priority $\varpi(\psi)$. Intuitively, the level of unsafety can be seen as the number of steps that violate the given safety rule multiplied by $\varpi(\psi)$. For instance, a trace $s_A s_D s_C s_E s_C s_B$, where $s_X \in S$ for all $x \in \{A, B, C, D, E\}$, visits regions A, D, C, E, C, B in this order has the level of unsafety equal to2 for the rule ψ_1 "Never enter region C.", with $\varpi(\psi_1) = 1$ and it has the level of unsafety equal to 3 for the rule ψ_2 "Region D can only be visited after visiting region E.", where $\varpi(\psi_2) = 3$. On the other hand, the level of unsafety of a trace visiting regions A, D, B is 0 for the former rule and remains 3 for the latter one. The smaller the level of unsafety of a trace, the closer the trace is to a trace that satisfies the given safety rule. Indeed, the trace satisfies the rule ψ if and only if its level of unsafety is equal to 0.

More formally, let $\mathcal{T} = (S, s_{\text{init}}, T, AP, L)$ be a deterministic transition system, and let ψ be a property over AP that can be expressed as a finite automaton.

Definition 6.3.1 (Level of unsafety for a safety rule)**.** *Let $r = s_0 \dots s_n$ be a finite trace of $\mathcal{T}$ producing a word $w(r) = w(0)w(1)\dots w(n)$ and $\{i_1, \dots, i_k\} \subseteq \{0, \dots, n\}$. Define*

$$\text{vanish}(r, \{i_1, \dots i_k\}) = s_0 \dots s_{i_1-1} s_{i_1+1} \dots s_{i_k-1} s_{i_k+1} \dots s_n,$$

i.e., the finite sequence of states obtained by erasing states indexed with $i_1, \dots, i_k$ from r.

The level of unsafety $\lambda(r, \psi)$ of a finite trace r with respect to a safety rule ψ is the cardinality-wise minimal $I \subseteq \{0, \dots n\}$ such that the trace $\text{vanish}(r, I)$ *satisfies the safety rule ψ:*

$$\lambda(r, \psi) = \min_{I \subseteq \{0,\dots,n\},\, \text{vanish}(r,I) \models \psi} |I| \cdot \varpi(\psi) \tag{6.3}$$

The level of unsafety of the word $w(r)$ is defined analogously.

Now, consider a sequence of nonempty sets of safety rules $\Phi = (\Psi_1, \dots, \Psi_n)$. Suppose each rule $\psi \in \Psi_i$, for all $1 \leq i \leq n$, is given in any form that can be translated into a finite automaton. Define the *priority function* as $\varpi : \Phi \to \mathbb{N}$, which assigns a *priority* to each rule $\psi \in \Psi_i$, for all $1 \leq i \leq n$. In the sequel, the ordered set Φ together with the priority function ϖ will be called a *set of safety rules with priorities* (Φ, ϖ).

Given a transition system $\mathcal{T}$ and a set of rules with priorities, denoted by $(\Phi = (\Psi_1, \dots, \Psi_n), \varpi)$, we extend the definition of the level of unsafety for a trace r of $\mathcal{T}$ to a set of safety rules $\Psi_i \in \Phi$ and a set of safety rules with priorities (Φ, ϖ) as follows.

Definition 6.3.2 (Level of unsafety for a set of rules)**.** *Level of unsafety $\lambda(r, \Psi_i)$ of a finite trace r with respect to a set of safety rules Ψ_i is defined as*

$$\lambda(r, \Psi_i) = \sum_{\psi \in \Psi_i} \lambda(r, \psi). \tag{6.4}$$

Level of unsafety $\lambda(r, \Phi)$ of finite trace r with respect to a set of safety rules with priorities (Φ, ϖ) is defined as a tuple

$$\lambda(r, \Phi) = \big(\lambda(r, \Psi_1), \ldots, \lambda(r, \Psi_n)\big). \tag{6.5}$$

The measure of unsafety, as given by Def. 6.3.2, implies an ordering relation, which we formalize below.

Definition 6.3.3 (Unsafety order)**.** *A finite trace r is said to be safer than a finite trace r' if and only if*

$$\lambda(r, \Phi) \leq \lambda(r', \Phi)$$

in the standard lexicographical ordering, i.e., iff

- $\lambda(r, \Phi) = \lambda(r', \Phi)$, *or*
- $\exists 1 \leq i \leq n$, *such that* $\lambda(r, \Psi_i) < \lambda(r', \Psi_i)$ *and* $\lambda(r, \Psi_j) \leq \lambda(r', \Psi_j)$, *for all* $j < i$.

Then, our problem can be formally stated as follows.

Problem Statement 6.3.4 (Minimum-violating Trace)**.** *Given*

- *a weighted deterministic transition system* $\mathcal{T} = (S, s_{\text{init}}, T, AP, L, W)$;
- *a task "Go from* A *to* B," *where* $\mathsf{A}, \mathsf{B} \in AP$, *and* $\mathsf{A} \in L(s_{\text{init}})$; *and*
- *a set of safety rules with priorities* (Φ, ϖ) *over* AP,

find a finite trace r of $\mathcal{T}$ that

(i) satisfies the task;

(ii) is safer than all the other traces satisfying the condition (i); and

(iii) minimizes $W(r)$ among the traces satisfying the conditions (i) and (ii).

Intuitively, the sets $\Psi_1, \ldots \Psi_n$ from Φ can be seen as *priority classes*. According to Def. 6.3.3, it is always preferred to satisfy the rules in Ψ_i rather than those in Ψ_j, for all $i < j$. However, one can trade off when violating rules that are in the same priority class. For an illustrative example, see Section 6.3.4.

6.3.2 Problem Solution

In this section, we first describe the proposed approach. Along the way, we provide intermediate results to subsequently prove the correctness and analyze the computational complexity of our algorithm. Before closing this section, we also consider a more general problem description, and point out how the proposed algorithm can be modified to address the resulting problem.

The Proposed Approach

In our solution, we leverage ideas from automata theory, graph algorithms and reachability analysis of transition systems. First, we translate each specification $\psi_{i,j}$ in $\Psi_i \in \Phi$ into a finite automaton $\mathcal{A}_{i,j}$. We propose a method to augment this automaton with new transitions and weights, such that the resulting weighted automaton $\overline{\mathcal{A}}_{i,j}$ also accepts all words w that do *not* satisfy the rule $\psi_{i,j}$. However, the weights are picked such that the weight of this accepting run over w determines the level of unsafety of w with respect to $\psi_{i,j}$ (see Def. 6.3.5). Second, we combine all the weighted automata into a single automaton $\overline{\mathcal{A}}_{\Phi}$, using its weights to capture the level of unsafety of all words, and thus traces of $\mathcal{T}$, with respect to the given set of safety rules with priorities (Φ, ϖ) (see Def. 6.3.7). Third, we build a weighted product of the transition system $\mathcal{T}$ and the automaton $\overline{\mathcal{A}}_{\Phi}$ (see Def. 6.3.9), and we search for the shortest path from the initial state to a final state. We show that this path projects onto a trace of $\mathcal{T}$ that satisfies the conditions *(i)–(iii)* put forth in the statement of the minimum-violating trace problem (Problem 6.3.4). In the rest of this section, we describe each of these steps in detail in this order.

Let $\mathcal{T} = (S, s_{\text{init}}, T, AP, L, W)$ be our input weighted deterministic transition system, and let $(\Phi = (\Psi_1, \ldots, \Psi_n), \varpi)$ be a set of safety rules with priorities, where $\Psi_i = \{\psi_{i,1}, \ldots, \psi_{i,m_i}\}$. Moreover, let $\mathcal{A}_{\psi_{i,j}} = (Q_{i,j}, q_{\text{init},i,j}, 2^{AP}, \delta_{i,j}, F_{i,j})$ denote a finite automaton capturing the safety rule specification $\psi_{i,j} \in \Psi_i$ over AP.

Augmented Automaton $\overline{\mathcal{A}}_{\psi_{i,j}}$

Definition 6.3.5 (Automaton $\overline{\mathcal{A}}_{\psi_{i,j}}$)**.**
Given an automaton $\mathcal{A}_{\psi_{i,j}}$, a weighted finite automaton

$$\overline{\mathcal{A}}_{\psi_{i,j}} = (\overline{Q}_{i,j}, \overline{q}_{\text{init},i,j}, 2^{AP}, \overline{\delta}_{i,j}, \overline{F}_{i,j}, \overline{W}_{i,j})$$

is built as follows:

- $\overline{Q}_{i,j} = Q_{i,j}$;

- $\overline{q}_{\text{init},i,j} = q_{\text{init},i,j}$;
- $\overline{\delta}_{i,j} = \delta_{i,j} \cup \{(q, \sigma, q) \mid q \in Q \text{ and } \sigma \in \Sigma\}$;
- $\overline{F}_{i,j} = F_{i,j}$; *and*
- $$\overline{W}\big((q, \sigma, q')\big) = \begin{cases} 0 & \text{if } (q, \sigma, q') \in \delta_{i,j} \\ \varpi(\psi_{i,j}) & \text{if } (q, \sigma, q') \in \overline{\delta}_{i,j} \setminus \delta_{i,j}. \end{cases}$$

Lemma 6.3.6. *Let $w = w(0) \dots w(k-1)$ be a word over 2^{AP}. Then w is accepted by $\overline{\mathcal{A}}_{\psi_{i,j}}$ and the minimal weight $W(\rho_{\text{fin}}) = \sum_{j=0}^{k-1} \overline{W}\big((q_j, w(j), q_{j+1})\big)$ of an accepting run $\rho_{\text{fin}} = q_0 \dots q_k$ of $\overline{\mathcal{A}}_{\psi_{i,j}}$ over w is equal to the level of unsafety $\lambda(w, \psi_{i,j})$ from Eq. (6.3).*

Proof. The proof is via induction with respect to the level of unsafety $\lambda(w, \psi_{i,j})$. We prove that for each w there exists an accepting run of $\overline{\mathcal{A}}_{\psi_{i,j}}$ with its weight equal to $\lambda(w, \psi_{i,j})$ and we prove, that no accepting run over w has its weight less than $\lambda(w, \psi_{i,j})$.

First, let $\lambda(w, \psi_{i,j}) = 0$. Then w is accepted by $\mathcal{A}_{\psi_{i,j}}$ and therefore, from the construction of $\overline{\mathcal{A}}_{\psi_{i,j}}$ there exists an accepting run of $\overline{\mathcal{A}}_{\psi_{i,j}}$ over w with its weight equal to 0. At the same time, clearly the weight of no accepting run over w is less than 0.

Second, let $k \in \mathbb{N}$ and let us assume that the lemma holds for each w, such that $\lambda(w, \psi_{i,j}) \leq k \cdot \varpi(\psi_{i,j})$. Consider a word w', such that $\lambda(w', \psi_{i,j}) = (k+1) \cdot \varpi(\psi_{i,j})$. Necessarily, there exist $u, v \in (2^{AP})^*$, and $\sigma \in 2^{AP}$, such that $\lambda(u \cdot v, \psi_{i,j}) = k \cdot \varpi(\psi_{i,j})$ and $w' = u \cdot \sigma \cdot v$. Thus, from the induction assumption, there exists an accepting run $q_0 \dots q_{|u \cdot v|}$ of $\overline{\mathcal{A}}_{\psi_{i,j}}$ over word $u \cdot v$ with its weight equal to $k \cdot \varpi(\psi_{i,j})$. From the construction of $\overline{\mathcal{A}}_{\psi_{i,j}}$ there is a transition $(q_{|u|}, \sigma, q_{|u|}) \in \overline{\delta}_{i,j}$ with $W\big((q_{|u|}, \sigma, q_{|u|})\big) = \varpi(\psi_{i,j})$. Thus, there exists an accepting run over $w' = u \cdot \sigma \cdot v$ with its weight equal to $k \cdot \varpi(\psi_{i,j}) + \varpi(\psi_{i,j}) = (k+1) \cdot \varpi(\psi_{i,j})$.

Now, assume that there exists an accepting run $q_0 q_1 \dots q_{|w'|-1} q_{|w'|}$ over w' with its weight equal to $l \cdot \varpi(\psi_{i,j})$, where $l < k+1$. Then, the run $q_0 \dots q_{|w'|-1} q_{|w'|}$ has the property that $(q_{x_y}, w'(x_y), q_{x_y+1}) \in \overline{\delta}_{i,j} \setminus \delta_{i,j}$ for exactly l different indexes x_y, where $y \in \{1, \dots, l\}$. Hence, the word $\text{vanish}(w', \{x_1, \dots, x_l\}) = w'(0) \dots w'(x_1 - 1) w'(x_1 + 1) \dots w'(x_l - 1) w'(x_l + 1) \dots w'(|w'|)$ is accepted by automaton $\mathcal{A}_{\psi_{i,j}}$ and thus the level of unsafety $\lambda(w', \psi_{i,j}) \leq l$ which contradicts the assumption. The proof is thus complete. □

Weighted Automaton $\overline{\mathcal{A}}_\Phi$

We combine all automata $\overline{\mathcal{A}}_{\psi_{i,j}}$, where $\psi_{i,j} \in \Psi_i \in \Phi$ into a single weighted automaton $\overline{\mathcal{A}}_\Phi$ capturing, through its weight function, the level of unsafety with respect to the whole set of safety rules with priorities (Φ, ϖ).

Definition 6.3.7 (Automaton $\overline{\mathcal{A}}_\Phi$)**.** *The weighted finite automaton*

$$\overline{\mathcal{A}}_\Phi = (\overline{Q}, \overline{q}_{\text{init}}, 2^{AP}, \overline{\delta}, \overline{F}, \overline{W})$$

is defined as follows:

- $\overline{Q} = Q_{1,1} \dots \times \dots Q_{1,m_1} \dots \times \dots Q_{n,1} \dots \times \dots Q_{n,m_n}$*;*
- $\overline{q}_{\text{init}} = (q_{\text{init},1,1}, \dots, q_{\text{init},n,m_n})$*;*
- $(p, \sigma, p') \in \overline{\delta}$ *and* $\overline{W}\big((p, \sigma, p')\big) = (x_1, \dots, x_n)$*, if* $p = (q_{1,1}, \dots, q_{n,m_n}), p' = (q'_{1,1}, \dots, q'_{n,m_n})$*,* $(q_{i,j}, \sigma, q'_{i,j}) \in \overline{\delta}_{i,j}$*, for all* $i \in \{1, \dots, n\}, j \in \{i, \dots m_i\}$*, and* $x_i = \sum_{j=1}^{m_i} \overline{W}_{i,j}(q_{i,j}, \sigma, q'_{i,j})$*.*
- $\overline{F} = \{(q_{1,1}, \dots, q_{n,m_n}) \mid q_{i,j} \in \overline{F}_{i,j},$ *for all* $i \in \{1, \dots, n\}, j \in \{i, \dots m_i\}\}$

Consider a finite run $\rho_{\text{fin}} = q_0 \dots q_k$ of $\overline{\mathcal{A}}_\Psi$ over a word $w_{\text{fin}} = w(0) \dots w(k-1)$ and assume that $W\big((q_j, w(j), q_{j+1})\big) = (x_{1,j}, \dots, x_{n,j})$ denotes the weight of the corresponding transition, for all $j \in \{1, \dots, k-1\}$. With the slight abuse of notation, the weight of the run $\rho_{\text{fin}} = q_0 \dots q_k$ over the word w is

$$W(\rho_{\text{fin}}) = (X_1, \dots, X_n), \text{ where } X_i = \sum_{j=0}^{k-1} x_{i,j}, \text{ for all } i \in \{1, \dots n\}.$$

In other words, the weight of a run is the tuple obtained by component-wise sum of the weights associated with the transitions executed along the run. The *shortest run over* $w(1) \dots w(k-1)$ is the run ρ minimizing the weight $W(\rho_{\text{fin}})$ in the lexicographical ordering.[1]

Lemma 6.3.8. *Let w be a word over 2^{AP}. Then w is accepted by $\overline{\mathcal{A}}_\Phi$ and the weight of the shortest accepting run of $\overline{\mathcal{A}}_\Phi$ over w is equal to the level of unsafety $\lambda(w, \Phi)$ from Eq. (6.5).*

Proof. Consider a word $w \in (2^{AP})^*$. From Lemma 6.3.6, we have that w is accepted by $\overline{\mathcal{A}}_{\psi_{i,j}}$ and the weight of the shortest accepting run $q_{0,i,j} \dots q_{|w|,i,j}$ of $\overline{\mathcal{A}}_{\psi_{i,j}}$ over w is $\lambda(w, \psi_{i,j})$, for all $\psi_{i,j} \in \Phi$. Thus there exists an accepting run

1. The lexicographical ordering $\leq$ is defined as follows: $(x_1, \dots, x_n) \leq (x'_1, \dots, x'_n)$ if $(x_1, \dots, x_n) = (x'_1, \dots, x'_n)$ or if $\exists 1 \leq i \leq n$, such that $\forall j < i.\, x_j \leq x'_j$ and $x_i < x'_i$.

$(q_{0,1,1}, \dots q_{0,n,m_n}) \dots (q_{|w|,1,1}, \dots q_{|w|,n,m_n})$ over w and its weight is $(\sum_{k=1}^{|w|} x_{1,k}, \dots, \sum_{k=1}^{|w|} x_{n,k})$, where $x_{i,k} = \sum_{j=1}^{m_i} \overline{W}_{i,j}(q_{k-1,i,j}, w(k), q_{k,i,j})$, for all $k \in \{1, \dots |w|\}$. On the other hand, each accepting run $\rho = (q_{0,1,1}, \dots q_{0,n,m_n}) \dots (q_{|w|,1,1}, \dots q_{|w|,n,m_n})$ of $\overline{\mathcal{A}}_\Phi$ over w maps to an accepting run of $q_{0,i,j} \dots q_{|w|,i,j}$ of $\overline{\mathcal{A}}_{\psi_{i,j}}$ over w, such that

$$W(\rho) \leq (\textstyle\sum_{k=1}^{|w|} x_{k,1}, \dots, \sum_{k=1}^{|w|} x_{k,n}), \text{ where}$$

$x_{k,i} = \sum_{j=1}^{m_i} \overline{W}_{i,j}(q_{k-1,i,j}, w(k), q_{k,i,j})$, for all $k \in \{1, \dots |w|\}$ and all $\psi_{i,j} \in \Phi$. □

Weighted Product Automaton $\mathcal{P}$

Finally, we build a product automaton that captures the traces of $\mathcal{T}$ satisfying the specification given by $\overline{\mathcal{A}}_\Phi$ while preserving the purpose of the weight function.

Definition 6.3.9 (Product automaton $\mathcal{P}$)**.** *We build the weighted product automaton $\mathcal{P} = \mathcal{T} \otimes \overline{\mathcal{A}}_\Phi = (Q_\mathcal{P}, p_{\text{init}}, \delta_\mathcal{P}, F_\mathcal{P}, W_\mathcal{P})$ as follows:*

- $Q_\mathcal{P} = (S \cup \{init \mid init \notin S\}) \times \overline{Q}$
- $p_{\text{init}} = (init, \overline{q}_{\text{init}})$
- $\big((s,q),(s',q')\big) \in \delta_\mathcal{P}$ *and* $W_\mathcal{P}\big((s,q),(s',q')\big) = (\overline{W}(q, L(s'), q'), W(s,s'))$ *if* $(s,s') \in T$, *and* $(q, L(s'), q') \in \overline{\delta}$
- $\big((init,q),(s_{\text{init}},q')\big) \in \delta_\mathcal{P}$ *and* $W_\mathcal{P}\big((init,q),(s_{\text{init}},q')\big) = (\overline{W}(q, L(s'), q'), 0)$ *if* $(q, L(s'), q') \in \overline{\delta}$
- $F_\mathcal{P} = S \times \overline{F}$

The following lemma summarizes the role of the product automaton $\mathcal{P}$.

Lemma 6.3.10. *The shortest path (in the lexicographical ordering with respect to $W_\mathcal{P}$) $p_0 \dots p_n$ of $\mathcal{P}$ from the state $p_0 = p_{\text{init}}$ to a state $p_n \in F_\mathcal{P}$, such that $\mathsf{B} \in L(p_n)$ projects onto a trace $r = proj(p_0 \dots p_n)$ of $\mathcal{T}$ that is the solution to the minimum-violating trace problem (Problem 6.3.4).*

Proof. The proof follows directly from Lemma 6.3.8 the following two facts. (1) Given a trace $r = s_0 \dots s_n$ of $\mathcal{T}$ that produces a word $w(r)$, such that $s_0 = s_{\text{init}}$ and $\mathsf{B} \in L(s_n)$, there exists an accepting run of $\mathcal{P}$ over $w(r)$. Furthermore, the weight of the shortest accepting run of $\mathcal{P}$ over $w(r)$ is equal to $(\lambda(r, \Phi), W(r))$. (2) Given an accepting run $p_0 \dots p_n = (q_0, s_0) \dots (q_n, s_n)$ of $\mathcal{P}$, such that $p_0 = p_{\text{init}}$, $p_n \in F_\mathcal{P}$, and $\mathsf{B} \in L(p_n)$ with the weight equal to $(\lambda(r, \Phi), W(r))$, the trace

$r = s_0 \dots s_n$ is a trace of $\mathcal{T}$ that produces a word $w(r)$ and $q_0 \dots q_n$ is a run over $w(r)$ with its weight equal to $\lambda(r, \Phi)$). □

Computation of the shortest path in $\mathcal{P}$ can be done with the use of a slight modification of the well-known Dijsktra's shortest path algorithm [Dij59], where all the comparisons in the algorithm are based on the lexicographical ordering of $(n+1)-$tuples.

For convenience, the proposed algorithm that solves Problem 6.3.4 is summarized in Algorithm 15.

Algorithm 15 Solution to Problem 6.3.4

Input: A TS $\mathcal{T} = (S, s_{\text{init}}, T, AP, L, W)$; a region label $\mathsf{B} \in AP$; a set of safety rules with priorities $(\Phi = (\Psi_1, \dots, \Psi_n), \varpi)$ expressed as an FA $\mathcal{A}_{\psi_{i,j}} = (Q_{i,j}, q_{\text{init},i,j}, \Sigma, \delta_{i,j}, F_{i,j})$ for each rule $\psi_{i,j} \in \Psi_i$.
Output: A trace of $\mathcal{T}$ that is solution to Problem 6.3.4
1: **for all** $i \in \{1, \dots, n\}, \psi_{i,j} \in \Psi_i$ **do**
2: build modified automaton $\overline{\mathcal{A}}_{\psi_{i,j}}$ (Def. 6.3.5)
3: **end for**
4: build modified automaton $\overline{\mathcal{A}}_{\Psi}$ (Def. 6.3.7)
5: build product automaton $\mathcal{P} = \mathcal{T} \otimes \overline{\mathcal{A}}_{\Psi}$ (Def. 6.3.9)
6: *path* := shortest_path$(\mathcal{P}, \mathsf{B})$
7: *trace* := project_onto_first_component(*path*)
8: **return** *trace*

Correctness and Complexity

The correctness of the algorithm is stated in the following theorem. The computational complexity of the algorithm analyzed subsequently.

Theorem 6.3.11. *Algorithm 15 returns a solution to the minimum-violating trace problem (Problem 6.3.4).*

Proof. Follows directly from the proof of Lemma 6.3.10 and the correctness of the Dijkstra's shortest path algorithm. □

Let $|\mathcal{T}|$ denote the size of the input transition system, and $|\mathcal{A}_{\psi_{i,j}}|$ the size of the input automaton $\mathcal{A}_{\psi_{i,j}}$. The size of the automaton $\overline{\mathcal{A}}_{\psi_{i,j}}$ is $|\mathcal{A}_{\psi_{i,j}}| \cdot |\Sigma|$ in the worst case and the size of $\overline{\mathcal{A}}_{\Phi}$ is then up to $\prod_{\psi_{i,j} \in \Phi} |\mathcal{A}_{\psi_{i,j}}| \cdot |\Sigma|$. The size of the product automaton $\mathcal{P}$ is $|\mathcal{P}| = |\mathcal{T}| \cdot \prod_{\psi_{i,j} \in \Phi} |\mathcal{A}_{\psi_{i,j}}| \cdot |\Sigma|$ and the modified Dijkstra's algorithm runs in $\mathcal{O}(|\mathcal{P}| \log |\mathcal{P}|)$.

6.3.3 An Alternative Problem Statement

In case a weighted transition system is used as a model, it is questionable, whether the definition of level of unsafety fits our needs as it does not take into consideration the length of the transitions leading into states that need to be vanished from a trace in order to make the trace satisfying. For instance, if a safety rule says "Do not *stay* in the left lane," one expects that the length of the transitions leading into states violating these rules, *i.e.,* distances or travel times spent in the left lane, should impact the level of unsafety of the trace.

Therefore, we introduce an alternative definition for the level of unsafety, taking into account the lengths of the transitions.

Definition 6.3.12 (Weighted level of unsafety)**.** *Weighted level of unsafety* $\lambda_W(r,\psi)$ *of finite trace* r *with respect to a safety rule* ψ *is*

$$\lambda_W(r,\psi) = \min_{I\subseteq\{0,\dots,n\},\,\mathrm{vanish}(r,I)\models\psi} \sum_{i\in I} W\big((s_{i-1},s_i)\big)\cdot\varpi(\psi).$$

The definition of the weighted level of unsafety with respect to a set of safety rules with priorities remains analogous to the one in Def. 6.3.2. The minimum-violating trace problem can be now formulated using the weighted level of unsafety (Def. 6.3.12) instead of "regular" level of unsafety (Def. 6.3.1). The proposed solution can also be easily adapted to the resulting problem. In fact, the only modification lies in the definition of the weight function of the product automaton:

$$W_{\mathcal{P}}\big(((s,q),(s',q'))\big) = \Big(\big(\overline{W}\big((q_1,L(s'),q_1')\big)\cdot W\big((s,s')\big),\dots,\\ \overline{W}\big((q_n,L(s'),q_n')\big)\cdot W\big((s,s')\big)\big), W\big((s,s')\big)\Big),$$

for all $((s,q),(s',q'))\in\delta_{\mathcal{P}}$, where $q=(q_1,\dots,q_n)\in\overline{Q}$.

However, this approach also does not fit all the scenarios. For instance, the safety rule formulated as "Never *enter* the left lane." intuitively should not be influenced by the weights of the transitions. Ideally, we would like to combine the above two approaches and associate each safety rule with an indicator determining whether the "regular" level of unsafety (Def. 6.3.1) or the weighted level of unsafety (Def. 6.3.12) should be used. Therefore, we interconnect both definitions and introduce the combined level of unsafety.

Definition 6.3.13 (Combined level of unsafety)**.** *Combined level of unsafety* $\lambda_C(r,\Psi_i)$ *of finite trace* r *with respect to the set of safety rules* Ψ_i *is*

$$\lambda_C(r,\Psi_i) = \sum_{j\in J_R}\lambda(r,\psi_j) + \sum_{j\in J_W}\lambda_W(r,\psi_j),$$

where $\psi_{i,j}$ such that $j \in J_R \subseteq \{1, \dots m_i\}$ are the safety rules associated with the level of unsafety (Def. 6.3.2) and $\psi_{i,j}$ such that $j \in J_W \subseteq \{1, \dots m_i\}$ are the safety rules associated with the weighted level of unsafety (Def. 6.3.12). Note, that $J_R \cap J_W = \emptyset$ and $J_R \cup J_W = \{1, \dots, |\Psi_i|\}$.

Combined level of unsafety $\lambda(r, \Phi)$ of finite trace r with respect to the set of rules with priorities (Φ, ϖ) is then a tuple

$$\lambda_C(r, \Phi) = \big(\lambda_C(r, \Psi_1), \dots, \lambda_C(r, \Psi_n)\big).$$

Then, the problem differs from Problem 6.3.4 with the change in objective (ii), where we aim to find a trace minimizing the *combined* level of unsafety. To address the new problem, we make only a single modification in the proposed algorithm, which is to change the definition of the weight function of the product

$$W_{\mathcal{P}}\big(((s, q), (s', q'))\big) = \big((x_1, \dots, x_n), W\big((s, s')\big)\big),$$

with the property that

$$x_i = \sum_{j \in J_R} \overline{W}_{i,j}(q_{i,j}, \sigma, q'_{i,j}) + \sum_{j \in J_W} \overline{W}_{i,j}(q_{i,j}, \sigma, q'_{i,j}) \cdot W(s, s'),$$

such that $\mathcal{A}_{i,j}$, where $j \in J_R \subseteq \{1, \dots m_i\}$, are the automata for rules associated with the level of unsafety, and $\mathcal{A}_{i,j}$, where $j \in J_W \subseteq \{1, \dots m_i\}$, are the automata for rules associated with the weighted level of unsafety, for all $i \in \{1, \dots, n\}$. Since such a product automaton is built, the rest of the our algorithm remains the same.

6.3.4 Autonomous Vehicle Example

To demonstrate our approach we have implemented the suggested algorithms and we have run a series of simulations. In particular, this section introduces six different instances of a robotic vehicle in a urban traffic to show the applicability of the proposed solution.

Consider a robotic vehicle in the partitioned environment depicted in Figure 6.3. The robot's motion capabilities are modeled as a transition system. Each state of the transition of the system is given by (i) the cell the robot occupies and (ii) the robot's direction within the cell. For simplicity, only four directions N, E, S, W, *i.e.,* North, East, South and West, respectively, are considered. The simulation environment depicted in Figure 6.3 has 240 cells and 960 corresponding states. Define $Dir = \{N, E, S, W\}$, $Col = \{1, \dots, 20\}$, and $Row = \{1, \dots 12\}$, where Col and Row represent the column number as measured from left to right and the row number as measured from bottom to top in

Figure 6.3, respectively. Then, the set of states of the transition system can be described conveniently as

$$Q = \{q^d_{c,r} \mid c \in Col, r \in Row, d \in Dir\}.$$

For example, in this notation, the current state of the robot, as depicted in Figure 6.3, is $q^E_{2,2}$. Starting from any state, the robot can move into the adjacent cell in front of it by keeping its current direction or into the cell on its left-hand side or its right-hand side by changing it's direction accordingly (if such cells exist in the environment). For instance, when the current state is $q^E_{2,2}$, these transitions would lead into $q^E_{3,2}$, $q^N_{2,3}$ and $q^S_{2,1}$, respectively. The weight of each transition is set to the distance between the adjacent cells, which equals to 1 in this case.

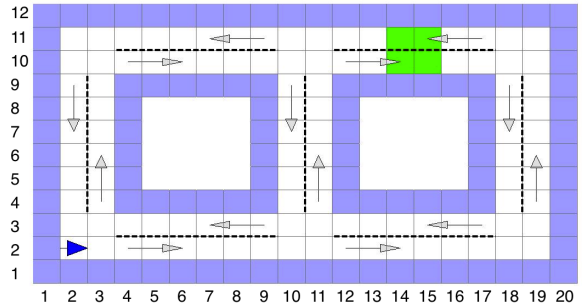


Figure 6.3: A road-like environment partitioned into cells. Each road segment has two lanes and the purple-shaded area illustrates a sidewalk. The robot's position and direction are depicted as the blue arrow. The white arrows in the lane illustrate the direction the vehicle should be heading when present there. Note that the vehicle is allowed to head two different directions in each intersection cell. The green-shaded area is the goal region.

Throughout the case studies, the set of atomic propositions is defined as

$$AP = \{\mathsf{road\_}X, \mathsf{car\_}X, \mathsf{sidewalk}, \mathsf{goal}\},$$

where $X \in \{N, E, S, W\}$. Each state $q^X_{c,r}$ is labeled with proposition $\mathsf{car\_}X$. The states of the TS that correspond to the road cells of the environment where the vehicle should be positioned in direction X are labeled with $\mathsf{road\_}X$. For instance, $\mathsf{road\_}E \in L(q^Y_{2,r})$, for all $r \in \{2, \ldots, 19\}, Y \in Dir$ and similarly $\mathsf{road\_}S \in L(q^Y_{c,2})$, for all $c \in \{2, \ldots, 11\}, Y \in Dir$. The states that correspond to sidewalk cells (in purple) of the TS are labeled with the proposition $\mathsf{sidewalk}$. The states corresponding to the goal region (shown in green in Figure 6.3) are labeled with the proposition *goal*. For instance, the state $q^W_{15,11}$ is labeled with $L(q^W_{15,11}) = \{\mathsf{road\_}W, \mathsf{car\_}W, \mathsf{goal}\}$, the state $q^N_{15,11}$ with $L(q^N_{15,11}) =$

$\{\mathsf{road\_}W, \mathsf{car\_}N, \mathsf{goal}\}$, the state $q^N_{20,1}$ with $L(q^N_{20,1}) = \{\mathsf{sidewalk}, \mathsf{car\_}N\}$, and the state $q^E_{19,2}$ with $L(q^E_{19,2}) = \{\mathsf{road\_}N, \mathsf{road\_}E, \mathsf{car\_}E\}$.

As in Problem 6.3.4, the mission is to drive from the current state into a goal state. At the same time, however, we require the vehicle to follow the rules of the road: (i) always head one of the allowed directions (*direction rule*) and (ii) do not drive on a sidewalk (*sidewalk rule*). This might be not possible if there are some (static) obstacles present on the road. In such a case, we focus on finding a path that is, in some sense as close as possible to obeying the rules, where closeness is defined in terms of level of unsafety discussed in Section 6.3.1. Here, driving on the sidewalk is considered less safe than driving in a lane while heading the incorrect direction. That is, the sidewalk rule belongs to a higher priority class when compared to the direction rule.

Furthermore, we would like to prevent the vehicle from performing dangerous maneuvers. For instance, consider two obstacles present in the right lane close to each other. The vehicle could pass the first obstacle, move back to the right lane and then immediately switch back to the wrong lane again in order to pass the second obstacle. However, rather than that, we expect the vehicle to stay in the wrong lane while passing both of the obstacles. Similar holds for the sidewalk. We specifically set the *dangerous maneuver* as coming back from heading the wrong direction (or from a sidewalk) into the right lane, staying in the right lane for up to 3 more steps and entering the wrong lane (or a sidewalk, respectively) again. Such a maneuver is forbidden with a higher priority as the direction rule (or the sidewalk rule, respectively). However, spending 10 steps heading the incorrect direction (or going on the sidewalk) is considered to be less safe than performing the dangerous maneuver. Therefore, we equip the dangerous maneuver rule with priority 3 and the direction rule (or the sidewalk rule, respectively) with priority 1. The level of unsafety of a trace with a single dangerous maneuver is then 3,6, or 9, depending whether the vehicle spends during the maneuver 1,2, or 3 steps in the right lane, respectively. Note, that we could easily distinguish even between unsafety of different dangerous maneuvers by defining a dangerous maneuver of staying in the right lane for up to 2, or even only 1 step and setting their priority accordingly.

The rules that we describe above can be expressed formally using the syntactically safe FLTL as follows.

- Never be on the sidewalk.

$$\psi_{1,1} = \mathsf{G}(\neg\mathsf{sidewalk})$$

- Do not perform the dangerous sidewalk maneuver. In other words, if

coming back from a sidewalk to a road, stay in a road for at least 4 steps.

$$\begin{aligned}\psi_{1,2} = \mathsf{G}\big(&\mathsf{sidewalk} \wedge \mathsf{X}\,\neg\mathsf{sidewalk} \Rightarrow \\ &\mathsf{X}\,(\neg\mathsf{sidewalk} \wedge \\ &\mathsf{X}\,(\neg\mathsf{sidewalk} \wedge \\ &\mathsf{X}\,(\neg\mathsf{sidewalk} \wedge \\ &\mathsf{X}\,(\neg\mathsf{sidewalk})))))\big)\end{aligned}$$

- Never head the wrong direction. Heading the incorrect direction is formalized as a Boolean combination of atomic propositions $\mathsf{car\_}X \wedge \neg\mathsf{road\_}X$, for some $X \in Dir$, *i.e.,* as the vehicle heading different direction than expected in the current cell. For instance, the vehicle heads the wrong direction in states $q^W_{11,2}$, $q^N_{11,2}$, $q^E_{10,4}$, or $q^S_{11,4}$.

$$\psi_{2,1} = \bigwedge_{X \in Dir} \mathsf{G}\,(\mathsf{car\_}X \Rightarrow \mathsf{road\_}X)$$

- Do not do the dangerous wrong lane maneuver. In other words, if coming back from heading the wrong direction to the right one, stay in the right lane for at least 4 steps.

$$\begin{aligned}\psi_{2,2} = \bigwedge_{X,Y_1,Y_2,Y_3,Y_4 \in Dir} \\ \mathsf{G}\Big(\big(\neg(\mathsf{car\_}X \ \wedge\ \mathsf{road\_}X) \wedge \mathsf{X}\,(\mathsf{car\_}Y_1 \wedge \mathsf{road\_}Y_1)\big) \Rightarrow \\ \mathsf{X}\,(\mathsf{car\_}Y_1 \wedge \mathsf{road\_}Y_1 \wedge \\ \mathsf{X}\,(\mathsf{car\_}Y_2 \wedge \mathsf{road\_}Y_2 \wedge \\ \mathsf{X}\,(\mathsf{car\_}Y_3 \wedge \mathsf{road\_}Y_3 \wedge \\ \mathsf{X}\,(\mathsf{car\_}Y_4 \wedge \mathsf{road\_}Y_4)))))\Big)\end{aligned}$$

Altogether, the set of rules with priorities is

$$(\Psi = (\Psi_1, \Psi_2,), \varpi) = ((\{\psi_{1,1}, \psi_{1,2}\}, \{\psi_{2,1}, \psi_{2,2}\}), \varpi),$$

where $\varpi(\psi_{1,1}) = 1$, $\varpi(\psi_{1,2}) = 3$, $\varpi(\psi_{2,1}) = 1$,$\varpi(\psi_{2,2}) = 3$.

The finite automata corresponding to the rules $\psi_{1,1}$ and $\psi_{1,2}$ are depicted in Figure 6.4 and 6.5, respectively. The finite automata for the rules $\psi_{2,1}$ and $\psi_{2,2}$ are constructed analogously.

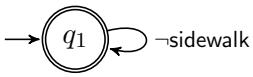


Figure 6.4: Finite automaton $\mathcal{A}_{1,1}$ for the safety rule $\psi_{1,1}$. The transition labeled with ¬sidewalk in fact represents $2^{|AP|-1}$ transitions labeled with all the subsets of atomic propositions that do not contain proposition sidewalk.

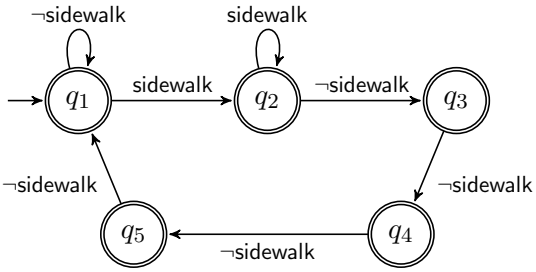


Figure 6.5: Finite automaton $\mathcal{A}_{1,2}$ for the safety rule $\psi_{1,2}$. Each transition labeled with sidewalk in fact represents $2^{|AP|-1}$ transitions labeled with all the subsets of atomic propositions that contain proposition sidewalk and each transition labeled with ¬sidewalk represent $2^{|AP|-1}$ transitions labeled with all the subsets of atomic propositions that do not contain proposition sidewalk.

Simulation Results

Below six different scenarios, each of which feature a different set of obstacles, are considered. In each simulation example, the vehicle is started from the initial state, $q^E_{2,2}$. This condition allows for the overall distance and maneuvering penalties to be compared uniformly across all examples.

A weighted product automaton was built for each of the cases, whose transitions are weighted by 3-tuples. The shortest run of the product automaton (in the lexicographical ordering) projects onto an optimal trace of the transition system. The first two elements of the weight tuple represent the level of unsafety of the trace with respect to Φ and can be viewed also as penalties collected for violation of the both sidewalk and both direction rules, respectively. The third element is the trace weight, *i.e.,* the total distance travelled.

The results for the considered cases are captured in Figure I-VI. The red cells depict the obstacles and the blue arrows illustrate the computed robotic traces in the given environments.

Case I. *No obstacles.*

In this case, there were no obstacles present in this environment. The

algorithm simply picked the trace of shortest distance, which can be either the path shown in Figure I, or a trace that travels in the North lane of the middle S-N road. The weight assigned to either of these paths was W_A=(0,0,20), *i.e.,* the computed trace satisfies all formulas in Φ and its weight, *i.e.,* the total distance traveled is 20.

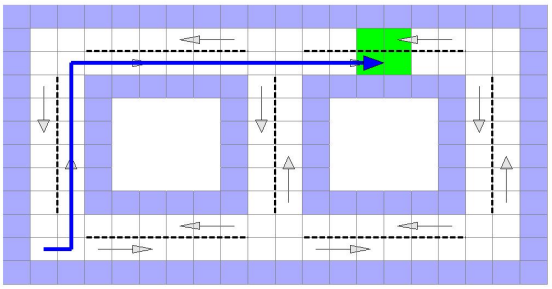


Figure I: Paht for case I.

Case II. *Obstacles present in the right lane in both leftmost and middle S-N roads.*

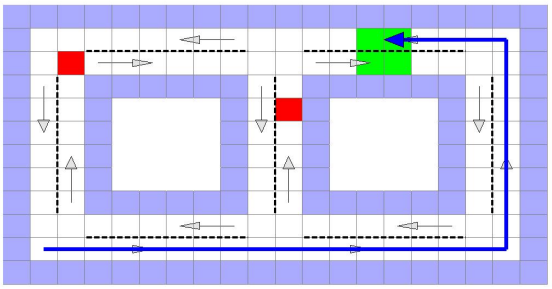


Figure II: Path for case II.

The obstacles present in this environment as illustrated in Figure II removed the options of traveling in the North lane of the middle S-N road and in the North lane of the leftmost S-N road, the shortest paths available for the environment. Considering these restrictions, the algorithm instead chose to travel in the North lane of the rightmost S-N road. This decision was made in lieu of attempting to switch lanes to get around one of the obstacles. While this was not the trace of shortest distance, it was

the path of smallest weight, with W_B=(0,0,30). This path was unique in that there were no others paths whose weight was equivalent.

Case III. *Obstacles present in each S-N road.*

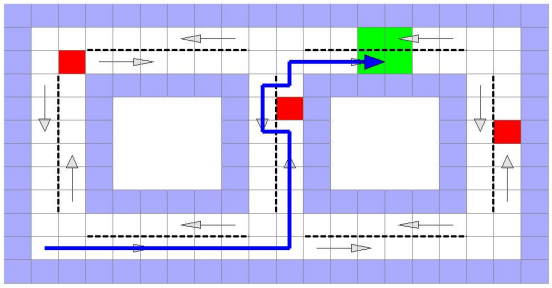


Figure III: Path for case III.

The three obstacles in the environment as illustrated in Figure III effectively blocked all paths which did not incur any sidewalk or direction rules violation. In this case, the algorithm chose a lane switching maneuver with the resulting weight equal to W_C=(0,4,22). The level of unsafety of the computed trace is 0 with respect to the sidewalk rules Ψ_1 and 4 with respect to the wrong direction rules Ψ_2. The four penalties accrued during this path were a result of two lane switches which resulted in incorrect (West and East) orientation in the South and North lanes, respectively, and the two North transitions while in the South lane. By inspection, this type of maneuvering on the rightmost S-N road, would have incurred an identical lane penalty, but would have had a larger weight due to the overall distance metric.

Case IV. *Obstacles totally blocking the leftmost and the middle S-N roads and two obstacles close to each other in the rightmost S-N road.*

In this case, depicted in Figure IV, the algorithm found a trace that involved lane switching and no sidewalks. As expected from the definition of the dangerous maneuver rule $\Psi_{2,2}$ and its priority, the vehicle does not switch lanes in the presence of obstacles in the same lane that are spaced four units of distance or less apart. The path as shown in Figure IV has weight $W_D = (0, 7, 32)$. The seven lane penalties were result of seven transitions with the incorrect orientation in the South lane. In this case, the lane switching does not cause increase in the level of unsafety as the

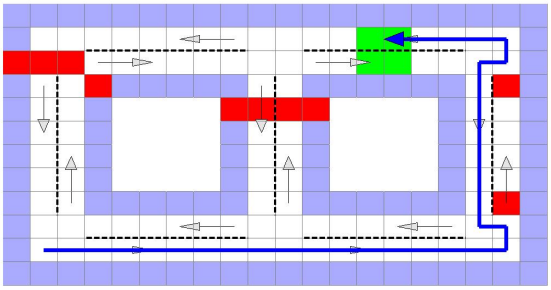


Figure IV: Path or case IV.

lane switching maneuvers happen in the intersection areas, where heading East and West is allowed in North and South lanes, respectively.

Case V. *Sidewalk needed to pass the obstacles.*

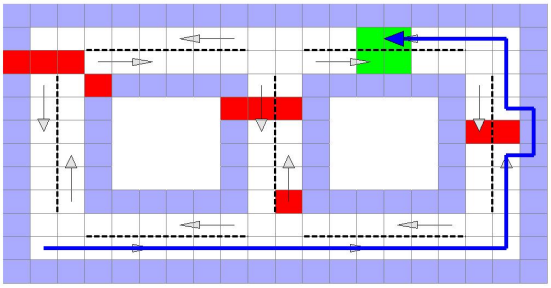


Figure V: Path for case V.

With the addition of extra obstacles in the middle S-N column (see Figure V), the vehicle was unable to finish any path without entering the sidewalks. The rightmost S-N road was chosen, because it only required the vehicle to go on the sidewalk to avoid the obstacles, as opposed to both switching lanes and going on the sidewalk required by the middle S-N road. The weight of the path along rightmost S-N road was W_E=(3,1,32), because the vehicle was on the sidewalk for three units of distance and it was penalized once it returned to the North lane from the sidewalk with the incorrect orientation.

Case VI. *Sidewalk needed to pass the obstacles.*

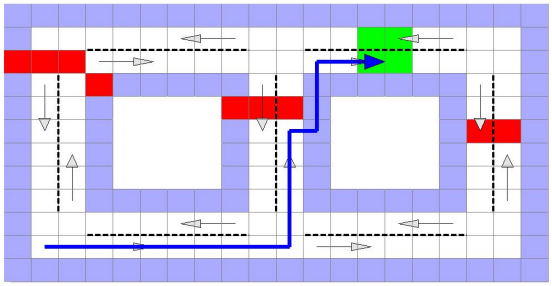


Figure VI: Path or case VI.

Because all roads were blocked by obstacles as illustrated in Figure VI, in this case it was required once again that the vehicle travel on the sidewalk. Instead of returning to the North lane on the middle S-N road, the vehicle chose to re-enter the highway on the uppermost E-W road. The weight of the chosen path, W_F=(3,1,20) was a result of traveling on the sidewalk for three units of distance and entering the uppermost E-W road with the incorrect initial orientation. If the vehicle were to return to the North lane of the middle S-N road instead of directly enter the East lane of the uppermost E-W road, the resulting weight would be $(3, 1, 22)$.

The proposed algorithm was implemented in MATLAB for experimental evaluation on this case study. In each of the scenarios, the product automaton had 24000 states and was built in approx. 8 minutes for each case. The time of the computation of the Dijkstra's shortest path algorithm varied from approx. 1 minute to 20 minutes depending on the total weight of the computed run of the product automaton. The reported times were measured on a Macbook Pro laptop with 2.3 GHz processor and 4GB memory.

6.4 Conclusions

In this chapter, we have studied the least-violating control strategy synthesis problem, *i.e.,* roughly speaking, to find a robot trajectory that satisfies the most important pieces of the specification, when the specification cannot be satisfied as a whole. We have addressed separately planning with long-term tasks and safety rules and for each of these classes we have proposed a metric to distinguish "better" and "worse" traces.

Namely, for planning with long-term tasks, we have considered a set of LTL formulas that are assigned a reward each. Our goal is to maximize the sum of

the rewards collected for the reached formulas.We have proposed an algorithm that provides substantial computational savings when compared to a straightforward solution. We have analyzed the proposed algorithm in terms of correctness, completeness and computational complexity. We have also demonstrated the performance of the proposed algorithm on an illustrative example of a rescue mission.

For planning with safety rules, we have considered formulation, where the mission specification is to reach a goal state while abiding by a given set of rules that are assigned priorities. We have proposed the level of unsafety as a suitable metric to measure how much a particular trace violates a the rules. We have proposed an algorithm that provably violates on the lowest-priority rules for the shortest amount of time. Furthermore, we have evaluated the algorithm in a case study involving a robotic car navigating in an urban environment.

There are many directions for future work. In particular, synthesis of *optimal* strategies that are least violating, and also synthesis of such strategies to be implemented in *dynamic environments* are possible directions for future work. Future work will include also addressing more complex task specifications in the least-violating with road rules part, *i.e.,* specifications other than "reaching a goal state." Examples may include liveness specifications, such as those that describe surveillance missions.

6.5 Notes

The material presented in this chapter is based on joint publications [TRCK$^+$12, THK$^+$12] with Sertac Karaman, Emilio Frazzoli, Daniela Rus and also Luis I. Reyes-Castro and Gavin C. Hall, respectively. The author of this thesis has been the leading author of these papers. Luis C. Reyes is the author of the implementation of the case study for the first considered problem, and Gavin C. Hall is the major author of the implementation of the algorithm for the second problem.

Chapter 7

Conclusions

This thesis has aimed, in general, at the development and the improvement of computer-aided techniques that help us to design provably correct software and hardware systems. We have focused on two major areas: *formal verification*, where the goal is to prove or disprove that a system satisfies given requirements and *control strategy synthesis*, where the aim is to control a given system in a way guaranteeing that the specified requirements are met. This thesis has addressed both the theoretical and the applied aspects of model checking and control strategy synthesis. On one hand, we have designed general-purpose algorithms to offer a wider possibility to verify and control novel classes of models with quantitative features, that have not been addressed before. On the other hand, we have focused on the development of algorithms specifically tailored to match the need of verification and provably correct optimal control of real-world systems and robotic systems in particular.

The first part of the thesis has been devoted to the development of a theoretical framework for *quantitative model checking of systems with degradation*. Systems with degradation are those that naturally incorporate a decaying quality, such as electronic devices with degrading electric charge, broadcasting networks with decreasing power or quality of a transmitted signal.

We have proposed a modeling formalism (Transition System with Degradation, or TSD, for short) that involves a possibility to capture the level of degradation in between two system states and we suggest a specification formalism called Linear Temporal Logic with Degradation Constraints, or DLTL, for short, to capture complex system properties, including requirements on the level of degradation. As the newly proposed DLTL has some resemblance to metric interval time logic (MITL) that is designed for timed systems, we have investigated their relation. We have shown that the translation of DLTL verification problem for systems with degradation into previously solved MITL verification problem for timed systems is possible, and the targeted DLTL model checking problem can be this way solved with limited, yet arbitrary, precision. Furthermore, we have proposed an extension to Büchi automata, called Büchi automata with degradation constraints and we have developed a model checking

algorithm leveraging ideas from automata-theoretic approach to model checking of "standard" transition systems with respect to "standard" Büchi automata. For a specific subclass of DLTL formulas, we ave presented an algorithm for the direct translation of such DLTL formulas into Büchi automata with degradation constraints and thus, for this subclass we have provided a full-precision model checking method.

The suggested framework can be used not only to model check systems with degradation, but also to find control strategies for them. A slight modification of the presented algorithm can be used to address problems such as the following. Consider an autonomous robot that moves in regions of a partitioned environment, collects the data in data gather states and brings them to data upload states. The collected data are stored in a volatile memory and are subject to degradation. The task is to find a path of the robot, such that both data gather and data upload states are periodically visited and a data upload state is always visited before the integrity of the collected data drops bellow a certain threshold.

Next to the development of general-purpose formal verification frameworks, the second part of the thesis has aimed at *automatic optimal control strategy synthesis specifically tailored for robotic systems.* We have focused on the development of high-level controllers for autonomous robotic vehicles given a specification of their complex, long-term mission. In particular, we have considered an autonomous robot traversing regions of a partitioned environment (modeled as a finite state transition system) and a linear temporal logic mission, such as "Periodically visit a data gather state and a data upload state. Never enter a dangerous region. Whenever data are gathered, do not gather again unless they are uploaded," to be reached.

The first problem that we have considered is to guarantee the accomplishment of the given mission while minimizing the time in between two successive visits to a certain set of places, *e.g.,* the data gather states. We have presented an algorithm leveraging ideas from automata-based approach to model checking that automatically generates infinite robotic path in the environment satisfying both desired objectives and we have demonstrated the algorithm in an experimental testbed. Second, we have investigated situations, when a robot is given a complex mission that cannot be satisfied as a whole, *e.g.,* due to unrealizability of the specification or environmental constraints. Our goal is to reach a compromise and to violate as little low-priority subtasks of the mission as possible. We have aimed at two different categories in robotic mission; the long-term goals and the safety rules. For each of these, we have defined a metric to distinguish which robot paths are more violating and which are less. Using formal methods, we generate an optimal *i.e.,* least-violating robot path with respect to this metric. We have demonstrated the approaches on several illustrative case studies.

Bibliography

[ABK12] Shaull Almagor, Udi Boker, and Orna Kupferman. Formalizing quality of reactive systems. January 2012. Unpublished, retrieved from `http://www.cs.huji.ac.il/~shaull/pub/icalp12.pdf`.

[ACD93] Rajeev Alur, Costas Courcoubetis, and David Dill. Model-checking in dense real-time. *Information and Computation*, 104:2–34, 1993.

[AD90] Rajeev Alur and David L. Dill. Automata for modeling real-time systems. In *International Colloquium on Automata, Languages and Programming (ICALP)*, pages 322–335. Springer-Verlag New York, Inc., 1990.

[AD94] Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[AETP01] Rajeev Alur, Kousha Etessami, Salvatore La Torre, and Doron Peled. Parametric temporal logic for "model measuring". *ACM Transactions on Computational Logic*, 2(3):388–407, 2001.

[AFH96] Rajeev Alur, Tomás Feder, and Thomas A. Henzinger. The benefits of relaxing punctuality. *Journal of the ACM*, 43:116–146, 1996.

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

[AM95] Marco Antoniotti and Bud Mishra. Discrete event models + temporal logic = supervisory controller: Automatic synthesis of locomotion controllers. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1441–1446, 1995.

[Asi50] Isaac Asimov. *I, Robot*. Gnome Press, 1950.

[ASSB00] Adnan Aziz, Kumud Sanwal, Vigyan Singhal, and Robert Brayton. Model-checking continuous-time Markov chains. *ACM Transactions on Computational Logic*, 1(1):162–170, July 2000.

[BBČ+07] Jiří Barnat, Luboš Brim, Ivana Černá, Milan Češka, and Jana Tůmová. ProbDiVinE: A parallel qualitative LTL model checker. In *International Conference on Quantitative Evaluation of Systems (QEST)*, pages 215–216. IEEE Computer Society, 2007.

[BBČ+08] Jiří Barnat, Luboš Brim, Ivana Černá, Milan Češka, and Jana Tůmová. ProbDiVinE-MC: Multi-core LTL model checker for probabilistic systems. In *International Conference on Quantitative Evaluation of Systems (QEST)*, pages 77–78. IEEE Computer Society, 2008.

[BBČ+09] Jiří Barnat, Luboš Brim, Ivana Černá, Milan Češka, and Jana Tůmová. Local quantitative LTL model checking. In *Formal Methods for Industrial Critical Systems (FMICS)*, pages 53–68. Springer-Verlag, 2009.

[BBR09] Jiří Barnat, Luboš Brim, and Petr Ročkai. DiVinE 2.0: High-performance model checking. In *High Performance Computational Systems Biology (HIBI)*, pages 31–32. IEEE Computer Society, 2009.

[BČT09] Jiří Barnat, Ivana Černá, and Jana Tůmová. Quantitative model checking of systems with degradation. In *International Conference on Quantitative Evaluation of Systems (QEST)*, pages 21 –30, 2009.

[BČT12a] Jiří Barnat, Ivana Černá, and Jana Tůmová. Timed automata approach to verification of systems with degradation. In *Annual Doctoral Workshop on Mathematical and Engineering Methods in Computer Science (MEMICS)*, volume LNCS 7119, pages 86–95, 2012.

[BČT12b] Jiří Barnat, Ivana Černá, and Jana Tůmová. Verification of systems with degradation. *Computing and Informatics*, 31(3):507–530, 2012.

[BDL04] Gerd Behrmann, Alexandre David, and Kim G. Larsen. A tutorial on UPPAAL. In *Formal Methods for the Design of Real-Time Systems*, number 3185 in LNCS, pages 200–236. Springer–Verlag, 2004.

[BDL12] Benedikt Bollig, Normann Decker, and Martin Leucker. Frequency linear-time temporal logic. In *International Symposium on The-*

oretical Aspects of Software Engineering (TASE), pages 85–92. IEEE Computer Society, 2012.

[BEGL99] Francesco Buccafurri, Thomas Eiter, Georg Gottlob, and Nicola Leone. Enhancing model checking in verification by AI techniques. *Artificial Intelligence*, 112(1-2):57 – 104, 1999.

[BGK+11] Ezio Bartocci, Radu Grosu, Panagiotis Katsaros, C. R. Ramakrishnan, and Scott A. Smolka. Model repair for probabilistic systems. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pages 326–340. Springer-Verlag, 2011.

[BGL+04] Christel Baier, Marcus Größer, Martin Leucker, Benedikt Bollig, and Frank Ciesinski. Controller synthesis for probabilistic systems. In *IFIP International Conference on Theoretical Computer Science (IFIP TCS)*. Kluwer, 2004.

[BIP05] Calin Belta, Volkan Isler, and George J. Pappas. Discrete abstractions for robot motion planning and control in polygonal environment. *IEEE Transactions on Robotics*, 21(5):864–875, 2005.

[BJP+12] Roderick Bloem, Barbara Jobstmann, Nir Piterman, Amir Pnueli, and Yaniv Sa'ar. Synthesis of reactive(1) designs. *Journal of Computer and System Sciences*, 78(3):911 – 938, 2012. In Commemoration of Amir Pnueli.

[BK08] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. MIT Press, 2008.

[BKV10] Amit Bhatia, Lydia E. Kavraki, and Moshe Y. Vardi. Sampling-based motion planning with temporal goals. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2689–2696, 2010.

[Büc62] Julius R. Büchi. On a decision method in restricted second-order arithmetic. In *International Congress on Logic, Mehtodology and Pilosophy of Science*, pages 1–11, 1962.

[CB06] Frank Ciesinski and Christel Baier. Liquor: A tool for qualitative and quantitative linear time analysis of reactive systems. In *International Conference on Quantitative Evaluation of Systems (QEST)*, pages 131–132. IEEE Computer Society, 2006.

[CE81] Edmund M. Clarke and Allen E. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Logics of Programs*, volume LNCS 131, pages 52–71. Springer-Verlag, 1981.

[CRST08] Allesandro Cimatti, Marco Roveri, Vikgor Schuppan, and Andrei Tchaltsev. Diagnostic information for realizability. In *International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI)*, pages 52–67. Springer-Verlag, 2008.

[CTB12] Yushan Chen, Jana Tůmová, and Calin Belta. LTL robot motion control based on automata learning of environmental dynamics. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 5177–5182, 2012.

[CTUB13] Yushan Chen, Jana Tůmová, Alphan Ulusoy, and Calin Belta. Temporal logic robot control based on automata learning of environmental dynamics. *International Journal of Robotics Research*, 2013. To appear.

[CVWY92a] Constantin Courcoubetis, Moshe Y. Vardi, Pierre Wolper, and Mihalis Yannakakis. Memory-efficient algorithms for the verification of temporal properties. *Formal Methods in System Design*, 1(2/3):275–288, 1992.

[CVWY92b] Costas Courcoubetis, Moshe Y. Vardi, Pierre Wolper, and Mihalis Yannakakis. Memory-efficient algorithms for the verification of temporal properties. *Formal Methods in System Design*, 1(2-3):275–288, 1992.

[CY98] Costas Courcoubetis and Mihalis Yannakakis. Markov decision processes and regular events. *IEEE Transactions on Automatic Control*, 43(10):1399–1418, 1998.

[dAFH+05] Luca de Alfaro, Marco Faella, Thomas A. Henzinger, Rupak Majumdar, and Mariëlle Stoelinga. Model checking discounted temporal properties. *Theoretical Computer Science*, 345(1):139–170, 2005.

[dAFS04] Luca de Alfaro, Marco Faella, and Mariëlle Stoelinga. Linear and branching metrics for quantitative transition systems. In *International Colloquium on Automata, Languages and Programming (ICALP)*, pages 97–109, 2004.

[dar13] DARPA Grand Challenge 2007 webpage. `http://archive.darpa.mil/grandchallenge/`, online, January 2013.

[DBC10] Xu Chu Ding, Calin Belta, and Christos G. Cassandras. Receding horizon surveillance with temporal logic specifications. In *IEEE Conference on Decision and Control (CDC)*, pages 256 –261, 2010.

[DF11] Werner Damm and Bernd Finkbeiner. Does it pay to extend the perimeter of a world model? In *International Symposium on Formal Mehods (FM)*, pages 12–26. Springer-Verlag, 2011.

[Dij59] Edsger. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[DLB12] Xu Chu Ding, Mircea Lazar, and Calin Belta. Receding horizon temporal logic control for finite deterministic systems. In *American Control Conference (ACC)*, pages 715 –720, 2012.

[DMB+12] Alexandre Donzé, Oded Maler, Ezio Bartocci, Dejan Nickovic, Radu Grosu, and Scott A. Smolka. On temporal logic and signal processing. In *International Symposium on Automated Technology for Verification and Analysis (ATVA)*, pages 92–106, 2012.

[DSBR11] Xu Chu Ding, Stephen L. Smith, Calin Belta, and Daniela Rus. Mdp optimal control under temporal logic constraints. In *Proceedings of the IEEE Conference on Decision and Control and European Control Conference (CDC-ECC)*, pages 532 –538, 2011.

[Fai11] Georgios E. Fainekos. Revising temporal logic specifications for motion planning. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 40–45, 2011.

[FFK+01] Kathi Fisler, Ranan Fraer, Gila Kamhi, Moshe Y. Vardi, and Zijiang. Is there a best symbolic cycle-detection algorithm? In *Tools and Algorithms for Construction and Analysis of Systems (TACAS)*, volume LNCS 2031, pages 420–434. Springer-Verlag, 2001.

[FGKGP09] Georgios. E. Fainekos, Antoine Girard, Hadas Kress-Gazit, and George J. Pappas. Temporal logic motion planning for dynamic robots. *Automatica*, 45(2):343–352, 2009.

[FJKG11] Cameron Finucane, Gangyuan Jing, and Hadas Kress-Gazit. Designing reactive robot controllers with LTLMoP. In *Automated Action Planning for Autonomous Mobile Robots*, 2011.

[FLS08] Marco Faella, Axel Legay, and Mariëlle Stoelinga. Model checking quantitative linear time logic. *Electronic Notes in Theoretical Computer Science*, 220(3):61–77, 2008.

[GO01] Paul Gastin and Denis Oddoux. Fast LTL to Büchi automata translation. In *International Conference on Computer Aided Verification (CAV)*, volume LNCS 2102, pages 53–65. Springer-Verlag, 2001.

[GO13] Paul Gastin and Denis Oddoux. LTL2BA tool. `http://www.lsv.ens-cachan.fr/~gastin/ltl2ba/`, online, January 2013.

[goo13] Google official blog: The self-driving car logs more miles on new wheels. `http://googleblog.blogspot.hu/2012/08/the-self-driving-car-logs-more-miles-on.html`, online, January 2013.

[GP02] Elsa L. Gunter and Doron Peled. Temporal debugging for concurrent systems. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pages 431–444. Springer-Verlag, 2002.

[GPVW96] Rob Gerth, Doron Peled, Moshe Y. Vardi, and Pierre Wolper. Simple on-the-fly automatic verification of linear temporal logic. In *International Symposium on Protocol Specification, Testing and Verification*, pages 3–18. Chapman & Hall, Ltd., 1996.

[Hau12] Kris Hauser. The minimum constraint removal problem with three robotics applications. In *Proceedings of the International Workshop on the Algorithmic Foundations of Robotics (WAFR)*, 2012.

[HJ89] Hans Hansson and Bengt Jonsson. A Framework for Reasoning about Time and Reliability. In *IEEE Real-Time Systems Symposium*, pages 102–111, 1989.

[HKNP06] Andrew Hinton, Marta Kwiatkowska, Gethin Norman, and David Parker. PRISM: A tool for automatic verification of probabilistic systems. In *Tools and Algorithms for Construction and Analysis of*

Systems (TACAS), volume LNCS 3920, pages 441–444. Springer, 2006.

[Hol97] Gerard J. Holzmann. The model checker SPIN. *IEEE Transactions on Software Engineering*, 25(5):279–295, 1997.

[Hol03] Gerard J. Holzmann. *The Spin Model Checker: Primer and Reference Manual*. Addison-Wesley Professional, 2003.

[KB08] Marius Kloetzer and Calin Belta. A fully automated framework for control of linear systems from temporal logic specifications. *IEEE Transactions on Automatic Control*, 53(1):287 –297, 2008.

[KF08a] Sertac Karaman and Emilio Frazzoli. Complex mission optimization for multiple-uavs using linear temporal logic. In *American Control Conference (ACC)*, pages 2003–2009, 2008.

[KF08b] Sertac Karaman and Emilio Frazzoli. Vehicle routing problem with metric temporal logic specifications. In *IEEE Conference on Decision and Control (CDC)*, pages 3953–3958, 2008.

[KF09] Sertac Karaman and Emilio Frazzoli. Sampling-based motion planning with deterministic μ-calculus specifications. In *IEEE Conference on Decision and Control (CDC)*, pages 2222–2229, 2009.

[KF12a] Sertac Karaman and Emilio Frazzoli. Sampling-based optimal motion planning with deterministic μ-calculus specifications. In *American Control Conference (ACC)*, pages 735–742, 2012.

[KF12b] Kangjin Kim and Georgios Fainekos. Approximate solutions for the minimal revision problem of specification automata. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 265–271, 2012.

[KFS12] Kangjin Kim, Georgios Fainekos, and Sriram Sankaranarayanan. On the revision problem of specification automata. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 5171–5176, 2012.

[KGFP09] Hadas Kress-Gazit, Georgios E. Fainekos, and George J. Pappas. Temporal logic-based reactive mission and motion planning. *IEEE Transactions on Robotics*, 25(6):1370–1381, 2009.

[kiv13] Kiva systems webpage.
`http://www.kivasystems.com/`, online, January 2013.

[Koy90] R. Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Systems*, 2(4):255–299, 1990.

[Koz83] Dexter Kozen. Results on the propositional mu-calculus. *Theoretical Computer Science*, 27:333–354, 1983.

[KRKF09] Sertac Karaman, Steven Rasmussen, Derek Kingston, and Emilio Frazzoli. Specification and planning of UAV missions: a process algebra approach. In *American Control Conference (ACC)*, pages 1442–1447, 2009.

[KV05] Orna Kupferman and Moshe Y. Vardi. Safraless decision procedures. In *IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 531 – 540, 2005.

[KV07] Bernhard H. Korte and Jens Vygen. *Combinatorial Optimization: Theory and Algorithms*, volume 21 of *Algorithmics and Combinatorics*. Springer-Verlag, 4 edition, 2007.

[KYV01] Orna Kupferman and Moshe Y. Vardi. Model checking of safety properties. *Formal Methods in System Design*, 19(3):291–314, 2001.

[KZH+11] Joost-Pieter Katoen, Ivan S. Zapreev, Ernst Moritz Hahn, Holger Hermanns, and David N. Jansen. The ins and outs of the probabilistic model checker MRMC. *Performance Evaluation*, 68(2):90 – 104, 2011.

[Lap09] Gilbert Laporte. Fifty years of vehicle routing. *Transportation Science*, 43(4):408–416, 2009.

[LaV06] Steven M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006.

[LK04] Savvas G. Loizou and Kostas J. Kyriakopoulos. Automatic synthesis of multiagent motion tasks based on LTL specifications. In *IEEE Conference on Decision and Control (CDC)*, pages 153–158, 2004.

[LMP10] Franjois Laroussinie, Antoine Meyer, and Eudes Petonnet. Counting LTL. In *International Symposium on Temporal Representation and Reasoning (TIME)*, pages 51 –58, 2010.

[LWAB10] Morteza Lahijanian, Joe Wasniewski, Sean B. Andersson, and Calin Belta. Motion planning and control from temporal logic specifications with probabilistic satisfaction guarantees. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3227 –3232, 2010.

[MN04] Oded Maler and Dejan Nickovic. Monitoring temporal properties of continuous signals. In *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems (FORMATS)*, volume LNCS 3253, pages 152–166. Springer, 2004.

[MP95] Zohar Manna and Amir Pnueli. *Temporal Verification of Reactive Systems: Safety*. Springer, 1995.

[Pnu77] Amir Pnueli. The temporal logic of programs. In *Annual IEEE Symposium on Foundations of Computer Science (SFCS)*, pages 46–57, 1977.

[QBIZ04] Michael M. Quottrup, Thomas Bak, and Roozbeh Izadi-Zamanabadi. Multi-robot motion planning: A timed automata approach. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 4417–4422, 2004.

[QS82] Jean-Pierre Queille and Joseph Sifakis. Specification and verification of concurrent systems in CESAR. In *Colloquium on International Symposium on Programming*, pages 337–351, 1982.

[Rab69] Michael O. Rabin. Decidability of Second-Order Theories and Automata on Infinite Trees. *Transactions of The American Mathematical Society*, 141, 1969.

[RKG11] Vasumathi Raman and Hadas Kress-Gazit. Analyzing unsynthesizable specifications for high-level robot behavior using ltlmop. In *International Conference on Computer Aided Verification (CAV)*, pages 663–668, 2011.

[RKG12] Vasumathi Raman and Hadas Kress-Gazit. Automated feedback for unachievable high-level robot behaviors. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 5156–5162, 2012.

[Sip12] Michael Sipser. *Introduction to the Theory of Computation*. Course Technology, 3rd edition, 2012.

[STBČ12] Mária Svoreňová, Jana Tůmová, Jiří Barnat, and Ivana Černá. Attraction-based receding horizon path planning with temporal logic constraints. In *IEEE Conference on Decision and Control (CDC)*, pages 6749–6754, 2012.

[STBR10] Stephen L. Smith, Jana Tůmová, Calin Belta, and Daniela Rus. Optimal path planning under temporal logic constraints. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3288–3293, 2010.

[STBR11] Stephen. L. Smith, Jana Tůmová, Calin Belta, and Daniela Rus. Optimal path planning for surveillance with temporal logic constraints. *International Journal of Robotics Research*, 30(14):1695–1708, 2011.

[THK+12] Jana Tůmová, Gavin C. Hall, Sertac Karaman, Emilio Frazzoli, and Daniela Rus. Least-violating control strategy synthesis with safety rules. In *Hybrid systems: Computation and Control (HSCC)*, 2012. To appear.

[TRCK+12] Jana Tůmová, Luis I. Reyes-Castro, Sertac Karaman, Emilio Frazzoli, and Daniela Rus. Minimum-violating planning with conflicting specifications. 2012. Submitted.

[Tům] Jana Tůmová. Verification of probabilistic systems against quantified linear properties. Master thesis. Masaryk University, 2009.

[TV01] Paolo Toth and Daniele Vigo, editors. *The Vehicle Routing Problem*. Monographs on Discrete Mathematics and Applications. SIAM, 2001.

[TYB+10] Jana Tůmová, Boyan Yordanov, Calin Belta, Ivana Černá, and Jiří Barnat. A symbolic approach to controlling piecewise affine systems. In *IEEE Conference on Decision and Control (CDC)*, pages 4230 –4235, 2010.

[USD+11] A. Ulusoy, S. L. Smith, X. C. Ding, C. Belta, and D. Rus. Optimal multi-robot path planning with temporal logic constraints. In *Prob IROS*, San Francisco, US, 2011.

[USDB12] Alphan Ulusoy, Stephen L. Smith, Xu Chu Ding, and Calin Belta. Robust multi-robot optimal path planning with temporal logic constraints. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 4693–4698, 2012.

[VW83] Moshe Y. Vardi and Pierre Wolper. Reasoning about infinite computation paths. *IEEE Symposium on Foundation of Computer Science*, pages 185–194, 1983.

[VW86] Moshe Y. Vardi and Pierre Wolper. An automata-theoretic approach to automatic program verification. In *Logic in Computer Science*, pages 322–331, 1986.

[WTM09] Tichakorn Wongpiromsarn, Ufuk Topcu, and Richard M. Murray. Receding horizon temporal logic planning for dynamical systems. In *IEEE Conference on Decision and Control and the Chinese Control Conference (CDC/CCC)*, pages 5997 –6004, 2009.

[WTM10] Tichakorn Wongpiromsarn, Ufuk Topcu, and Richard M. Murray. Receding horizon control for temporal logic specifications. In *Hybrid systems: Computation and Control (HSCC)*, pages 101–110, 2010.

[WTO+11] Tichakorn Wongpiromsarn, Ufuk Topcu, Necmiye Ozay, Huan Xu, and Richard M. Murray. TuLiP: a software toolbox for receding horizon temporal logic planning. In *Hybrid systems: Computation and Control (HSCC)*, pages 313–314. ACM, 2011.

[Yov97] Sergio Yovine. Kronos: A verification tool for real-time systems. *International Journal on Software Tools for Technology Transfer*, 1:123–133, 1997.

[YTB+10] Boyan Yordanov, Jana Tůmová, Calin Belta, Ivana Černá, and Jiří Barnat. Formal analysis of piecewise affine systems through formula-guided refinement. In *IEEE Conference on Decision and Control (CDC)*, pages 5899–5904, 2010.

[YTČ+12] Boyan Yordanov, Jana Tůmová, Ivana Černá, Jiří Barnat, and C. Belta. Temporal logic control of discrete-time piecewise affine systems. *IEEE Transactions on Automatic Control*, 57(6):1491 – 1504, 2012.

[YTČ+13] Boyan Yordanov, Jana Tůmová, Ivana Černá, Jiří Barnat, and Calin Belta. Formal analysis of piecewise affine systems through formula-guided refinement. *Automatica*, 49(1):261 – 266, 2013.

Appendix A

Proof of BADC Normalization Correctness

Lemma A.0.1. *Assertions on lines 16, 17 in Algorithm 2 hold.*

Proof. Follows directly from lines 6 and 7. □

Lemma A.0.2. *Assertion on line 14 in Algorithm 2 holds.*

Proof. Consider $\mathcal{B} = (Q, Q_{\text{init}}, \Sigma, X, \delta, F)$ and $\mathcal{B}' = (Q', Q'_{\text{init}}, \Sigma, X', \delta', F')$ on line 14 and their corresponding transition systems $\mathcal{S}(\mathcal{B}) = (S, S_{\text{init}}, Act, T)$ and $\mathcal{S}(\mathcal{B}') = (S', S'_{\text{init}}, Act', T')$, respectively. Let $x \in X^{in}$ and $x_{11}, \ldots, x_{1m_1}, \ldots, x_{n1}, \ldots, x_{nm_n}$ be the new degradation variables added to X on line 6 during the iteration of cycle 1-15 for variable x. We prove that for each run $\rho' = (q_0, \nu'_0)(q_1, \nu'_1)\ldots$ of $\mathcal{B}'$ over word $w = (\sigma_0, d_0)(\sigma_1, d_1)\ldots$ there is a run $\rho = (q_0, \nu_0)(q_1, \nu_1)\ldots$ of $\mathcal{B}$ over w and vice versa.

First, we show that for each $i \geq 0$ such that $\nu_i(x_{11}) = \ldots = \nu_i(x_{1m_1}) = \ldots = \nu_i(x_{n1}) = \ldots \nu_i(x_{nm_n}) = \nu'_i(x)$ it holds

- $\big((q_i, \nu'_i), (\sigma_i, d_i), (q_{i+1}, \nu'_{i+1})\big) \in T' \Leftrightarrow \big((q_i, \nu_i), (\sigma_i, d_i), (q_{i+1}, \nu_{i+1})\big) \in T$,
- $\nu_{i+1}(x_{11}) = \ldots = \nu_{i+1}(x_{nm_n}) = \nu'_{i+1}(x)$.

Clearly, $\nu_0(x_{11}) = \ldots = \nu_0(x_{nm_n}) = \nu'_0(x) = 1$. Let us assume that $\nu_i(x_{11}) = \ldots = \nu_i(x_{nm_n}) = \nu'_i(x)$ for each $i \geq 0$ and let $\big((q_i, \nu'_i), (\sigma_i, d_i), (q_{i+1}, \nu'_{i+1})\big) \in T'$ via $\tau'_i = (q_i, \sigma_i, \gamma', R', q_{i+1}) \in \delta'$. It holds that $\tau'_i \in \delta$ on line 2. Let $\tau_i = (q_i, \sigma_i, \gamma, R, q_{q+1}) \in \delta$ be the transition τ'_i on line 14, i.e. after γ' is possibly changed on line 7 into γ and R' is possibly changed on line 11 into R. Note that ν'_i satisfies γ' iff ν_i satisfies γ, and $x \in R' \Leftrightarrow \{x_{11}, \ldots, x_{nm_n}\} \in R$. Thus $\big((q_i, \nu_i), (\sigma_i, d_i), (q_{i+1}, \nu_{i+1})\big) \in T$ via τ_i and $\nu_{i+1}(x_{11}) = \ldots = \nu_{i+1}(x_{nm_n}) = \nu'_{i+1}(x)$.

Dually, let us assume that for $i \geq 0$ it holds $\nu_i(x_{11}) = \ldots \nu_i(x_{nm_n}) = \nu'_i(x)$ and let $\big((q_i, \nu_i), (\sigma_i, d_i), (q_{i+1}, \nu_{i+1})\big) \in T$ via $\tau_i = (q_i, \sigma_i, \gamma, R, q_{q+1}) \in \delta$. Let $\tau'_i = (q_i, \sigma_i, \gamma', R', q_{i+1}) \in \delta'$ be the transition τ_i before γ is possibly changed on line 7 and R is possibly changed on line 11. Because ν'_i satisfies γ' iff ν_i satisfies γ, and $x \in R'$ iff $\{x_{11}, \ldots, x_{nm_n}\} \in R$, then $(q_i, \sigma_i, \gamma', R', q_{i+1}) \in \delta'$ via τ'_i and $\nu_{i+1}(x_{11}) = \ldots = \nu_{i+1}(x_{nm_n}) = \nu'_{i+1}(x)$.

Altogether we have that for each run $\rho' = (q_0, \nu'_0)(q_1, \nu'_1)\dots$ of $\mathcal{B}'$ over word $w = (\sigma_0, d_0)(\sigma_1, d_1)\dots$ there is a run $\rho = (q_0, \nu_0)(q_1, \nu_1)\dots$ of $\mathcal{B}$ over w and vice versa. Furthermore note that if ρ' is an accepting run then ρ is an accepting run as well. □

Lemma A.0.3. *Assertion on line 18 in Algorithm 2 holds.*

Proof. Follows directly from Lemma A.0.2. □

Consider a BADC $\mathcal{B} = (Q, Q_{\text{init}}, \Sigma, X, \delta, F)$ and its corresponding TS $\mathcal{S}(\mathcal{B}) = (S, S_{\text{init}}, Act, T)$. An *extended run* of $\mathcal{B}$ over word $w = (\sigma_0, d_0)(\sigma_1, d_1)\dots$ is an infinite sequence $\rho_E = (q_0, \nu_0)\tau_0(q_1, \nu_1)\tau_1\dots$, such that $q_0 \in Q_{init}$, $\nu_0(x) = 1$ for each $x \in X$, and for each $i \geq 0$ it holds that $\big((q_i, \nu_i), (\sigma_i, d_i), (q_{i+1}, \nu_{i+1})\big) \in T$ *via transition* τ_i, *i.e.,* such that $\tau_i = (q_i, \sigma_i, \gamma_i, R_i, q_{i+1})$, ν_i satisfies γ_i, and $\nu_{i+1}(x) = d_i$ if $x \in R_i$, and $\nu_i \cdot d_i$ otherwise.

Lemma A.0.4. *Assertion on line 12 in Algorithm 3 holds.*

Proof. Let us consider the BADC $\mathcal{B} = (Q, Q_{\text{init}}, \Sigma, X, \delta, F)$ before Algorithm 3 is applied, denoted by $\mathcal{B}' = (Q, Q_{\text{init}}, \Sigma, X, \delta', F)$ and let us assume that $\mathcal{L}(\mathcal{B}') = \mathcal{L}(\mathcal{B}^{in})$. Let $\rho' = (q_0, \nu'_0)\tau'_0(q_1, \nu'_1)\tau'_1\dots$ be an extended run of $\mathcal{B}'$ over a word $w = (\sigma_0, d_0)(\sigma_1, d_1)\dots$. We show that there exists an extended run $\rho = (q_0, \nu_0)\tau_0(q_1, \nu_1)\tau_1\dots$ of $\mathcal{B}$ over w (and vice versa).

Note that if $q_i = q_x \in Q_x$ for some $i \geq 0$ and τ'_i bounds x, then, according to the definitions of $Paths, Q_s, Q_o$ and δ_o (lines 3, 4, 5, 6), there exists $q_k \in Q_s$, $k \leq i$, such that $k = 0$ or $x \in reset(\tau'_{k-1})$ and $\forall m \in \{k, \dots, i\}.\, \tau'_m \in \delta_o$.

Let $\tau'_i = (q_i, \sigma_i, \gamma, R, q_{i+1})$. We put

$$\tau_i = \begin{cases} \tau'_i, & \text{if } \tau'_i \in \delta_0 \\ (q_i, \sigma_i, \gamma, R \cup \{x\}, q_{i+1}) & \text{if } \tau'_i \in \delta_n \end{cases}$$

First of all, $\nu'_0(y) = \nu_0(y)$ for all $y \in X$. Let $\nu'_i(y) = \nu_i(y)$ for each $y \neq x \in X$ and $\big((q_i, \nu'_i), (\sigma_i, d_i), (q_{i+1}, \nu'_{i+1})\big) \in T'$ via τ'_i and each τ'_k, $k \leq i$ does not bound x in $\mathcal{B}'$. Then $\big((q_i, \nu_i), (\sigma_i, d_i), (q_{i+1}, \nu_{i+1})\big) \in T$ via τ_i in $\mathcal{B}$ and $\nu'_{i+1}(y) = \nu_{i+1}(y)$ for each $y \neq x \in X$. Furthermore, if $x \in reset(\tau'_i)$, then $\nu'_{i+1}(x) = \nu_{i+1}(x)$. If in addition $\nu'_i(x) = \nu_i(x)$ and $\tau'_i \in \delta_o$ we have $\nu'_{i+1}(x) = \nu_{i+1}(x)$.

Let τ'_i be the very first occurrence of a transition that bounds x on ρ'. Then necessarily there exists $q_k \in Q_s$, $k \leq i$, such that $k = 0$ or $x \in reset(\tau'_{k-1})$ and $\forall m \in \{k, \dots, i\}.\, \tau'_i \in \delta_o$. According to the previous, $\nu'_k(y) = \nu_k(y)$ and inductively $\nu'_i(y) = \nu_i(y)$ for all $y \in X$, especially for $\nu'_i(x) = \nu_i(x)$. Because $constraint(\tau'_i) = constraint(\tau_i)$, we have $\big((q_i, \nu_i), (\sigma_i, d_i), (q_{i+1}, \nu_{i+1})\big) \in T$ via τ_i in $\mathcal{B}$. Furthermore, $\nu'_{i+1}(y) = \nu_{i+1}(y)$ for each $y \neq x \in X$ and if $x \in reset(\tau'_i)$, then $\nu'_{i+1}(x) = \nu_{i+1}(x)$. If in addition $\tau'_i \in \delta_o$ we have $\nu'_{i+1}(x) = \nu_{i+1}(x)$.

Inductively, let us assume that $\nu'_i(y) = \nu_i(y)$ for each $y \neq x \in X$ and that $\big((q_i, \nu'_i), (\sigma_i, d_i), (q_{i+1}, \nu'_{i+1})\big) \in T'$ via τ'_i. If τ'_i does not bound x in $\mathcal{B}'$, then $\big((q_i, \nu_i), (\sigma_i, d_i), (q_{i+1}, \nu_{i+1})\big) \in T$ via τ_i in $\mathcal{B}$ and $\nu'_{i+1}(y) = \nu_{i+1}(y)$ for all $y \neq x \in X$. Furthermore, if $x \in reset(\tau'_i)$, then $\nu'_{i+1}(x) = \nu_{i+1}(x)$. If in addition $\nu'_i(x) = \nu_i(x)$ and $\tau'_i \in \delta_o$ we have $\nu'_{i+1}(x) = \nu_{i+1}(x)$. If τ'_i bounds x in $\mathcal{B}'$, then necessarily $\nu'_i(x) = \nu_i(x)$ and $\big((q_i, \nu_i), (\sigma_i, d_i), (q_{i+1}, \nu_{i+1})\big) \in T$ via τ_i in $\mathcal{B}$. Furthermore, $\nu'_{i+1}(y) = \nu_{i+1}(y)$ for each $y \neq x \in X$. Moreover, if $x \in reset(\tau'_i)$, then $\nu'_{i+1}(x) = \nu_{i+1}(x)$. If in addition $\tau'_i \in \delta_o$ we have $\nu'_{i+1}(x) = \nu_{i+1}(x)$.

All in all, if $\rho' = (q_0, \nu'_0)\tau'_0(q_1, \nu'_1)\tau'_1 \dots$ is an extended run of $\mathcal{B}'$ over w then there exists a run $\rho = (q_0, \nu_0)\tau_0(q_1, \nu_1)\tau_1 \dots$ of $\mathcal{B}$ over w. Obviously, if ρ' is an accepting run of $\mathcal{B}'$, then ρ is an accepting run of $\mathcal{B}$.

Dually, let $\rho = (q_0, \nu_0)\tau_0(q_1, \nu_1)\tau_1 \dots$ be an extended accepting run of $\mathcal{B}$ over word $w = (\sigma_0, d_0)(\sigma_1, d_1)(\sigma_2, d_2) \dots$. We show that there exists an extended accepting run $\rho' = (q_0, \nu'_0)\tau'_0(q_1, \nu'_1)\tau'_1 \dots$ of $\mathcal{B}'$ over w. Let

$$\tau'_i = \begin{cases} \tau_i, & \text{if } \tau_i = (q_i, \sigma_i, \gamma_i, R_i, q_{i+1}) \text{ and } \tau_i \in \delta_o \\ (q_i, \sigma_i, \gamma_i, R_i, q_{i+1}), & \text{if } \tau_i = (q_i, \sigma_i, \gamma_i, R_i \cup \{x\}, q_{i+1}) \text{ and } \tau_i \in \delta_n \end{cases}$$

The details of the proof are analogous as for the first part. □

Lemma A.0.5. *Assertion on line 24 in Algorithm 4 holds.*

Proof. Let us consider the BADC $\mathcal{B} = (Q, Q_{\text{init}}, \Sigma, X, \delta, F)$ before Algorithm 3 is applied, denoted by $\mathcal{B}' = (Q, Q_{\text{init}}, \Sigma, X, \delta', F)$ and let us assume that $\mathcal{L}(\mathcal{B}') = \mathcal{L}(\mathcal{B}^{in})$. Let $\rho' = (q_0, \nu'_0)\tau'_0(q_1, \nu'_1)\tau'_1 \dots$ be an extended run of $\mathcal{B}'$ over a word $w = (\sigma_0, d_0)(\sigma_1, d_1) \dots$. We show that there exists an extended run $\rho = (q_0, \nu_0)\tau_0(q_1, \nu_1)\tau_1 \dots$ of $\mathcal{B}$ over w (and vice versa). Throughout the proof we assume that $\nu_i(y) = \nu'_i(y)$ for all $y \neq x \in X$, which is obvious. Let

$$q_i = \begin{cases} q'_i & \text{if } q'_i \notin Q_o \\ \breve{q}'_i & \text{if } q'_i \in Q_o \text{ and } x \notin reset(\tau'_{i-1}), i \neq 0 \text{ and } \nu'_{i-1}(x) \text{ satisfies } lb(x) \\ \widehat{q}'_i & \text{if } q'_i \in Q_o \text{ and } x \notin reset(\tau'_{i-1}), i \neq 0 \\ & \text{and } \nu'_{i-1}(x) \text{ does not satisfy } lb(x). \text{ Then also } \nu'_i(x) = \nu_i(x). \\ \ddot{q}'_i & \text{if } q'_i \in Q_o \text{ and } x \in reset(\tau'_{i-1}) \text{ or } i = 0. \end{cases}$$

Note that q'_i belongs to exactly one of the categories described above, *i.e.,* we define q_i unambiguously for each possible q'_i.

First of all, $q_0 = \ddot{q}'_0$ if $q'_0 \in Q_o$ and $q_0 = q'_0$ otherwise. Let us assume that $\big((q'_i, \nu_i), (\sigma_i, d_i), (q'_{i+1}, \nu_{i+1})\big) \in T'$ via $\tau'_i = (q'_i, \sigma, \gamma', R', q'_{i+1})$. We show that

$\big((q_i,\nu_i),(\sigma_i,d_i),(q_{i+1},\nu_{i+1})\big) \in T$ via transition $\tau_i = (q_i,\sigma,\gamma,R,q_{i+1})$. Particularly, we show case by case that such τ_i exists.

(i) Let $q'_i \notin Q_o$, i.e. $q_i = q'_i$:

a) $q'_{i+1} \notin Q_o$. Then $\tau_i = \tau'_i$ and $q_{i+1} = q'_{i+1}$.

b) $q'_{i+1} \in Q_o$. Then necessarily that $x \in reset(\tau'_i)$ and $\tau'_i \neq \tau_x$, according to the definition of Q_o. Then $q_{i+1} = \ddot{q}'_{i+1}$ and $\tau_i = (q'_i,\sigma,\gamma,R,\ddot{q}'_{i+1})$ (line 7).

(ii) Let $q'_i \in Q_o$ and $x \notin reset(\tau_{i-1'}), i \neq 0$ and $\nu'_{i-1}(x)$ satisfies $lb(x)$, *i.e.,* $q_i = \breve{q}'_i$:

a) $q'_{i+1} \notin Q_o$. Then $q_{i+1} = q'_{i+1}$. If $\tau_i \neq \tau_x$ then $\tau_i = \tau'_i$ (line 12). If $\tau_i = \tau_x$ then $\nu'_i(x)$ satisfies $lb(x)$, thus $\bowtie \in \{<,\leq\}$. $\tau_i = (\breve{q}'_i,\sigma,\gamma,R\cup\{x\},q'_{i+1})$ (line 18).

b) $q'_{i+1} \in Q_o, x \notin reset(\tau'_i)$ and $\nu_i(x)'$ satisfies $lb(x)$. Then $q_{i+1} = \breve{q}'_{i+1}$. If $\tau'_i \neq \tau_x$ then $\tau_i = (\breve{q}'_i,\sigma,\gamma,R\cup\{x\},\breve{q}'_{i+1})$ (line 10). If $\tau'_i = \tau_x$ then $\tau_i = (\breve{q}'_i,\sigma,\gamma,R\cup\{x\},\breve{q}'_{i+1})$ (line 18).

c) $q'_{i+1} \in Q_o, x \in reset(\tau'_i)$ and $\nu_i(x)'$ does not satisfy $lb(x)$. This situation cannot happen, because $\nu_{i-1}(x)$ satisfies $lb(x)$ and $x \notin reset(\tau_{x-1})$, thus $\nu_i(x)$ satisfies $lb(x)$.

d) $q'_{i+1} \in Q_o, x \in reset(\tau'_i)$. Then $q_{i+1} = \ddot{q}'_{i+1}$. If $\tau'_i \neq \tau_x$ then transition $\tau_i = (\breve{q}'_i,\sigma,\gamma,R,\ddot{q}'_{i+1})$ (line 8). If $\tau'_i = \tau_x$ then $\nu_i(x)$ satisfies $lb(x)$, because $\nu_{i-1}(x)$ satisfies $lb(x)$ and $x \notin reset(\tau_{x-1})$. Thus $\bowtie \in \{<,\leq\}$. Altogether $\tau_i = (\breve{q}'_i,\sigma,\gamma,R\cup\{x\},\ddot{q}'_{i+1})$ (line 18).

(iii) Let $q'_i \in Q_o$ and $x \notin reset(\tau_{i-1}), i \neq 0$ and $\nu'_{i-1}(x)$ does not satisfy $lb(x)$ and also $\nu'_i(x) = \nu_i(x)$, *i.e.,* $q_i = \widehat{q}'_i$

a) $q'_{i+1} \notin Q_o$. Then $q_{i+1} = q'_{i+1}$. If $\tau_i \neq \tau_x$ then $\tau_i = (\widehat{q}'_i,\sigma,\gamma,R,q'_{i+1})$ (line 12). If $\tau_i = \tau_x \wedge \nu'_i(x)$ satisfies $lb(x)$ then $\bowtie \in \{<,\leq\}$. $\nu'_i(x) = \nu_i(x)$, therefore $\nu_i(x)$ satisfies $lb(x)$. $\tau_i = (\widehat{q}'_i,\sigma,\gamma\wedge lb(x),R\cup\{x\},q'_{i+1})$ (line 18). If $\tau_i = \tau_x \wedge \nu'_i(x)$ does not satisfy $lb(x)$ then $\bowtie \in \{>,\geq\}$. $\nu'_i(x) = \nu_i(x)$, therefore $\nu_i(x)$ does not satisfy $lb(x)$. $\tau_i = (\widehat{q}'_i,\sigma,\gamma\wedge\neg lb(x),R,q'_{i+1})$ (line 21).

b) $q'_{i+1} \in Q_o, x \notin reset(\tau'_i)$ and $\nu_i(x)'$ satisfies $lb(x)$. Then $q_{i+1} = \breve{q}'_{i+1}$. If $\tau'_i \neq \tau_x$ then $\tau_i = (\widehat{q}'_i,\sigma,\gamma\wedge lb(x),R\cup\{x\},\breve{q}'_{i+1})$ (line 10). If $\tau'_i = \tau_x$ then $\tau_i = (\widehat{q}'_i,\sigma,\gamma\wedge lb(x),R\cup\{x\},\breve{q}'_{i+1})$ (line 18).

c) $q'_{i+1} \in Q_o, x \in reset(\tau'_i)$ and $\nu_i(x)'$ does not satisfy $lb(x)$, *i.e.,* $q_{i+1} = \widehat{q}'_{i+1}$. Note that because $x \notin reset(\tau'_i)$ and $\nu_i(x) = \nu_i(x)'$, then also $\nu_{i+1}(x) = \nu_{i+1}(x)'$. If $\tau'_i \neq \tau_x$ then $\tau_i = (\widehat{q}'_i,\sigma,\gamma\wedge\neg lb(x),R,\widehat{q}'_{i+1})$ (line 10). If $\tau'_i = \tau_x$ then $\tau_i = (\widehat{q}'_i,\sigma,\gamma\wedge\neg lb(x),R,\widehat{q}'_{i+1})$ (line 21).

d) $q'_{i+1} \in Q_o, x \in reset(\tau'_i)$, i.e. $q_{i+1} = \ddot{q}'_{i+1}$. If $\tau'_i \neq \tau_x$ then $\tau_i = (\widehat{q}'_i,\sigma,\gamma,R,$ $\ddot{q}'_{i+1})$ (line 8). If $\tau'_i = \tau_x$ and $\nu_i(x)'$ satisfies $lb(x)$ then $\tau_i = (\widehat{q}'_i,\sigma,\gamma\wedge$

$lb(x), R \cup \{x\}, \ddot{q}'_{i+1}$) (line 18). If $\tau'_i = \tau_x$ and $\nu_i(x)'$ does not satisfy $lb(x)$ then $\tau_i = (\widehat{q}'_i, \sigma, \gamma \wedge \neg lb(x), R, \ddot{q}'_{i+1})$ (line 21).

(iv) $x \in reset(\tau_{i-1})$ or $i = 0$, *i.e.,* $q_i = \ddot{q}'_i$. In such case, also $\nu_i(x) = \nu_i(x)'$.

a) $q'_{i+1} \notin Q_o$. Then $q_{i+1} = q'_{i+1}$. If $\tau_i \neq \tau_x$ then $\tau_i = (\ddot{q}'_i, a, \gamma, R, q'_{i+1})$ (line 13). If $\tau_i = \tau_x$ and $\nu'_i(x)$ satisfies $lb(x)$ then $\bowtie \in \{<, \leq\}$. $\nu'_i(x) = \nu_i(x)$, therefore $\nu_i(x)$ satisfies $lb(x)$. $\tau_i = (\ddot{q}'_i, \sigma, \gamma \wedge lb(x), R \cup \{x\}, q'_{i+1})$ (line 18). If $\tau_i = \tau_x$ and $\nu'_i(x)$ does not satisfy $lb(x)$ then $\bowtie \in \{>, \geq\}$. $\nu'_i(x) = \nu_i(x)$, therefore $\nu_i(x)$ does not satisfy $lb(x)$. $\tau_i = (\ddot{q}'_i, \sigma, \gamma \wedge \neg lb(x), R, q'_{i+1})$ (line 21).

b) $q'_{i+1} \in Q_o, x \notin reset(\tau'_i)$ and $\nu_i(x)'$ satisfies $lb(x)$. Then $q_{i+1} = \breve{q}'_{i+1}$. If $\tau'_i \neq \tau_x$ then $\tau_i = (\ddot{q}'_i, \sigma, \gamma \wedge lb(x), R \cup \{x\}, \breve{q}'_{i+1})$ (line 11). If $\tau'_i = \tau_x$ then $\tau_i = (\ddot{q}'_i, \sigma, \gamma \wedge g(x), R \cup \{x\}, \breve{q}'_{i+1})$ (line 18).

c) $q'_{i+1} \in Q_o, x \in reset(\tau'_i)$ and $\nu_i(x)'$ does not satisfy $lb(x)$, i.e. $q_{i+1} = \widehat{q}'_{i+1}$. Note that because $x \notin reset(\tau'_i)$ and $\nu_i(x) = \nu_i(x)'$, then also $\nu_{i+1}(x) = \nu_{i+1}(x)'$. If $\tau'_i \neq \tau_x$ then $\tau_i = (\ddot{q}'_i, \sigma, \gamma \wedge \neg lb(x), R, \widehat{q}'_{i+1})$ (line 10). If $\tau'_i = \tau_x$ then $\tau_i = (\ddot{q}'_i, \sigma, \gamma \wedge \neg lb(x), R, \widehat{q}'_{i+1})$ (line 21).

d) $q'_{i+1} \in Q_o, x \in reset(\tau'_i)$, i.e. $q_{i+1} = \ddot{q}'_{i+1}$. If $\tau'_i \neq \tau_x$ then $\tau_i = (\ddot{q}'_i, \sigma, \gamma, R, \ddot{q}'_{i+1})$ (line 9). If $\tau'_i = \tau_x$ and $\nu_i(x)'$ satisfies $lb(x)$ then $\tau_i = (\ddot{q}'_i, \sigma, \gamma \wedge lb(x), R \cup \{x\}, \ddot{q}'_{i+1})$ (line 18). If $\tau'_i = \tau_x$ and $\nu_i(x)'$ does not satisfy $lb(x)$ then $\tau_i = (\ddot{q}'_i, \sigma, \gamma \wedge \neg lb(x), R, \ddot{q}'_{i+1})$ (line 21).

Altogether if $\big((q'_i, \nu_i), (\sigma_i, d_i), (q'_{i+1}, \nu_{i+1})\big) \in T'$ via $\tau'_i = (q'_i, \sigma, \gamma', R', q'_{i+1})$ then $\big((q_i, \nu_i), (\sigma_i, d_i), (q_{i+1}, \nu_{i+1})\big) \in T$ via transition $\tau_i = (q_i, \sigma, \gamma, R, q_{i+1})$ for all $i \geq 0$, i.e. for each extended run $\rho' = (q'_0, \nu'_0)\tau'_0(q'_1, \nu'_1)\tau'_1 \ldots$ of $\mathcal{B}'$ over word $w = (\sigma_0, d_0)(\sigma_1, d_1) \ldots$ in $\mathcal{S}(\mathcal{B}')$ there exists an extended run $\rho = (q_0, \nu_0)\tau_0(q_1, \nu_1)\tau_1 \ldots$ of $\mathcal{B}$ over σ. Obviously, if ρ' is accepting, ρ is accepting as well.

Let, the other way around, $\rho = (q_0, \nu_0)\tau_0(q_1, \nu_1)\tau_1 \ldots$ be an extended run of $\mathcal{B}$ over $w = (\sigma_0, d_0)(\sigma_1, d_1) \ldots$. We show that there exists an extended run $\rho' = (q'_0, \nu'_0)\tau'_0(q'_1, \nu'_1)\tau'_1 \ldots$ of $\mathcal{B}'$ over w.

Let $q'_i = q$ if $q_i \in \{q, \ddot{q}, \breve{q}, \widehat{q}\}$, and assume that

- If $q_i = q$, then $q \notin Q_o$
- If $q_i = \ddot{q}$, then $q \in Q_o$, $x \in reset(\tau'_{i-1})$ or $i = 0$. Then also $\nu_i(x) = \nu'_i(x)$.
- If $q_i = \breve{q}$, then $q \in Q_o$, $x \notin reset(\tau'_{i-1}), i \neq 0$ and $\nu'_{i-1}(x)$ satisfies $lb(x)$.
- If $q_i = \widehat{q}$, then $q \in Q_o$, $x \notin reset(\tau'_{i-1}), i \neq 0$ and $\nu'_{i-1}(x)$ does not satisfy $lb(x)$. Then also $\nu_i(x) = \nu'_i(x)$.

First of all, $q_0 = \ddot{q}_0$ in case $q_0 \in Q_o$, and $q_0 = q_0$ otherwise, for all $q_0 \in Q_{\text{init}}$. In any case, $\nu'_0(x) = \nu_0(x)$. Let $\big((q_i, \nu_i), (\sigma_i, d_i), (q_{i+1}, \nu_{i+1})\big) \in T$ via

$\tau_i = (q_i, \sigma, \gamma, R, q_{i+1})$. We show that $\big((q_i, \nu'_i), (\sigma_i, d_i), (q_{i+1}, \nu'_{i+1})\big) \in T'$ via $\tau'_i = (q'_i, \sigma, \gamma', R', q'_{i+1})$ case by case:

(i) Let $q_i = q_1$ and $q_1 \notin Q_o$.

a) $q_{i+1} = \ddot{q}_2$. Then τ_i is added to T on line 7, $\tau_i = (q_1, \sigma, \gamma, R, \ddot{q}_2)$ and transition τ'_i is $(q_1, \sigma, \gamma, R, q_2)$, where $q_2 \in Q_o$, $x \in reset(\tau'_i)$ and $\nu_{i+1}(x) = \nu'_{i+1}(x)$.

b) $q_{i+1} = q_2$. Then $q_2 \notin Q_o$. $\tau_i = \tau'_i = (q_1, \sigma, \gamma, R, q_2)$.

c) q_{i+1} cannot be $\widehat{q}_2$ or $\breve{q}_2$ for any q_2.

(ii) Let $q_i = \ddot{q}_1$, $i = 0$ or $q_1 \in Q_o$, $x \in reset(\tau'_{i-1})$ and $\nu_i(x) = \nu'_i(x)$.

a) $q_{i+1} = q_2$. Then τ_i is to T on line 13, 18 or 21. In all three cases $q_2 \notin Q_o$ and $\tau'_i = (q_1, \sigma, \gamma, R, q_2)$.

b) $q_{i+1} = \ddot{q}_2$. Then τ_i is added to T on line 9,18 or 21. If $\tau_i = (\ddot{q}_1, \sigma, \gamma, R, \ddot{q}_2)$ is a transition added to T on line 9, then $q_2 \in Q_o$ and $x \in R$ and $\tau'_i = (q_1, \sigma, \gamma, R, q_2)$. If added on line 18, then $\tau_i = (\ddot{q}_1, \sigma, \gamma \wedge lb(x), R \cup \{x\}, \ddot{q}_2)$. Therefore $\nu_i(x) = \nu'_i(x)$ satisfies $lb(x)$. Then $\nu'_i(x)$ satisfies γ, because $\bowtie \in \{<, \leq\}$ and τ'_i is $\tau_x = (q_1, \sigma, \gamma, R, q_2)$. If added on line 21, then $\tau_i = (\ddot{q}_1, \sigma, \gamma \wedge \neg lb(x), R, \ddot{q}_2)$. Therefore $\nu_i(x) = \nu'_i(x)$ does not satisfy $lb(x)$. Then $\nu'_i(x)$ satisfies γ because $\bowtie \in \{>, \geq\}$ and τ'_i is $\tau_x = (q_1, \sigma, \gamma, R, q_2)$. Together, in all the three cases $q_2 \in Q_o$, $x \in reset(\tau'_i)$, $\nu_{i+1}(x) = \nu'_{i+1}(x)$ and $\tau'_i = (q_1, \sigma, \gamma, R, q_2)$.

c) $q_{i+1} = \breve{q}_2$. Then τ_i is added to T on line 11 or 18. If $\tau_i = (\ddot{q}_1, \sigma, \gamma \wedge lb(x), R \cup \{x\}, q_2)$ is a transition added to T on line 11, then $q_2 \in Q_o$ and $x \notin R$. Transition τ'_i is $(q_1, \sigma, \gamma, R, q_2)$. Also $\nu_i(x) = \nu'_i(x)$ satisfies $lb(x)$. If $\tau_i = (\ddot{q}_1, \sigma, \gamma \wedge lb(x), R \cup \{x\}, q_2)$ is added on line 18, $q_2 \in Q_o$ and $x \notin R$. $\nu_i(x) = \nu'_i(x)$ satisfies $lb(x)$. Then $\nu'_i(x)$ satisfies γ because $\bowtie \in \{<, \leq\}$ and τ'_i is $\tau_x = (q_1, \sigma, \gamma, R, q_2)$. Together, in both cases $q_2 \in Q_o, x \notin reset(\tau'_i)$, $\nu'_i(x)$ satisfies $lb(x)$ and $\tau'_i = (q_1, \sigma, \gamma, R, q_2)$.

d) $q_{i+1} = \widehat{q}_2$. Then τ_i is added to T on line 11 or 21. If $\tau_i = (\ddot{q}_1, \sigma, \gamma \wedge \neg lb(x), R, \widehat{q}_2)$ is a transition added to T on line 11, then $q_2 \in Q_o$ and $x \notin R$. Transition τ'_i is $(q_1, \sigma, \gamma, R, q_2)$. $\nu_i(x) = \nu'_i(x)$ satisfies $\neg lb(x)$. Also $\nu_{i+1}(x) = \nu'_{i+1}(x)$. If $\tau_i = (\ddot{q}_1, \sigma, \gamma \wedge \neg lb(x), R, q_2)$ is added on line 21, $q_2 \in Q_o$ and $x \notin R$. $\nu_i(x) = \nu'_i(x)$ satisfies $\neg lb(x)$. Then $\nu'_i(x)$ satisfies γ because $\bowtie \in \{>, \geq\}$ and τ'_i is $\tau_x = (q_1, \sigma, \gamma, R, q_2)$. Also $\nu_{i+1}(x) = \nu'_{i+1}(x)$. Together, in both cases $q_2 \in Q_o, x \notin R$, $\nu'_i(x)$ does not satisfy $lb(x)$, $\nu_{i+1}(x) = \nu'_{i+1}(x)$ and $\tau'_i = (q_1, \sigma, \gamma, R, q_2)$

(iii) Let $q_i = \breve{q}_1$, $i \neq 0$, $q_1 \in Q_o$, $x \notin reset(\tau'_{i-1})$ and $\nu'_{i-1}(x)$ satisfies $lb(x)$.

a) $q_{i+1} = q_2$. Then τ_i is added to T on line 12 or 18. In both cases $q_2 \notin Q_o$ and $\tau'_i = (q_1, \sigma, \gamma, R, q_2)$.

b) $q_{i+1} = \ddot{q}_2$. Then τ_i is added to T on line 8 or 18. If $\tau_i = (q_1, \sigma, \gamma, R, \ddot{q}_2)$ is a transition added to T on line 8, transition τ'_i is $(q_1, \sigma, \gamma, R, q_2) \neq \tau_x$, where $q_2 \in Q_o$, $x \in reset(\tau'_i)$ and $\nu_{i+1}(x) = \nu_{i+1}(x)'$. If added on line 18, $\tau_i = (q_1, \sigma, \gamma, R \cup \{x\}, \ddot{q}_2)$, then $q_2 \in Q_o$, and $x \in R$. Because $\nu'_{i-1}(x)$ satisfies $lb(x)$ and $x \notin reset(\tau_{i-1})$, then also $\nu'_i(x)$ satisfies $lb(x)$ and $\nu'_i(x)$ satisfies γ because $\bowtie \in \{<, \leq\}$. Transition τ'_i is $\tau_x = (q_1, \sigma, \gamma, R, q_2)$. Together, in both cases $x \in reset(\tau'_i)$, $q_2 \in Q_o$, $\nu_{i+1}(x) = \nu'_{i+1}(x)$ and $\tau'_i = (q_1, \sigma, \gamma, R, q_2)$.

c) $q_{i+1} = \breve{q}_2$. Then τ_i is added to T on line 10 or 18. If $\tau_i = (q_1, \sigma, \gamma, R \cup \{x\}, q_2)$ is a transition added to T on line 10, then $q_2 \in Q_o$ and $x \notin R$. Transition τ'_i is $(q_1, \sigma, \gamma, R, q_2)$. $\nu'_i(x)$ satisfies $lb(x)$, because $\nu'_{i-1}(x)$ satisfies $lb(x)$ and $x \notin reset(\tau'_{i-1})$. If τ_i is added on line 18, $q_2 \notin Q_o$ or $x \notin R$. Because $\nu'_{i-1}(x)$ satisfies $lb(x)$ and $x \notin reset(\tau'_{i-1})$, then also $\nu'_i(x)$ satisfies $lb(x)$ and $\nu'_i(x)$ satisfies γ because $\bowtie \in \{<, \leq\}$. Then τ'_i is $\tau_x(q_1, \sigma, \gamma, R, q_2)$. Together, in both cases $q_2 \in Q_o, x \notin reset(\tau'_i)$, $\nu'_i(x)$ satisfies $lb(x)$ and $\tau'_i = (q_1, \sigma, \gamma, R, q_2)$.

d) q_{i+1} cannot be $\widehat{q_2}$ for any q_2.

(iv) Let $q_i = \widehat{q_1}$, $i \neq 0$, $q_1 \in Q_o$, $x \notin reset(\tau'_{i-1})$, $\nu'_{i-1}(x)$ does not satisfy $lb(x)$ and $\nu_i(x) = \nu_i(x)'$.

a) $q_{i+1} = q_2$. Then τ_i is added to T on line 12,18 or 21. In all three cases $q_2 \notin Q_o$, $\tau'_i = (q_1, \sigma, \gamma, R, q_2)$..

b) $q_{i+1} = \ddot{q}_2$. Then τ_i is added to T on line 8, 18 or 21. If $\tau_i = (\widehat{q_1}, \sigma, \gamma, R, \ddot{q}_2)$ is a transition added to T on line 8, then $q_2 \in Q_o$ and $x \in reset(\tau'_i)$. Transition τ'_i is $(q_1, \sigma, \gamma, R, q_2) \neq \tau_x$ and $\nu_{i+1}(x) = \nu_{i+1}(x)'$. If added on line 18, then $\tau_i = (\widehat{q_1}, \sigma, \gamma \wedge lb(x), R \cup \{x\}, \ddot{q}_2)$, $q_2 \in Q_o$ and $x \in reset(\tau'_i)$. $\nu_i(x) = \nu_i(x)'$ satisfies $lb(x)$ and $\nu'_i(x)$ satisfies γ because $\bowtie \in \{<, \leq\}$. Transition τ'_i is $\tau_x = (q_1, \sigma, \gamma, R, q_2)$. If added on line 21, then $\tau_i = (\widehat{q_1}, \sigma, \gamma \wedge \neg lb(x), R, \ddot{q}_2)$, $q_2 \in Q_o$ and $x \in R$. $\nu_i(x) = \nu_i(x)'$ satisfies $\neg lb(x)$ and $\nu'_i(x)$ satisfies γ because $\bowtie \in \{>, \geq\}$. Transition τ'_i is $\tau_x = (q_1, \sigma, \gamma, R, q_2)$. Together, in all the three cases $x \in reset(\tau'_i)$, $q_2 \in Q_o$, $\nu_{i+1}(x) = \nu'_{i+1}(x)$ and $\tau'_i = (q_1, \sigma, \gamma, R, q_2)$.

c) $q_{i+1} = \breve{q}_2$. Then τ_i is added to T on line 10 or 18. If $\tau_i = (\widehat{q_1}, \sigma, \gamma \wedge lb(x), R \cup \{x\}, q_2)$ is a transition added to T on line 10, then $q_2 \in Q_o$ and $x \notin R$. Transition τ'_i is $(q_1, \sigma, \gamma, R, q_2)$. $\nu_i(x) = \nu'_i(x)$ satisfies $lb(x)$. If $\tau_i = (\widehat{q_1}, \sigma, \gamma \wedge lb(x), R \cup \{x\}, q_2)$ is added on line 18, $q_2 \in Q_o$ and $x \notin R$. $\nu_i(x) = \nu'_i(x)$ satisfies $lb(x)$. Then also $\nu'_i(x)$ satisfies γ because

$\bowtie \in \{<, \leq\}$ and transition τ_i' is $(q_1, \sigma, \gamma, R, q_2) \neq \tau_x$. Together, in both cases $q_2 \in Q_o, x \notin reset(\tau_i')$, $\nu_i'(x)$ satisfies $lb(x)$ and $\tau_i' = (q_1, \sigma, \gamma, R, q_2)$.

d) $q_{i+1} = \widehat{q_2}$. Then τ_i is added to T on line 10 or 21. If $\tau_i = (\widehat{q_1}, \sigma, \gamma \wedge \neg lb(x), R, \widehat{q_2})$ is a transition added to T on line 10, then $q_2 \in Q_o$ and $x \notin R$. $\nu_i(x) = \nu_i'(x)$ satisfies $\neg lb(x)$. Transition τ_i' is $(q_1, \sigma, \gamma, R, q_2) \neq \tau_x$. Also $\nu_{i+1}(x) = \nu_{i+1}'(x)$. If $\tau_i = (\widehat{q_1}, \sigma, \gamma \wedge \neg lb(x), R, \widehat{q_2})$ is added on line 21, $q_2 \in Q_o$ and $x \notin R$. $\nu_i(x) = \nu_i'(x)$ satisfies $\neg lb(x)$. Then also $\nu_i'(x)$ satisfies γ because $\bowtie \in \{>, \geq\}$ and transit on τ_i' is $\tau_x = (q_1, \sigma, \gamma, R, q_2)$. Also $\nu_{i+1}(x) = \nu_{i+1}'(x)$. Together, in both cases $q_2 \in Q_o, x \notin R$, $\nu_i'(x)$ does not satisfy $lb(x)$, $\nu_{i+1}(x) = \nu_{i+1}'(x)$ and $\tau_i' = (q_1, \sigma, \gamma, R, q_2)$.

We have shown that for each extended run $\rho' = (q_0', \nu_0')\tau_0'(q_1', \nu_1')\tau_1' \ldots$ over w in $S(\mathcal{B}')$ there exists an extended run $\rho = (q_0, \nu_0)\tau_0(q_1, \nu_1)\tau_1 \ldots$ over w in $\mathcal{S}(\mathcal{B})$ and vice versa. Obviously, if ρ' is accepting, then ρ is accepting as well and the other way around. □

Lemma A.0.6. *Assertion on line 11 in Algorithm 1 holds.*

Proof. Follows directly from lemmas A.0.2, A.0.3, A.0.4. □

Lemma A.0.7. *Assertion on line 9 in Algorithm 1 holds.*

Proof. Inductively. For $Done = \emptyset$ holds trivially. Let us consider the iteration of the cycle 4-9 in Algorithm 1 for the variable x. For each $\tau \in \delta_n$ we have $x \in reset(\tau)$. Furthermore, each transition $\tau \in \delta_o$ is processed in procedure split_into_cases(x) such that it is replaced with transitions of form τ', $x \in reset(\tau')$ or $constraint(\tau')$ contains bound $x \bowtie c$, where $\bowtie \in \{>, \geq\}$. This means that for *each* transition $\tau \in \delta$ on line 9 in Algorithm 1 it holds that $constraint(\tau)$ contains bound of form $x \bowtie c$, where $\bowtie \in \{>, \geq\}$, or $x \in reset(\tau)$. The iteration of the cycle 4-9 in 1 for x does not change any resets or constrains $y \neq x \in X$. Hence, each $x \in Done$ is in $\mathcal{B}$ in normal form. □

Lemma A.0.8. *Assertion on line 12 in Algorithm 1 holds.*

Proof. Follows directly from Lemma A.0.7. □

Appendix B

Author's Contribution

Journal Papers

[CTUB13] Yushan Chen, Jana Tůmová, Alphan Ulusoy, Calin Belta. Temporal Logic Robot Control based on Automata Learning of Environmental Dynamics. *The International Journal of Robotics Research*, 2013. In print.

Contribution to: algorithms for learning door behavior, definition of the surveillance LTL fragment, writing (Sec. 3, 4). *Minor contribution to:* problem formulation, writing (Sec. 2).

[YTČ+13] Boyan Yordanov, Jana Tůmová, Ivana Černá, Jiří Barnat, Calin Belta. Formal Analysis of Piecewise Affine Systems through Formula-Guided Refinement. *Automatica*, 49(1):261 – 266, 2013.

Contribution to: graph algorithms for the product automaton, implementation of the graph algorithms. *Minor contribution to:* problem formulation, writing (Sec. 2, 4).

[BČT12b] Jiří Barnat, Ivana Černá, Jana Tůmová. Verification of Systems with Degradation. *Computing and Informatics*, 31(3):507-530, 2012.

Leading author. Contribution to: problem formulation and solution, definition of DLTL, algorithms for timed automata approach to DLTL model checking, algorithms for DLTL to BADC translation, writing (Sec. 1-6).

[YTČ+12] Boyan Yordanov, Jana Tůmová, Ivana Černá, Jiří Barnat, and Calin Belta. Temporal Logic Control of Discrete-Time Piecewise Affine Systems. *IEEE Transactions on Automatic Control*, 57(6):1491 –1504, 2012.

Contribution to: algorithms for Rabin games with stuttering, implementation of the graph algorithms, writing (Sec. 3, 7.B). *Minor contribution to:* problem formulation, writing (Sec. 2, 8).

[STBR11] Stephen L. Smith, Jana Tůmová, Calin Belta, Daniela Rus. Optimal Path Planning for Surveillance with Temporal Logic Constraints. *The International Journal of Robotics Research*, 30(14):1695–1708, 2011.

Contribution to: problem solution, case studies and experimental measurements, writing (Sec. 4.1, 5). *Minor contribution to:* problem formulation, writing (Sec. 2, 3, 4.3).

Conference and Workshop Proceedings

[TRCK+12] Jana Tůmová, Luis I. Reyes-Castro, Sertac Karaman, Emilio Frazzoli and Daniela Rus. Minimum-violating Planning with Conflicting Specifications. 2013. (Submitted).

Leading author. Contribution to: problem formulation, problem solution, rescue mission example, writing (Sec. 1-6).

[THK+12] Jana Tůmová, Gavin C.Hall, Sertac Karaman, Emilio Frazzoli and Daniela Rus. Least-violating Control Strategy Synthesis with Safety Rules. In *Hybrid Systems: Computation and Control (HSCC)*, 2013. (To appear).

Leading author. Contribution to: problem formulation, problem solution, the set of case study examples, writing (Sec. 1-6).

[STBČ12] Mária Svoreňová, Jana Tůmová, Jiří Barnat, and Ivana Černá. Attraction-Based Receding Horizon Path Planning with Temporal Logic Constraints. In *IEEE Conference on Decision and Control (CDC)*, pages 6749–6754, 2012.

Contribution to: problem formulation. *Minor contribution to:* problem solution, writing (Sec. 1-5).

[CTB12] Yushan Chen, Jana Tůmová, Calin Belta. LTL Robot Motion Control based on Automata Learning of Environmental Dynamics. *In IEEE International Conference on Robotics and Automation (ICRA)*, pages 5177–5182, 2012.

Contribution to: algorithms for learning door behavior, definition of the survellance LTL fragment, writing (Sec. 2.B, 3). *Minor contribution to:* problem formulation.

[BČT12a] Jiří Barnat, Ivana Černá, Jana Tůmová. Timed Automata Approach to Verification of Systems with Degradation. In *Annual Doctoral Workshop on Mathematical and Engineering Methods in Computer Science (MEMICS)*, volume LNCS 7119, pages 86–95, 2011. **Best paper award.**

Leading author. Contribution to: problem formulation and solution, definition of DLTL, algorithms for timed automata approach to DLTL model checking, writing (Sec. 1-5).

[TYB+10] Jana Tůmová, Boyan Yordanov, Calin Belta, Ivana Černá, and Jiří Barnat. A Symbolic Approach to Controlling Piecewise Affine Systems. In *IEEE Conference on Decision and Control (CDC)*, pages 4230 –4235, 2010.

Leading author. Contribution to: algorithms for Rabin games with stuttering, implementation of the graph algorithms, writing (Sec. 5, 6). *Minor contribution to:* problem formulation, writing (Sec. 7).

[YTB+10] Boyan Yordanov, Jana Tůmová, Calin Belta, Ivana Černá, and Jiří Barnat. *Formal Analysis of Piecewise Affine Systems through Formula-Guided Refinement.* In *IEEE Conference on Decision and Control (CDC)*, pages 5899–5904, 2010.

Contribution to: graph algorithms for the product automaton, implementation of the graph algorithms, writing (Sec. 5). *Minor contribution to:* problem formulation, writing (Sec. 2, 4).

[STBR10] Stephen L. Smith, Jana Tůmová, Calin Belta, and Daniela Rus. Optimal Path Planning under Temporal Logic Contstraints. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pages 3288–3293, 2010.

Contribution to: problem solution, case studies and experimental measurements, writing (Sec. 4.A, 5). *Minor contribution to:* problem formulation, writing (Sec. 2, 3).

[BČT09] Jiří Barnat, Ivana Černá, and Jana Tůmová. Quantitative Model Checking of Systems with Degradation. In *International Conference on Quantitative Evaluation of SysTems (QEST)*, pages 21 –30, 2009.

Leading author. Contribution to: problem formulation and solution, definition of BADC, normalization and model checking algorithms, writing (Sec. 2-6).

[BBČ+09] Jiří Barnat, Luboš Brim, Ivana Černá, Milan Češka and Jana Tůmová. *Local Quantitative LTL Model Checking.* In *International Workshop on Formal Methods for Industrial Critical Systems (FMICS)*, pages 63–78, 2008.

Contribution to: experimental evaluation. *Minor contribution to:* algorithms, writing (Sec. 3).

Conference Tool Papers

[BBČ+08] Jiří Barnat, Luboš Brim, Ivana Černá, Milan Češka and Jana Tůmová. ProbDiVinE-MC: Multi-core LTL Model Checker for Probabilistic Sys-

tems. In *International Conference on Quantitative Evaluation of Systems (QEST)*, pages 77–78, 2008.

Contribution to: experimental evaluation.

[BBČ+07] Jiří Barnat, Luboš Brim, Ivana Černá, Milan Češka and Jana Tůmová. ProbDiVinE: A Parallel Qualitative LTL Model Checker. In *International Conference on Quantitative Evaluation of Systems (QEST)*, pages 215–216, 2007.

Contribution to: implementation.